Dipartimento di Filosofia,
Comunicazione e Spettacolo (Fil.Co.Spe.)

Dottorato in Filosofia e
Teoria delle Scienze Umane,
XXVII ciclo

Institut de Mathématiques de Marseille (I2M)

*École Doctorale en Mathématiques
et Informatique de Marseille - ED 184
UFR Sciences*

tesi in cotutela / *thèse en cotutelle*

Paolo Pistone

On Proofs and Types in Second Order Logic

Relatori / *Rapporteurs*:

Joinet	Jean-Baptiste	Université Jean Moulin Lyon 3
Longo	Giuseppe	Ecole Normale Supérieure / CNRS
Streicher	Thomas	Technische Universtität Darmstadt

Commissione / *Jury*:

Abrusci	Vito Michele	Università Roma Tre	(Co-direttore / *Co-directeur*)
Curien	Pierre-Louis	Université Paris 7	(Esaminatore / *Examinateur*)
Girard	Jean-Yves	Aix-Marseille Université	(Co-direttore / *Co-directeur*)
Joinet	Jean-Baptiste	Université Jean Moulin Lyon 3	(Relatore / *Rapporteur*)
Martini	Simone	Università di Bologna	(Esaminatore / *Examinateur*)
Moriconi	Enrico	Università di Pisa	(Esaminatore / *Examinateur*)

Contents

Prelude: Frege's *Grundgesetze*

Frege's *Grundgesetze* [Fre13] contain one of the first rigorous formulations of a formalism for second order logic. As everybody knows, Frege's theory was shown to be inconsistent by Russell in 1901. However, [Fre13] contains an argument purported to show that all expressions in his formalism "have a denotation", and in particular that all propositions denote a definite truth-value. If this had been the case, then the consistency of the theory would have followed from that. Hence, Frege's argument was not correct.

I believe that there is no better prelude to this thesis than to give a sketch of Frege's wrong argument, and to briefly highlight its fallacies: on the one hand this proof provides a very instructive example of the obstinate circularity of second order reasoning, the actual subject of this work; on the other hand, Frege's unfortunate attempt anticipates, seventy years before, a similar and equally unfortunate attempt which is discussed throughout this text: in 1970 Martin-Löf presented a very elegant higher order type theory containing an impredicative type of all types. The Swedish logician provided an argument for the normalization of his theory, obtained by a natural generalization of Girard's argument in [Gir72] for the normalization of System F. One year later, Girard showed Martin-Löf's theory to be inconsistent, by deriving a paradox in it.

Though being yet another victim of the obstinate circularity of higher order logic, Martin-Löf's elegant theory constitutes one of the main sources of both philosophical and technical inspiration for this thesis. Much of what is claimed or discussed in the following pages comes from the subtle analysis of impredicativity made possible by this unfortunate episode.

Let us come to Frege's proof, then.

The language of the *Grundgesetze* (let us call it $\mathcal{G}$) would be called nowadays a functional language. It was based on Frege's distinction between *saturated* and *unsaturated* expressions, which roughly corresponds to the distinction between *closed* and *open* terms in modern functional theories: unsaturated expressions are those which contain free variables. In Frege's terminology, saturated expressions are names for objects, while unsaturated expressions are names for functions. For instance, a free variable x stands, in $\mathcal{G}$, as a name of a function.

A peculiar class of saturated expressions is the class of *propositions*, which are, in Frege's terminology, names for the True or for the False.

Functions can be divided into two classes: *first-level functions* $f(x), g(x), \dots$ are unsaturated expressions whose free variables $x, y, z, \dots$ can be substituted for (names of) objects; *second-level functions* $\phi(X(x)), \psi(X(x))$ are unsaturated expressions whose free variables $X, Y, Z, \dots$ can be substituted for (names of) first-level functions; a special second-level function is the function $\lambda x.(X(x))$[1], which allows to associate, with any first-level function $f(x)$, a *course-of-value* expression, i.e. a saturated expression $\lambda x.f(x)$ intuitively denoting the class of all objects

[1] where we replace Frege's ϵ notation with a more modern λ notation.

falling under the concept expressed by the function $f(x)$.

A peculiar class of first level functions is the class of truth-functions, which yield a proposition as soon as their free variables are substituted for (names of) objects: for instance, the function $x^2-1=(x+1)\times(x-1)$ yields the value True as soon as the variable x is replaced by a numerical expression.

In §29 Frege defines what it means for an expression of $\mathcal{G}$ to have a denotation. Frege assumes that the expressions `True` and `False`, so as the numerals $1,2,3,\dots$ have a (obvious) denotation; then he states that a *saturated* expression has a denotation if it yields a denoting expression when it is substituted for the free variables of an (appropriate) denoting unsaturated expression. An *unsaturated* expression has a denotation if the result of replacing its free variables with denoting saturated expressions always yields a denoting saturated expression.

In §30 Frege finally states and (tries to) prove a consistency theorem of the form: every expression in $\mathcal{G}$ has a denotation. This would imply that every proposition has a denotation, which is either the True, either the False, and thus that the theory $\mathcal{G}$ is consistent (as it is remarked in [Dum91a], it is unclear from Frege's text if he was aware of this fact).

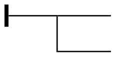
Remark that Frege's notion of denotation differs in many respect from the definition of a model-theoretic satisfaction relation. Indeed, expressions are not interpreted as elements of a model; on the contrary, Frege takes for granted that the constants of his language have a denotation and takes this as the basis of an inductive definition: as he remarks,

> These propositions are not to be construed as definitions of the words "to have a reference" or "to refer to something", because their application always assumes that some names have already been recognized as having a reference; they can however serve to widen, step by step, the circle of names so recognized. [Fre13]

Rather, to the eyes of the type-theorist, Frege's stipulations might remind the clauses defining the *computability* or *reducibility* predicates (see [Tai67, Gir72]) for typed λ-terms, a technique used to prove normalization theorems for typed λ-calculi. Indeed, if the reader takes `True` and `False` as the two only *normal* proposition, then he can look at Frege's consistency proof as a sort of normalization argument, showing that every proposition has a normal form.

Frege's proof is carried out following the inductive definition of the property of "having a denotation"; here we limit ourselves to the case of propositions. The basis case is obvious, since `True` and `False` denote, respectively, the True and the False, so as numerals $1,2,3,\dots$ denote the numbers $1,2,3,\dots$. For the case of a first-level function $f(x)$, he shows that, if N is a denoting object, then $f(N)$ is too; for instance, if $f(x)$ is the function $x \Rightarrow x^2$, he assumes P to be a denoting proposition, i.e. corresponding to a truth-value, and shows that $f(P)$ must denote the True. As a consequence, propositions built by substituting denoting objects for the free variables of a first-level functions have a denotation. He argues similarly for propositions of the form $\forall x.f(x)$[3].

The most important and delicate case concerns second-level functions: Frege first assumes $f(x)$ to be a denoting first-level functions and argues that, if $\phi(X(x))$ is a second-level function, then the first-level function $\phi(X(x))[f(x)/X] = \phi(f(x))$ has a denotation (as a consequence of the argument above for first-level functions); hence he can argue that, if $\phi(X(x))$ is a second-level function having a denotation, then the first-level function $\forall X.\phi(X(x))$[4] must have a denotation: for all object N, either *for all* first-level functions $f(x)$, $\phi(f(N))$ is the True, and then

[2] Written as  x in Frege's original notation.

x

[3] $\overset{\mathfrak{a}}{\smile}$ $f(\mathfrak{a})$ in Frege's notation.

[4] $\overset{\alpha}{\smile}$ $\phi(\alpha)$ in Frege's notation.

$\forall X.\phi(X(N))$ is the True, either for some first-level function $f(x)$, $\phi(f(N))$ is the False, and then $\forall X.\phi(X(N))$ is the False.

The reader may have noticed the circularity of the argument above: in showing that the new first-level function $\forall X.\phi(X(x))$ has a denotation, Frege is presupposing that all first-level functions $f(x)$ have a denotation, as a result of the argument developed above for first-level functions. Indeed, in order to show, for a given object N, that the proposition $\forall X.\phi(X(N))$ is the True, one has to show that, *for all* first-level functions $f(x)$, the proposition $\phi(f(N))$ is the True; hence, in particular, one has to show this for the first-level function $\forall x.\phi(X(x))$!

A similar form of circularity appears in the case of couse-of-values expressions: Frege has to show that the second-level function $\lambda x.X(x)$ has a denotation, and for that he has to show that, for any two first-level functions $f(x), g(x)$ having a denotation, the expression $g(\lambda x.f(x))$ has a denotation. This is shown by considering the possible cases for $g(x)$, taking as basis case the one of equality and appealing to the celebrated and unfortunate Basic Law V (stating that two course-of-value expressions $\lambda x.f(x), \lambda x.g(x)$ name the same object if and only if the proposition $\forall x(f(x) \Leftrightarrow g(x))$ is the True).

Again, Frege's argument contains a vicious circle: let $g(x)$ be the function $x = \lambda x.h(x)$; in order to show that the proposition $\lambda x.g(x) = \lambda x.h(x)$ has a denotation, one has to show that the proposition $\forall x(g(x) \Leftrightarrow h(x))$ has a denotation. This means that he has to show that, for every object N, $g(N) \Leftrightarrow h(N)$ is either the True or the False. Now, this presupposes in particular showing that $g(\lambda x.h(x))$, i.e. $\lambda x.g(x) = \lambda x.h(x)$ has a denotation.

As everybody knows, Russell was able to build a counterexample to Frege's consistency theorem by exploiting the circularity just sketched: he constructed a proposition R having no denotation. Indeed, R is such that, if it were the True then it would be the False, and if it were the False, then it would be the True. Hence, from the "normalization viewpoint", Russell had found an expression in $\mathcal{G}$ which has no normal form.

It is absolutely remarkable that, after Frege's unfortunate attempt, one had to wait almost eighty years before an actual proof of consistency for second order logic, through a normalization argument, was published (in Girard's thesis [Gir72]). The time the question remained unsettled, as well as the subtlety with which the circularity of second order reasoning is treated in this proof without falling into vicious circles bear witness to the hardness of the issues of understanding and justifying second order logic.

Part I

Introduction

Chapter 1

Explaining why *vs* explaining how

The perspective which underlies this thesis on the proof theory of second order logic is based over a methodological opposition which can be reconstructed through the heritage of the two main traditions in logic in the last century. The constructive tradition (intuitionism, realizability, computability theory) taught us to extract a finite, recursive content from proofs. The semantic tradition (model theory, proof-theoretic semantics) taught us to define and to prove the validity of more and more complex notions of proof - by relying, in accordance with Gödel's theorems, on more and more complex logical principles -.

The two points of view are complementary not only in their achievements, but also in their failures. The first fails to capture the difference between correct proofs and paradoxical, or meaningless, programs, as this distinction cannot be traced in a finite, recursive way: think of the problem of detecting the absence of loops in the execution of a computer program. The second fails to capture the finite and combinatorial structure of proofs, as semantical notions like truth or validity translate non elementary properties of formulae and proofs into non elementary properties of their denotations: typically, the validity of a formula involving quantification over an infinite domain is expressed by a quantification over an infinite domain.

The broad intent of this work is to draw the outline of a direction of research that will be (hopefully) developed by the author in the future years. This is why this text contains, in addition to philosophical arguments and some technical results, several proposals and technical ideas which are only sketched and left for future investigations.

Before introducing the reader to the context of this research (the debate over the legitimacy of a second order logic) and providing him an outline of the investigations contained in this thesis, we illustrate the idea of the opposition just introduced through a metaphor coming from a well-known novel by Borges.

1.1 The library of Babel and logical complexity

[...] the detailed history of the future, the autobiographies of the archangels, the faithful catalog of the Library, thousands and thousands of false catalogs, the proof of the falsity of those false catalogs, a proof of the falsity of the *true* catalog, ... [Bor00]

Meaningful proofs and meaningless codes The λ-calculus (so as many other universal models of computation) can be seen as an exemplification of Borges' *library of Babel.* Every algorithm, from the naïve computations of a young student to the wittiest products of a Palo

Alto company, from the attitude control system of a satellite to a randomly chosen sequence of instructions, finds its place among the shelfs of the library first conceived by Church in 1932.

At the same time, if a librarian randomly picked a book from this library, then, puzzled, he would be immediately faced with a question: what does it mean?

Indeed, most of the programs he would find consist in quite inscrutable sequences of λs and variables, or in visibly idiot programs, indefinitely reproducing themselves.

> One book, which my father once saw in a hexagon in circuit 15-94, consisted of the letters M C V perversely repeated from the first line to the last. Another (much consulted in this zone) is a mere labyrinth of letters whose penultimate page contains the phrase *O Time thy pyramids*. This much is known: for every rational line or forthright statement there are leagues of senseless cacophony, verbal nonsense, and incoherency. [Bor00]

Occasionally, he could bump into some books he would find himself able to read: books written, at least partially, in a language he understands. This language would tell him the circumstances in which to use these programs, and predict their possible outputs. In a word, he would recognize such programs as *typed* programs (in a certain type system among his favorite ones).

Anyway, without any acquaintance with (possibly many) type systems and without some luck, he would not be able to tell the meaning (nor the use) of those programs from the mere reading of a sequence of symbols.

> I know of one semibarbarous zone whose librarians repudiate the "vain and superstitious habit" of trying to find sense in books, equating such a quest with attempting to find meaning in dreams or in the chaotic lines of the palm of one's hand...[Bor00]

Acquaintance with many typing languages is not enough to tell, *in general*, meaningful programs, i.e. programs representing (total) functions, from meaningless, idiot, ones. This is the essence of Turing's theorem: one will never find an algorithm to put the library in order. Hence one will not find, in the library of Babel, a book telling the books worth reading from the rubbish ones.

Similarly to the case of λ-calculus and computation, a version of Borges' library for proofs arises from Kleene's ingenious remark that all the information needed to construct a proof can be compressed in a finite code. *Kleene's realizability* provides a library of codes (natural numbers in [Kle45]) which represent *all* arithmetical proofs (indeed, not just the proofs contained in a fixed formal system!).

At the same time, the librarian of Kleene's library might well spend his life trying to find the meaning hidden behind these meaningless lists of symbols.

> A realization number by itself of course conveys no information; but given the form of statement of which it is a realization, we shall be able in the light of our definition to read from it the requisite information. [Kle45]

The clauses defining realizability define the conditions under which a code *realizes* a certain formula. They provide the key to decrypt (some of) the books in the library. For instance, a code e realizes an arithmetical formula $\forall nA$ when, *for any* integer k, the code $\{e\}k$ (where $\{,\}$ denote Kleene's brackets) realizes the formula $A[\underline{k}/n]$.

A fundamental remark should strike the logician reader here: on the one hand proofs are coded, i.e. compressed into combinatorial objects. Logically speaking, this *coding* can all be expressed by means of formulae of a *fixed* logical complexity (say Σ^0_1 [1]). On the other hand,

[1] It is a well-known by logicians that recursive properties can be expressed by means of Σ^0_1 formulae, i.e. formulae of the form $\exists n.A$, where A contains no quantifiers.

the *decoding* clauses connecting codes to arithmetical formulae correspond to statements whose logical complexity *depends* on the logical complexity of the formulae. In the case above, the clause for a Π^0_1 formula, i.e. a formula starting with a universal arithmetical quantifier $\forall n$ and containing no other quantifier, is expressed by a formula which is (at least) Π^0_1.

It is common to semantical notions to have the property we have just described. For instance, the truth of a formula A, as characterized by Tarski's notorious condition

$$A \text{ is true if and only if } A \tag{1.1.1}$$

is a property whose logical complexity clearly grows with the logical complexity of the formula A under consideration (this has the well-known consequence that arithmetical truth cannot be uniformly expressed by an arithmetical formula). Similar remarks can be made for the notion of model-theoretic validity and for the notion of proof-theoretic validity (which is discussed in detail in this thesis).

Hence, the meaning of proofs of formulae of complexity greater or equal to Σ^0_1 cannot be *analyzed*, decomposed, by means of recursive (i.e. Σ^0_1) techniques. This is indeed a consequence of Gödel's theorems, which assert that the validity[2] of formulae of complexity superior to Σ^0_1 cannot be characterized by a recursive notion of provability: given a recursive and consistent description of provability, there exists a valid formula (of complexity Π^0_1) which is not provable following that description.

Proofs, as meaningless codes, are finite, combinatorial, objects. On the contrary, the meaning of those proofs, the properties which make these codes correct, or valid, proofs of a certain formula (an *evidence* for the formula, in Martin-Löf's terminology [ML87]), are described by clauses of growing logical complexity.

In a word, whereas the whole library of Babel can be described as a purely combinatorial structure, its meaningful part (or parts) cannot be entirely described in a recursive way.

"Proof-theory and logical complexity" Girard's monumental volumes [Gir90b, Gir89b] provide a rigorous and extensive application of this idea to vast parts of logic. In particular, they contain a proof-theoretical investigation of the logical complexities Π^1_1 and Π^1_2 by means of two recursive libraries of proofs:

- for the complexity Π^1_1, ω-proofs are "compressed" into codes for recursive (not necessarily well-founded) trees, while the property characterizing correct, or valid, ω-proofs, i.e. well-foundedness (of complexity Π^1_1), is non recursive;
- for the complexity Π^1_2, β-proofs are "compressed" into codes for recursive pre-dilators[3]. Here the property characterizing correct, or valid, β-proof is the non recursive Π^1_2 property of preserving well-foundedness.

The main advantage of the introduction of these recursive libraries was that the usual proof-theoretical properties could be investigated directly on the recursive proofs: as already remarked by Minc in [Min78], the cut-elimination algorithm could be directly defined and performed, in a primitive recursive way, on the "preω-proofs". On the contrary, the *Hauptsatz*, i.e. the fact that the algorithm terminates, required the logically complex hypothesis of well-foundedness.

[2] Technically, the truth of arithmetical formulae of complexity superior to Σ^0_1, which is equivalent to the validity of second order logical formulae of complexity superior to Π^1 (see chapter (2) for a presentation of these hierarchies of formulae).

[3] A *pre-dilator* (see [Gir85]) is a functor from the category of linear orders into itself preserving direct limits and pull-backs. A *dilator* is well-foundedness preserving pre-dilator, i.e. a pre-dilator which is a functor from the category of ordinals into itself. The notion of dilator was invented by Girard as a tool to investigate ordinal notation systems and Π^1_2-logic from an abstract mathematical point of view.

This technique allowed then to separate the recursive content of cut-elimination, which lies in the algorithmic transformation of proofs, from its logically complex one, given by termination.

> Let us give a more precise picture of what is going on:
>
> *a*) to each logically complex concept (ω-proof, β-proof, dilator) one associates a Π^0_1 (elementary) concept (preω-proof, preβ-proof, pre-dilator, respectively); this associated concept is weaker (e.g. every dilator is a predilator).
>
> *b*) Most constructions (cut-elimination procedures, the functor $\mathbf{\Lambda}$, ...) involving logically complex concepts can be extended to the associated elementary concepts. A typical example is the cut-elimination theorem for $L_{\omega_1\omega}$: in chapter 6 we prove cut-elimination for non-wellfounded ω-proofs of non-wellfounded formulas of $L_{\omega_1\omega}$ (i.e. preω-proofs of pre-formulas). A more familiar example is the extension of familiar ordinal constructions (sum, product, exponential, the Veblen hierarchy) to linear orders (= «preordinals »). Steps *a*) and *b*) can be thought of as an algebraization of current proof-theoretic constructions: typically, in *b*) we manage to do the constructions without «well-foundedness »assumptions. [Gir90b]

The idea I tried to illustrate through the image of the library of Babel constituted the main inspiration for this thesis on the proof theory of second order logic: on the one hand, the explanation of the meaning of second order proofs, so as their justification, runs into paradoxes and apparent "vicious circles" (see next subsection), at the point that it is generally considered controversial whether second order logic can be actually called logic. On the other hand, such "circular" proofs, as finite, recursive, objects, i.e. as *programs*, are the object of a quite rich and extensive literature, often confined to computer science departments and ignored in the philosophical literature.

In [Gir00] Girard describes the growing influence of theoretical computer science on proof-theory as a shift of the latter from its original foundational motivations ("why does mathematics work?") to more pragmatical, concrete, ones ("how can we make it work - on a computer, for instance - ?"). To this shift there corresponded a change in the technical equipment of the proof-theorist: from logical notions of greater and greater complexity (comprehension principles, transfinite inductions, determination axioms) to combinatorial tools (recursion theory, λ-calculus, natural deduction) and mathematical concepts (coming from category theory, topology, functional analysis).

> Cette citation imaginaire résume l'idéologie moyenne du théoricien de la demonstration de 1950. Elle situe d'emblée la théorie de la démonstration dans une problèmatique fondamentaliste (l'élimination des paradoxes) qui affirme que la logique donne le sens profond des mathèmatiques, ce que j'appellerai le «pourquoi ». Plus tard, vers 1985, l'informatique devait promouvoir une approche plus pragmatique, ce que j'appellerai le «comment »: ce comment est une préoccupation bien moins noble que le pourquoi, mais qui demande un appareillage beaucoup plus subtil. [Gir00]

Following Girard's suggestion, we can then draw a distinction between proof-theoretical investigations addressing the question "*why* does second order logic works?" (if it actually does) and proof-theoretical investigations addressing the question "*how* does second order logic work?".

The investigations of the first type concern the issues about the validity of second order reasoning, in particular consistency proofs, of syntactical or semantical nature. The resolution of Takeuti's conjecture ([Tak57]), regarding the *Hauptsatz* for second order logic, is a typical example. Issues about the representation of second order proofs (as the Curry-Howard correspondence between second order natural deduction and System F) and about their implementation (second order type inference, polymorphic functional programming) are examples of the second family of investigations.

Obviously there might be superpositions between these two directions of research: for instance, several important syntactical properties were discovered by means of semantic techniques (this was the case for the so-called parametric interpretation of polymorphism [Rey83], see chapter (5)).

Nevertheless the discussion above should convince the reader of the irreducibility of the two approaches: the validity of a second order Σ^1 statement or proof cannot be *analyzed* by means of recursive techniques. For instance, the normalization arguments for proofs of such statements must rely on comprehension principles, i.e. set-theoretic principles of growing logical complexity. This "pragmatic" (see [Dum91b]) or "epistemic" circularity affecting the "why-proof theory" of second order logic is discussed in detail in the second part of this thesis.

On the contrary, this circularity is of no harm from the viewpoint of the "how-proof theorist": to him, the numerous auto-applications occurring in second order proofs, which might appear incestuous to the Russellian philosopher, are just examples of standard recursive techniques. The third part of this thesis contains two combinatorial analyses of the vicious circles of second order proofs, the one based on the semantic property of parametricity, the other based on type inference and unification theory.

A final remark is that the "how-proof theorist", as the librarian of the library of Babel, cannot rely on a book telling him the border between valid proofs and rubbish. Indeed, one of the recurring aspects of this work, from the prelude to the last chapter, is the interest in *wrong* proofs. In a sense, just like a complete understanding of computation required to take into consideration also *partial* (i.e. wrong) algorithms, the investigations that follow are hinged on the belief that the combinatorial structure of the whole library might turn out to be of more interest than the logically complex characterization of its meaningful parts.

> Others, going about it in the opposite way, thought the first thing to do was to eliminate all worthless books. They would invade the hexagons, show credentials that were not always false, leaf disgustedly through a volume and condemn entire walls of books. It is to their hygienic, ascetic rage that we lay the senseless loss of millions of volumes. Their name is execrated to day, but those who grieve over the "treasures" destroyed in that frenzy everlook two widely acknowledged facts: one, that the Library is so huge that any reduction by human hands must be infinitesimal. And two, that each book is unique and irreplaceable, but (since the Library is total) there are always several hundred thousand imperfect facsimiles-books that differ by no more than a single letter, or a comma. Despite general opinion, I daresay that the consequences of the depredations committed by the Purifiers have been exaggerated by the horror those same fanatics inspired. They were spurred on by the holy zeal to reach someday - through unrelenting effort - the books of the Crimson Hexagon - books smaller than natural books, books omnipotent, illustrated, and magical. [Bor00]

1.2 The Quinean critic and proof theory

The debate on the foundations and the legitimacy of second order logic provides an interesting test bench for two rather antipodal perspectives on logic: on the one hand, the analytic tradition in the philosophy of logic, focusing on semantical justification, aiming at clarifying what the expressions of logical formalisms *stand for*; on the other hand, the proof-theoretical tradition, building on Gentzen's results on sequent calculus and the *Curry-Howard* bridge with theoretical computer science, rather focusing on the inner properties of logical syntaxes (e.g. cut-elimination, *Church-Rosser*, subformula etc.), crucial for programming purposes.

Two remarkable facts are among the motivations of this work. First, the fact that the philosophically-oriented tradition appears generally much more hostile than the other towards second order, or "impredicative", logics (with some notable exceptions, obviously, for instance [Boo75, Sha00]). Second, the fact that most of the technical advances and results on second order logic obtained within the computer science-oriented tradition (which largely belong to a period which goes from the publication of Girard's thesis in 1972 to the beginning of the nineties) are substantially ignored in the philosophical debate (again, with notable exceptions like [LF97]).

Here we recall some of the philosophical challenges which constitute the background for the philosopher getting acquainted with second order logic, as well as some of the technical cornerstones, which constitute the background for the "computer-science-oriented" proof-theorist.

1.2.1 Philosophical disputes over second order logic

Quine's "paradigmatic" challenge

> By treating predicate letters as variables of quantification we precipitated a torrent of universals against which intuition is powerless. We can no longer see what we are doing, nor where the flood is carrying us. Our precautions against contradictions are *ad hoc* devices, justified only in that, or in so far as, they seem to work. [Qui80]

Quine's well-known animadversions upon second order logic constitutes the center of gravity of the debate on the subject in analytic philosophy. It was the opinion of the influential american philosopher that the appeal to second order logic rested upon a confusion about the interpretation of predicate letters.

The "prodigal logician" Frege and the "confused logician" Russell are considered by Quine as responsible for this misunderstanding. On the one hand, in analogy with the fact that first-order variables are usually taken as *names* for individuals, they took predicate variables as *names* of attributes or universals. On the other hand, their resulting theories were to Quine completely unsatisfactory: Frege's *Grundgesetze* contained a contradiction, whereas the consistency of Russell's *Principia* was obtained at the price of introducing the *ad hoc* discipline of typing.

Quine's therapy for this confusion is resumed by the celebrated expression of second order logic as "set theory in sheep's clothing": when one freely talks about predicate variables and their related attributes, indeed "a fair bit of set theory has slipped in unheralded [Qui86]". Hence his attempt to expose the (first-order) set-theoretical commitments implicit in second order logic.

> [...] consider the hypothesis $\exists y \forall x(x \in y \Leftrightarrow F(x))$. It assumes a set $\{x|F(x)\}$ determined by an open sentence in the role of $F(x)$. This is the central hypothesis of set theory, and the one that has to be restrained in one way or another to avoid the paradoxes. This hypothesis itself falls out of sight in the so-called higher-order predicate calculus. We get $\exists G \forall x(G(x) \Leftrightarrow F(x))$, which evidently follows from the genuinely logical triviality $\forall x(F(x) \Leftrightarrow F(x))$ by an elementary logical inference. [Qui86]

As Boolos comments,

> reading him, one gets the sense of a culpable involvement with Russell's paradox and of a lack of forthrightness about its existential commitments. [...] Quine, of course, does not assert that higher-order predicate calculi are inconsistent. But even if they are consistent, the validity of $\exists X \forall x(X(x) \Leftrightarrow x \notin x)$, which certainly looks contradictory, would at any rate seem to demonstrate that their existence assumptions must be regarded as "vast". [Boo75]

The controversy over second order logic in the philosophical literature revolves around Quine's challenge: is this to be considered as a *primitive* part of logic, or is it rather just a confusing idea to be replaced by a rigorous first-order formalization?

In [Sha00] Shapiro tracks the origins of this controversy, underlining its *paradigmatic* character: Quine's major confidence in first-order set-theory is there explained as a byproduct of the historical success of first-order logic as a Kuhnian paradigm:

> It seems that this general consensus was not based on a philosophy of foundational studies. It was more of a research programme, suggesting that first-order model theory is the best place to focus intellectual attention. In short, first-order logic became a Kuhnian paradigm. [Sha00]

In order to highlight this paradigmatic character, Shapiro sketches an imaginary debate between an advocate of second order logic (called "Second") and an advocate of first-order set theory (called "First"), ending in a regress:

> [...] First raises a question concerning the range of the second-order variables. She asserts that the meaning of the second-order terminology is not very clear [...]. Second could retort that First knows perfectly well what locutions like "all subsets" mean, and he may accuse her of making trouble for the sake of making trouble. They would then be at a stand-off. [Sha00]

As Shapiro's numerous examples show, this debate over the right interpretation of predicate variables concentrates over the question: what do such variables *stand for*? Indeed, the technical tools involved in it (see for instance [Boo75, Sha00, Vaa01]) are mainly model-theoretical. Still, Shapiro's comprehensive book [Sha00] contains very few remarks on the proof-theory of second order logic and suggests the view that there is little hope to find a solution to the controversy above within a proof-theoretic approach:

> The more philosophical disputes noted here do not concern the correctness of informal mathematics, but rather things like how the discourse should be described, what it means, what it refers to, and what its non-logical terminology is. [...]
>
> This explains why the proof theories of the logics under examination here are remarkably similar, and underscores the foregoing thesis that the differences between first-order logic and higher-order logic lie primarily in the different views on the totality of the range of the extra variables - in the model theory. [Sha00]

In a first sense, this thesis is then an attempt to reject the suggestion above, by a closer examination of what is offered in the proof theory market. In particular, it will be argued that, by switching the focus from the interpretation of predicate variables to the interpretation of proofs in second order logic, a bunch of deep and stimulating ideas, often unexplored in the philosophical literature, opens up.

Proofs and the "vicious circle principle"

> The choice between predicative and impredicative theories [...] is sometimes said to depend upon whether mathematical entities are regarded as created by our thinking or as existing independently of us. We are then at a loss to know how to resolve a metaphysical issue couched in these metaphorical terms. Was the monster group *discovered* as Laverrier discovered Neptune? Or was it *invented*, like Conan Doyle invented Sherlock Holmes? How can we decide? And can the legitimacy or illegitimacy of a certain procedure of reasoning within mathematics possibly depend on our answer? [Dum91a]

A very influential approach to the interpretation of proofs arises from Prawitz's and Dummett's research on an alternative semantics for logic centered on the notion of proof. Unsatisfied with the Tarskian definition of validity, Prawitz provided in [Pra71a] a definition of validity for natural deduction derivations which does not rely on a set-theoretical interpretation of the expressions of the language, but rather on the possibility to *transform* (in the sense of Gentzen's cut-elimination) derivations into a so-called *canonical form*.

Prawitz's proof-theoretic validity is the main ingredient of a general program aiming at a philosophical justification of deduction (see [Dum91b]) from an inferentialist perspective, opposed to the usual Tarskian, referentialist, one; such a justification does not focus on what logical expressions stand for, but on how they are *used* (by means of their associated introduction and elimination rules) in the construction of proofs and deductive arguments.

Whereas proof-theoretic validity was originally conceived to include second-order logic (see [Pra71a]), the latter was later excluded from the general "justificationist" project. Indeed, as Dummett argues in [Dum91a, Dum06], the justification of second order proofs ends up in a "vicious cycle" which was historically first remarked by Poincaré [Poi06].

As a typical example, if one wishes to show that a certain individual t is *inductive*, i.e. that the predicate $N(x) := \forall X(\forall y(X(y) \Rightarrow X(\underline{s}(y))) \Rightarrow (X(\underline{0}) \Rightarrow X(x)))$ holds of t, then he is supposed to show that, *for any predicate* $P(x)$, the predicate $\forall y(P(y) \Rightarrow P(\underline{s}(y))) \Rightarrow (P(\underline{0}) \Rightarrow P(x))$ holds of t. But this means that, in particular, one is supposed to show that $\forall y(N(y) \Rightarrow N(\underline{s}(y))) \Rightarrow (N(\underline{0}) \Rightarrow N(t))$ holds and thus, since $\forall y(N(y) \Rightarrow N(\underline{s}(y)))$ and $N(\underline{0})$ clearly hold, that $N(x)$ holds of t: this is the start of an infinite regress.

This is how Russell described the "vicious cycle principle" in 1906:

> I recognize further this element of truth in M. Poincaré's objection to totality, that whatever in any way concerns all or any or some of a class must not be itself one of the members of a class. [...]
>
> In M. Peano's language, the principle I want to advocate may be stated: "Whatever involves an apparent variable must not be among the possible values of that variable". [Rus06b]

The reader has already encountered similar "vicious circles" in Frege's proof in the *Grundgesetze*. At the beginning of the 19th century Poincaré and Russell held that the existence of such circles constituted the reason for the antinomy in Frege's "pure" second order formalism.

On the other side of the dispute there was Carnap's remark [Car83] that, though the explanation given above is surely circular, actual proofs are not built in that way: a proof of the fact that the number 3 is inductive consists in a formal argument that the predicate $\forall y(X(y) \Rightarrow X(\underline{s}(y))) \Rightarrow (X(\underline{0}) \Rightarrow X(x))$ holds of t in which the predicate variable $X(x)$ is taken as a mere "parameter" and finally generalized.

> If we had to examine every single property, an unbreakable circle would indeed result, for then we would run headlong against the property "inductive". Establishing whether something had it would then be impossible in principle, and the concept would be meaningless. But the verification of a universal logical or mathematical sentence does not consist in running through a series of individual cases [...] The belief that we must run through all individual cases rests on a confusion of "numerical generality" [...] We do not establish specific generalities by running through individual cases but by logically deriving certain properties from certain others. [Car83]

Poincare's objections can be found, in an adapted form, in Dummett's rejection of second order logic from his justificationist program (this is discussed in detail in chapter (4)). In particular, Poincaré claimed that, by appealing to second order concepts, logic loses the neutral character which makes it a solid foundation for mathematics ("*la logique n'est plus stérile*" [Poi06]). Similarly, Dummett points out that, by introducing second order natural deduction rules, one is forced to give up the self-explanatory character of deduction and to adventure into the dangerous fields of mathematical invention.

Dummett's objection must be distinguished from Quine's: for the former, rather than tacitly assenting to set-theoretic existence assumptions (concerning the reference of the predicate variables), the logician adopting a second order language is endorsing a controversial view about the forms of reasoning that one is entitled to accept. In particular, a view whose intelligibility demands for more than a "self-explanatory" proof-theoretic analysis.

> [...] the vicious circle principle makes no assertion about what does or does not exists: it merely distinguishes between what does and what does not require a further explanation.

1.2.2 Type theory "in sheep's clothing"

From intuitionism to type λ-calculi One of the most fruitful directions in the proof theory of the last century arose from the development of a connection between the intuitionist notion of construction and the mathematical notion of computable function. Historically, Kleene was the first to look in that direction. In the intuitionistic explanation of proofs contained in the classical [Hey56], it is stated that a proof of a formula of the form $\forall n \exists m A(n, m)$ consists in a method μ yielding, for any k, an integer $\mu(k)$ along with an intuitionistic proof of $A(k, \mu(k))$. Kleene's guiding idea, in his 1945 paper on realizability [Kle45], was then to replace the philosophical notion of "method" with a mathematically rigorous one: from an intuitionistic proof of $\forall n \exists m A(n, m)$ one should extract then a computable function ϕ yielding, for any k, an integer $\phi(k)$ such that $A(k, \phi(k))$ holds intuitionistically. In particular, a proof of the *totality* of a recursive function ϕ (i.e. the statement $\forall n \exists m(n = \phi(m))$ should provide concrete instructions on how to compute the function ϕ.

On these lines Kleene defined an interpretation of the proofs of intuitionistic arithmetics as computable functions, i.e. as programs. By reconstructing the realizability interpretation within λ-calculus, Kreisel's "modified" version [Kre59] of realizability added an important idea: with every arithmetical proposition A one could associate a *type* A^*, such that all programs extracted from proofs of A could be given the type A^*. Hence, proofs of totality for recursive functions were interpreted as programs *of type* $\mathbf{N} \to \mathbf{N}$, where $\mathbf{N}$ is the type of natural numbers. A similar idea was developed by Gödel in his *Dialectica* interpretation of arithmetics [G5̈8].

Between the fifties and the sixties Curry [CF58] and Howard [How80] realized that the connection between intuitionistic proofs and typed programs could be given a yet more tight description: derivations in first order intuitionistic natural deduction can be directly interpreted as (they are, in a sense, isomorphic to) simply typed λ-terms. This Curry-Howard correspondence adds to Kreisel's one a dynamical aspect: Gentzen's transformations over natural deduction derivations correspond directly to reductions of the associated λ-terms. In a word, normalization steps in natural deduction correspond to normalization steps in λ-calculus.

The Curry-Howard correspondence between intuitionistic systems and typed λ-calculi is by now evolved into an extremely vast field of research, at the bridge between logic, pure mathematics and theoretical computer science. The proofs-as-programs paradigm, which is at the very heart of the investigations here presented, constitutes indeed one of the most powerful tools in proof theory, witness the many active research programs based on it (as Girard's *geometry of interaction* program [Gir89c] or Krivine's program [Kri12]).

System F The extension of the proofs-as-program correspondence to second order intuitionistic logic was provided independently by Girard in his thesis [Gir72] and later by Reynolds in [Rey74]. The second order (or polymorphic) λ-calculus, called System F in [Gir72], whose typed terms correspond to intuitionistic second order natural deduction derivations, introduces an "impredicative" type discipline which, unlike Russell's type discipline, allows the typing of functions applied to themselves.

Terms in System F are called *polymorphic* since they can be given several types at once: for instance, a term of a second order type $\forall \alpha \sigma$ can be regarded as a term of type $\sigma[\tau/\alpha]$, *for every* type τ of System F, included $\forall\alpha\sigma$. It is through this circularity thats second order type theory inherits the "vicious circles" of second order logic.

The main result of [Gir72] is a proof that System F enjoys the strong normalization property. By relying on the Curry-Howard correspondence, this result was used to provide a positive answer to a conjecture posed by Takeuti in 1957, i.e. whether the *Hauptsatz* holds for second order logic.

Though several semantical proofs of cut-elimination for second order logic were proposed in the sixties (see [Tai68, Tak67, Pra68]), the proof in [Gir72] was the first to provide a syntactical normalization argument. This argument was based on an extension of Tait's technique of *computability predicates* ([Tai67]) by means of the notion of *reducibility candidates*. The latter allow to define the computability of polymorphic terms in a way which, though impredicative, escapes "vicious circles". In a sense, Girard's reducibility argument fixes the bugs in Frege's consistency argument of the *Grundgesetze*. A closer examination of this technique can be found in chapter (4).

Girard's work on System F was the starting point of several fruitful lines of research on second order logic from the Curry-Howard perspective. First, "Girard's trick" ([Gal90]) for proving normalization introduced a new way to escape the circularity of impredicative types and propositions. The so-called Tait-Girard reducibility technique became indeed a standard tool for proving normalization for higher-order typed λ-calculi. Slightly modified versions of this technique were used by Prawitz ([Pra71a]) and Martin-Löf ([ML70a, ML75]) to prove the normalization of several intuitionistic higher order natural deduction theories.

Second, in [Gir72] it was observed that a program of a universal type $\forall\alpha.\sigma$ cannot actually "depend" on the information about the input type to be substituted for α. Girard showed that a paradox (hence, a counterexample to normalization) would result from the violation of this "genericity" ([LMS93]) constraint. This remark is at the basis of an interpretation of impredicative quantification (see [Rey83]) which, in a sense, provides a rigorous mathematical formulation of Carnap's argument against the "vicious circle principle" (see chapter (5)).

Finally, the investigations on the semantics of System F shed far more light on the relations between the second order and set theory than Quine and Shapiro thought. In 1984 Reynolds [Rey84] showed that, if one considers arbitrary set-theoretic interpretations of typed λ-calculi, then there can be no model of System F: he was able to exploit impredicative quantification to show that a counterexample to Cantor's theorem on the cardinals would result from the existence of such a model (see (5)). This (quite old!) result seems to contradict directly Shapiro's claim on the irrelevance of proof-theory for the second order logic/set theory debate.

Nevertheless, many mathematical constructions have been successfully applied to devise (non set-theoretic) models of System F: for instance, in [Gir86, GLT89] it was shown that one can interpret impredicative types, in a categorial framework, by means of *direct limits* of certain finite spaces (called *coherent spaces*), a very simple structure which became known for having led to the discovery of *linear logic* (see [Gir87]). Another well-known example is Hyland's *effective topos* [Hyl82], which allows to extend Kleene's realizability to System F within a topos theory[4].

It must be said that most of these advances are still confined to the literature on computer science-oriented logic. System F and its legacy constitute indeed a good example of the gaps existing between the literature on logic coming from philosophy departments and the literature coming from mathematics and computer science departments. Just to give an example, Shapiro's comprehensive book on second order logic has no reference to System F or to whatever has been written on the mathematical aspects of polymorphic type theories.

One of the aims of this thesis is to contribute to fill this gap, as it seems quite difficult to deny

[4]Reasons of space and time imposed to the author not to treat in detail the many and profound aspects which come from the literature on the denotational semantics of higher order type theory. This is surely a serious lack in the investigations contained in this thesis, to be left for a future work.

that the results and aspect aforementioned have a serious impact on the philosophical challenges and disputes over second order logic sketched in the previous subsection.

1.3 Outline of the thesis

In the second chapter of this first, introductory, part, we describe in detail the proof-theoretic correspondences between, respectively, second order arithmetics and second order logic, and second order logic and polymorphic type theory, or System F.

Starting from the idea that a "logic" is given by a language (i.e. a set of formulae), a set of proofs of such formulae and a set of transformations between proofs, we reconstruct these well-known correspondences as "functors" between the various logics, i.e. maps preserving all relevant proof-theoretic properties. This description highlights then the fact that arithmetical derivations, second order derivations and polymorphically-typed λ-terms essentially represent the same proofs.

The second part is dedicated to the "why-proof theory" of second order logic. In chapter (3) we reconstruct and confront two distinct, though historically related, approaches to the interpretation of proofs: the first one focuses on the analysis of the inferential content of proofs, and historically derives from the proof theoretic semantics tradition, introduced by Dummett and Prawitz (see [Pra71a, Dum91b]). The second one interprets proofs as untyped programs and focuses on the behavioral content of proofs, i.e. the way in which they interact through the cut-elimination algorithm. Roots of this interactionist point of view are traced to Kleene's realizability ([Kle45]) and to the Tait/Girard reducibility technique ([Tai67, Gir72]).

In chapter (4) we present and discuss the *Hauptsatz* for second order logic, and we address the epistemological issues arising from Girard's proof ([Gir72]) from the two viewpoints. The inferentialist proof-theorist appeals to an updated version of Poincaré's "vicious circle" objection and claims that impredicative reasoning cannot be justified proof-theoretically; by contrast, the technique of reducibility candidates, used in the proof, appears much more akin to the untyped perspective of the interactionist proof-theorist, and reveals a different, "epistemic", form of circularity. Still, this weaker circularity makes justification, in a sense, pointless: we sketch the example of Martin-Löf's inconsistent higher order theory (as the one in [ML70b]) admitting an epistemically circular normalization arguments.

The third part is dedicated to the "how-proof theory" of System F. In chapter (5), after recalling Reynolds' argument for the impossibility of a set-theoretical interpretation of second order proofs, the parametric and dinatural interpretations of polymorphism ([Rey83, GSS92]) are presented as providing a clear mathematical content to Carnap's defense of impredicative reasoning in [Car83]. By relying on a syntactic reformulation of these interpretations, the $\mathbf{\Pi}^1$-completeness theorem (5.2.4) is proved, which states that the closed normal λ-terms in the reducibility of the universal closure of a simple type are typable in simple type theory. This theorem provides, by a passage through impredicative quantification, a bridge between the interactionist and the inferential interpretation of propositional proof: by closing types universally, one indeed recovers the usual "last rule conditions" required by the inferential proof-theorist.

In chapter (6) a constructive viewpoint on impredicativity and its paradoxes is developed by an analysis of the typability problem form the λ-terms associated with (intuitionistic) second order proofs. To the "vicious circles" in the proofs there correspond recursive (i.e. circular) specifications for the types of the λ-terms. The "geometrical" structure of these vicious cycles is investigated (following [LC89, Mal90]). As shown by Girard's paradox, a typable term need not be normalizing: the combinatorial analysis of typing does not discriminate between terms corresponding to correct or to incorrect proofs.

A combinatorial characterization of the typability of λ-terms is investigated, by means of a generalization of the notion of "compatibility" between the constraints forced by recursive equations in [Mal90]. In particular, it is conjectured that this notion fully characterizes typability for system U^- (an inconsistent extension of System F connected with Martin-Löf's inconsistent type theory, see [Gir72]), and some results in this direction are shown.

Some interesting consequences motivating this conjecture are proved at the end of the chapter. Among them, the fact that every strongly normalizable term would be typable in U^-, the decidability of the typability problem for the systems U^- and N as well as the fact that every total recursive unary function (suitably coded in λ-calculus) can be given type $\mathbf{N} \to \mathbf{N}$ in System U^-.

Chapter (7), in the fourth, concluding, part, contains a sketch of some future lines of research which arise from the perspectives on "how-proof theory" developed in the third part. Indeed, the type-theoretic investigations contained in chapter (6) prompt a way to understand the limitations imposed by incompleteness, in line with the metaphor of the Library of Babel: since every true Π^0_2 statement corresponds to the totality of a certain recursive function, from the typing of a λ-term computing the function (an untyped realizer of the statement) one should retrieve a proof of the statement in an inconsistent extension of second order logic. At the same time, it should be expected that the line between valid and invalid, or "paradoxical", derivations in this extended system cannot be recursively drawn. In a sense, this would mean that we can have all the proofs, but we cannot tell once for all those we can actually trust.

Chapter 2

Arithmetics, logic and type theory

The interaction between the proof theories of (second order) arithmetics, logic and type theory constitutes the technical background of this thesis. The relation between the first and the second usually goes under the name of *Dedekind's translation*, from Dedekind's intuition of a purely logical (second order) definition of the natural numbers. The relation between the second and the third is given by the *Curry-Howard correspondence*, from the remarks by Curry [CF58] and Howard [How80] of a substantial isomorphism between intuitionistic sequent calculi and typed λ-calculi.

This introductive chapter is devoted to present these three formalisms and their aforementioned relationships by relying on a categorial intuition: as in denotational semantics, a "logic" is though as a category made of objects (formulae), morphisms (proofs) and diagrams (given by Gentzen's transformations over proofs); hence Dedekind's translation from second order arithmetics to second order logic and the Curry-Howard translation of the latter into second order type theory (System F) are described as functors between such logics.

Finally, we present the systems F^{ω}, U^{-}, U, N, which are extensions of System F which will be used in the next chapters, highlighting some of the theoretical challenges connected with the extension of second order type theory (indeed, all such systems but F^{ω} are inconsistent).

2.1 The proof-theoretic notion of "logic"

2.1.1 From Hilbert's program to structural proof theory

Whereas in model theory one is mainly interested in formulas and their interpretations, in proof theory one takes as the central notion the one of proof.

The problem of derivability Historically, the first systematic investigations on proofs were developed in the context of Hilbert's program (for instance [Hil96a]); the mathematical presentation of proofs was provided by derivations built within a formal system: the so-called *Hilbert systems* were made of a (usually quite large) set of axioms and by a set of rules, which in most cases was reduced to the sole rule of *modus ponens*.

By means of Hilbert-systems the vague notion of "demonstrability", central for Hilbert's program, was replaced by a rigorous one, i.e. derivability within a formal system; moreover, it was shown that the property of being a derivation could be coded by a primitive recursive predicate; this was the starting point of a series of results which marked the failure of Hilbert's program: in 1931 Gödel showed that there exist (true) sentences which are not derivable within sufficiently expressive formal systems, in 1936 Turing showed the existence of non-recursive problems and in the same year Church showed that derivability within the formal system of first order logic is one of them.

Serious improvements in the analysis of derivability were obtained with the development of the so-called "structural" approach to proof theory, started with Gentzen's pioneering thesis [Gen64]. In this approach, Hilbert's systems are replaced by *sequent calculi* and *natural deduction calculi*, characterized by a significantly smaller number of axioms and a long list of rules. Gentzen showed that, when dealing with questions of derivability within first order logic, one can restrict the search to derivations in which there are no occurrences of the *cut-rule* (sometimes called *analytic derivations*):

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma' \vdash A, \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ (cut) \tag{2.1.1}$$

Such derivations exhibit a very peculiar structure: at every stage of the derivation the formulas occurring in the rules are *subformulae* of the formulae occurring in the conclusion of the derivation. Remark that the rule (cut) clearly violates this property.

The *Hauptsatz* (as it was originally called by Gentzen), that is, the cut-elimination theorem, can be considered as a fundamental result in logic, from a proof-theoretic viewpoint. Indeed, it allows to provide purely proof-theoretical proofs of the *consistency* and *completeness* of first-order logic, two results which are often expressed and proved in a model-theoretic setting.

The consistency of first order logic is an immediate corollary of the cut-elimination theorem: if the falsity were provable, then it would have a cut-free proof; however, since no formula is a subformula of the falsity, there can be no cut-free derivation of the falsity.

The fact that the *Hauptsatz* implies the completeness of first-order logic was first established by Schütte [Sch56] starting from the following remark: given a formula A, it is possible to devise a procedure which looks for possible cut-free derivations of A by recursively looking for the premisses of a (one-sided) sequent; indeed, the subformula property provides a finite bound on the set of possible premisses of a sequent. This algorithm, starting from a formula A, gradually builds a tree by repeatedly looking for premisses and halts as soon as all of it branches terminate on an axiom sequent, i.e. a sequent of the form $\vdash \Gamma, A, \neg A$. In that case (thanks to König's lemma) the finite tree obtained must be a cut-free derivation of A. In particular, if A is derivable in first-order logic, the algorithm produces a cut-free derivation of A. Otherwise, i.e. if the algorithm never halts, the tree must contain an infinite branch made of rules of sequent calculus; now Schütte was able to show that the negation of the formulae occurring in this infinite branch generates a counter-model of A: hence, if A is not derivable, from the infinite proof-search for A we get a counter-model to A.

The dynamics of proofs At the basis of Gentzen's *Hauptsatz* there is a procedure which recursively transforms derivations in which there are occurrences of the *cut rule* into derivations in which this rule does not occur. For instance, an occurrence of the rule (cut) with *cut-formula* a conjunction:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ d_1\\ \Gamma, A, B \vdash \Delta\end{array}}{\Gamma, A \wedge B \vdash \Delta} \qquad \dfrac{\begin{array}{c}\vdots\ d_{21}\\ \Gamma_1' \vdash A, \Delta_1'\end{array} \quad \begin{array}{c}\vdots\ d_{22}\\ \Gamma_2' \vdash B, \Delta_2'\end{array}}{\Gamma' \vdash A \wedge B, \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ (cut) \tag{2.1.2}$$

can be transformed into a derivation in which the occurrences of the rule (*cut*) have with *cut-formula* formulae of strictly smaller logical complexity:

$$\dfrac{\dfrac{\dfrac{\vdots\, d_1}{\Gamma, A, B \vdash \Delta} \quad \dfrac{\vdots\, d_{21}}{\Gamma_1' \vdash A, \Delta_1'}}{\Gamma, \Gamma_1', B \vdash \Delta, \Delta_1'}\,(cut) \qquad \dfrac{\vdots\, d_{22}}{\Gamma_2' \vdash B, \Delta_2'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,(cut) \tag{2.1.3}$$

We owe entirely to Gentzen this idea of transformations over proofs. By applying a quite complex induction argument Gentzen was able to show that it is possible, by performing repeated applications of these transformation, to eliminate all cuts and transforming an arbitrary derivation of first-order logic into a cut-free one.

Gentzen's transformations provide a insight on the mutual structure of logical rules: the premisses of the rules for introducing a logical symbol (for instance $\wedge$ in the example above) at the right and at the left of the stroke symbol $\vdash$ must be in accordance in order for the transformation to be applied. This remark became well-known in proof-theory thanks to Prawitz's work on natural deduction [Pra65], under the name of *inversion principle*:

> Let α be an application of an elimination rule that has B as consequence. Then, deductions that satisfy the sufficient condition [...] for deriving the major premiss of α, when combined with deductions of the minor premisses of α (if any), already "contain" a deduction of B; the deduction of B is thus obtainable directly from the given deductions without the addition of α. [Pra65]

Remark that the inversion principle is a *local* criterion, allowing for a single application of a Gentzen transformation. Genzten's *Hauptsatz*, on the contrary, is a stronger, *global*, result, showing that the repeated application of the transformations terminates producing a cut-free derivation (this remark will be discussed in more details in chapter (3)).

Gentzen's transformational approach induced a severe change of focus in the study of proofs with respect to Hilbert's approach: from the (non recursive) question of derivability, i.e. the existence of a derivation within a formal system, the interest can be turned to the question of the inner structure of derivations (subformula, analyticity). At the same time, the study of the *construction* of proofs can be combined with the study of their possible *transformations* (a confrontation of these two viewpoints in proof theory constitutes the *leitmotif* of chapter (3)).

A fundamental remark, made independently by Curry [CF58] and later by Howard [How80], was at the basis of the discovery of a strict connection between structural proof-theory and computer science: they observed that Gentzen transformations behaved exactly in the same way as normalization in λ-calculus. In particular, it was shown that derivations in intuitionistic logic could be interpreted as programs in λ-calculus, and their transformations as the execution of those programs. This connection, known under the name of *Curry-Howard correspondence*, constitutes still today one of the most powerful tools in proof-theory, that will be discussed and exploited throughout the following pages.

Logics as categories The presentation of logic which comes from the development of proof-theory is essentially threefold: one has *formulae*, *derivations* (of formulae) and *transformations* (of derivations). This partition is indeed the starting point of the semantical approach to proofs, *denotational semantics* (for an introduction, see for instance [AL91]): the idea of a semantics of proofs comes directly from Gentzen's *Hauptsatz*; indeed, it is natural to think of the *denotation* of a proof as an *invariant* of the cut-elimination procedure.

Usually these semantics are presented in a categorial setting: formulae A, B are interpreted as *objects* $\mathbf{A}, \mathbf{B}$ of a certain category $\mathcal{C}$ (for instance *Scott domains* or *coherent spaces*); a derivation

d of $A \vdash B$ is interpreted as a *morphism* $\mathbf{d} \in \mathcal{C}[\mathbf{A}, \mathbf{B}]$; remark that, for every formula A, there exists a trivial derivation of $A \Rightarrow A$ (corresponding to the identity morphism $id_{\mathbf{A}}$). Finally, given two derivations d, e, respectively in $A \vdash B$ and $B \vdash C$, a cut between them is interpreted by the *composition* $d \circ e$ of the two, and the transformations over derivations correspond to the identities expressed by the *diagrams* in the category: this expresses the fact that the denotation of a derivation $\mathbf{d}$ is invariant under cut-elimination.

As we not are going to deal with denotational semantics in detail in this text, the categorical presentation will be left at an informal level. Nevertheless, the choice to adopt this categorical intuition[1] in the following pages is motivated by the fact that it provides a very elegant way to present the relationship between different logics. Indeed, once logics are thought in categorical terms, the fact that, in passing from a logic $\mathbf{C}$ to a logic $\mathbf{D}$, the proof-theoretic content is preserved can be expressed as the existence of a *functorial* translation $\mathcal{C} \stackrel{\mathbb{J}}{\to} \mathcal{D}$ from the first to the second logic; this means that one has indeed two maps:

- a map $A \mapsto A^{\mathbb{J}}$ from the formulae of $\mathcal{C}$ to the formulae of $\mathcal{D}$;
- for all formulae $A_1, \dots, A_n, B$ of $\mathcal{C}$, a map $f \mapsto \mathbb{J}(f)$ from the derivations of $A_1, \dots, A_n \vdash B$[2] to those of $A_1^{\mathbb{J}}, \dots, A_n^{\mathbb{J}} \vdash B^{\mathbb{J}}$ such that for all A, B, C objects of $\mathcal{C}$ the following hold:
  - $\mathbb{J}(id_A) = id_{A^{\mathbb{J}}}$;
  - for all d, e derivations respectively of $A_1, \dots, A_n \vdash B$ and $B, C_1, \dots, C_m \vdash D$, $\mathbb{J}(f \circ g) = \mathbb{J}(f) \circ \mathbb{J}(g)$.

The typical way to show the functoriality of a translation is to prove that the translation preserves Gentzen's transformations: if a derivation d in $\mathcal{C}$ reduces to a derivation d' by applying some transformations, then its translation $d^{\mathbb{J}}$ reduces to $d'^{\mathbb{J}}$ by applying some Gentzen's transformations in $\mathcal{D}$. In particular this implies that a cut-free derivation of the form $d^{\mathbb{J}}$ comes from a cut-free derivation d in $\mathcal{C}$. In definitive, a functor between two "logics" corresponds to a translation of formulae and derivations *which preserves the reduction relation between derivations.*

Intuitionistic *vs* classical second order logic The following two sections will be devoted to show the equivalence between three different "second order logics", thus showing that three apparently distinct approaches to second order logic share the same proof-theoretical content; these are:

- Second order (intuitionistic/classical) arithmetics $\mathbf{HA}^2$ ($\mathbf{PA}^2$);
- Second order (intuitionistc/classical) logic $\mathbf{LJ}^2$ ($\mathbf{LK}^2$);
- Second order type theory, also known as *polymorphic lambda calculus* or simply *System F*.

In the next pages (and in all the rest of the text) we will make use of *classical* formalism only when discussing completeness, as related to model-theoretic aspects. In all other cases *intuitionist* formalisms will be preferred for purely pragmatical motivations: the *forgetful translation* (see (2.3)) between sequent calculus and type theory is much easier to present and discuss in the intuitionistic fragment (indeed the *Curry-Howard correspondence*, see below, was originally based on intuitionistic logic). Nevertheless, many extensions of this correspondence to the classical

[1] In several places (for instance in [Gir11] and [Dos]) it is advocated that the categorical presentation of logic implies a radical change of viewpoint on the object of logic: with respect to the Fregean viewpoint centered around the notions of sentence and assertion, the categorical approach takes proofs (i.e. morphisms) as logical primitives and sentences (i.e. objects) as derived ones.

[2] The choice of an intuitionistic setting, i.e. of sequents of the form $\Gamma \vdash \Delta$, with $\sharp\Delta \leq 1$, is justified below.

case can be found in the literature (for instance by means of polarization techniques [Gir91], by Parigot's $\lambda\mu$-calculus [Par93] or by the appeal to realizability and control operators [Kri09]).

Clearly important issue of the relationship between classical and intuitionistic logic, with respect to their constructive and recursive content, would demand for an extensive investigation which goes beyond the goals of this thesis. At the same time, by paging through the following chapters, the reader will remark that the questions and challenges raised and discussed in the text concerning second order logic are quite insensitive to the classical/intuitionistic distinction. In particular, both the expressive power and the apparent circularity of second order systems crucially depends on the nature of the comprehension principles admitted[3], so that the switch from an intuitionistic or classical setting leaves most theoretical issues unaltered.

2.1.2 Second order arithmetics and logic

Second order logic The first "logic" is the second order predicate calculus, for which we recall the rules and transformations. Since we are interested in relating this logic with second order arithmetics, the language will include the arithmetical constant $\underline{0}$ and function symbols $\underline{s}, \underline{+}, \underline{\times}$. From a purely logical point of view, these symbols can be seen as arbitrary symbols for, respectively, a 0-ary, a unary and two binary functions.

We first introduce the language of second order logic and then the systems $\mathbf{LJ}^2, \mathbf{LK}^2$ of intuitionistic and classical second order logic.

Definition 2.1.1 ($\mathcal{L}$). *The language $\mathcal{L}$ (with arithmetical symbols) of second order logic is made of the following items:*

- *an individual constant $\underline{0}$, a unary function symbol $\underline{s}$ and two binary function symbols $\underline{+}, \underline{\times}$ and two kinds of variables:*

  i. First-order variables $x_1, x_2, x_3, \dots$ *(also noted $x, y, z, \dots$ when not confusing);*

  ii. Second order variables $X_1, X_2, X_3, \dots$ *of all arities $k \geq 0$ (also noted $X, Y, Z, \dots$ when not confusing).*

- *Terms and formulae of $\mathcal{L}$ defined as follows*

  First-order terms *The set $\mathcal{T}$ of first-order terms is the set of terms $t, u, \dots$ given by the grammar*
  $$t, u := x|\underline{0}|\underline{s}(t)|t\underline{+}u|t\underline{\times}u \tag{2.1.4}$$

  Formulae *The set $\mathcal{F}$ of formulae is the set of expressions $A, B, \dots$ given by the grammar*
  $$X(t_1, \dots, t_k)|A \Rightarrow B|\forall x_i A|\forall X_i A \qquad (t_1, \dots, t_k \in \mathcal{T}) \tag{2.1.5}$$

  Predicates *The set $\mathcal{P}$ of predicates or second order terms is the set of expressions of the form $\lambda x_1. \dots .\lambda x_k.A$, where $A \in \mathcal{F}$ and the variables $x_1, \dots, x_k$ are subject to α-conversion.*

- *A first order notion of substitution, a notion of application for predicates and a second order notion of substitution:*

  first-order subs. $t[u/x]$*, for $t, u \in \mathcal{T}$ and $A[t/x]$, for $t \in \mathcal{T}, A \in \mathcal{F}$, defined as usual;*

[3] A typical example is Friedman's classical result [Fri78] that Π^0_2 provable formulae are intuitionistically provable.

application $\lambda x_1.\ldots.\lambda x_k A(t_1,\ldots,t_h) = \lambda x_{h+1}.\ldots.\lambda x_k.A[t_1/x_1,\ldots,t_k/x_k]$, *for* $h \le k$, $A \in \mathcal{F}, t_1,\ldots,t_k \in \mathcal{T}$;

second-order subs. $X_i(t_1,\ldots t_k)[P/X_i] = A(t_1,\ldots,t_k)$, *for* $P \in \mathcal{P}, A \in \mathcal{F}$ *and* X_i, P *of the same arity.*

In the following by a *sequent* it is meant an expression of the form $\Gamma \vdash A$, where $A \in \mathcal{F}$ and Γ is a finite multiset[4] of formulae.

We introduce the systems of second order logic by defining their formulae, their derivations and the transformations over derivations; the latter are given by introducing, as usual, a reduction relation between derivations.

Definition 2.1.2 (Intuitionistic second order logic $\mathbf{LJ}^2$)**.** *Intuitionistic second order logic is given by the following items:*

Formulae *The formulae of* $\mathbf{LJ}^2$ *are those of the language* $\mathcal{L}$*;*

Derivations *The derivations of* $\mathbf{LJ}^2$ *are built up from the following rules:*

$$\boxed{\begin{array}{cc}
\dfrac{}{A \vdash A}\,(ax) & \dfrac{\Gamma, A \vdash \Delta \quad \Gamma' \vdash A}{\Gamma, \Gamma' \vdash \Delta}\,(cut) \\
\dfrac{\Gamma \vdash B}{\Gamma, A \vdash B}\,(W) & \dfrac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\,(C) \\
\dfrac{\Gamma \vdash A \quad \Gamma', B \vdash \Delta}{\Gamma, \Gamma', A \Rightarrow B \vdash \Delta}\,(\Rightarrow L) & \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow R) \\
\dfrac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\,(\forall L)_x & \dfrac{\Gamma \vdash A}{\Gamma \vdash \forall x A}\,(\forall R)_x, x \notin FV(\Gamma) \\
\dfrac{\Gamma, A[P/X] \vdash \Delta}{\Gamma, \forall X A \vdash \Delta}\,(\forall L)_X & \dfrac{\Gamma \vdash A}{\Gamma \vdash \forall X A}\,(\forall R)_X, X \notin FV(\Gamma)
\end{array}} \tag{2.1.6}$$

where Γ_σ *is any permutation of the order of the formulae occurring in* Γ.

Transformations *The reduction relation* $\preceq$ *of* $\mathbf{LJ}^2$ *is the reflexive-transitive closure of the relation* $\prec$ *generated by the following transformations or* reduction rules*:*

(ax)

$$\dfrac{A \vdash A \quad \begin{array}{c}\vdots\, d \\ \Gamma \vdash A\end{array}}{\Gamma \vdash A}\,(cut) \quad \prec \quad \begin{array}{c}\vdots\, d \\ \Gamma \vdash A\end{array} \tag{2.1.7}$$

(W)

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, d_1 \\ \Gamma \vdash B\end{array}}{\Gamma, A \vdash B}\,(W) \quad \begin{array}{c}\vdots\, d_2 \\ \Gamma' \vdash A\end{array}}{\Gamma, \Gamma' \vdash B}\,(cut) \quad \prec \quad \dfrac{\begin{array}{c}\vdots\, d_1 \\ \Gamma \vdash B\end{array}}{\Gamma, \Gamma' \vdash B}\,(W) \tag{2.1.8}$$

[4]A *multiset* is given by a set S and a multiplicity function, i.e. a map $g : S \to \mathbb{N}$ which assigns a multiplicity to any element of S. Hence a multiset of formulae is a set which can contain several occurrences of the same formula.

(C)

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma, A, A \vdash B\end{array}}{\Gamma, A \vdash B}\,(C) \quad \begin{array}{c}\vdots\, d_2\\ \Gamma' \vdash A\end{array}}{\Gamma, \Gamma' \vdash B}\,(cut) \quad \prec \quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma, A, A\vdash B\end{array} \quad \begin{array}{c}\vdots\, d_2\\ \Gamma'\vdash A\end{array}}{\Gamma,\Gamma',A\vdash B}\,(cut) \quad \begin{array}{c}\vdots\, d_2\\ \Gamma'\vdash A\end{array}}{\Gamma,\Gamma',\Gamma'\vdash B}\,(cut)}{\Gamma,\Gamma'\vdash B}\,(C) \tag{2.1.9}$$

$(\Rightarrow)$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, d_{11}\\ \Gamma_1 \vdash A\end{array} \quad \begin{array}{c}\vdots\, d_{12}\\ \Gamma_2, B \vdash C\end{array}}{\Gamma, A \Rightarrow B \vdash C}\,(\Rightarrow L) \quad \dfrac{\begin{array}{c}\vdots\, d_2\\ \Gamma', A \vdash B\end{array}}{\Gamma' \vdash A \Rightarrow B}}{\Gamma, \Gamma' \vdash B}\,(cut) \quad \prec \quad \dfrac{\dfrac{\begin{array}{c}\vdots\, d_{11}\\ \Gamma_1 \vdash A\end{array} \quad \begin{array}{c}\vdots\, d_2\\ \Gamma', A \vdash B\end{array}}{\Gamma_1, \Gamma' \vdash B}\,(cut) \quad \begin{array}{c}\vdots\, d_{12}\\ \Gamma_2, B \vdash C\end{array}}{\Gamma, \Gamma' \vdash B}\,(cut) \tag{2.1.10}$$

$(\forall x)$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma, A[t/x] \vdash B\end{array}}{\Gamma, \forall x A \vdash B}\,(\forall L)_x \quad \dfrac{\begin{array}{c}\vdots\, d_2\\ \Gamma' \vdash A\end{array}}{\Gamma' \vdash \forall x A}\,(\forall R)_x}{\Gamma, \Gamma' \vdash B}\,(cut) \quad \prec \quad \dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma, A[t/x] \vdash B\end{array} \quad \begin{array}{c}\vdots\, d_2\{t/x\}\\ \Gamma' \vdash A[t/x]\end{array}}{\Gamma, \Gamma' \vdash B}\,(cut) \tag{2.1.11}$$

where $d_2\{t/x\}$ is the derivation obtained by replacing all occurrences of x in d_2 by the term t (remark that this is well defined since x does not appear free in Γ').

$(\forall X)$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma, A[P/X] \vdash B\end{array}}{\Gamma, \forall X A \vdash B}\,(\forall L)_X \quad \dfrac{\begin{array}{c}\vdots\, d_2\\ \Gamma' \vdash A\end{array}}{\Gamma' \vdash \forall X A}\,(\forall R)_X}{\Gamma, \Gamma' \vdash B}\,(cut) \quad \prec \quad \dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma, A[P/X] \vdash B\end{array} \quad \begin{array}{c}\vdots\, d_2\{P/X\}\\ \Gamma' \vdash A[P/X]\end{array}}{\Gamma, \Gamma' \vdash B}\,(cut) \tag{2.1.12}$$

where $d_2\{P/X\}$ is the derivation obtained by replacing all occurrences of X in d_2 by the predicate P (remark that this is well defined since X does not appear free in the formulae in Γ').

$(commL)$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma', A \vdash B'\end{array}}{\Gamma, A \vdash B}\,(R) \quad \begin{array}{c}\vdots\, d_2\\ \Delta \vdash A\end{array}}{\Gamma, \Delta \vdash B}\,(cut) \quad \prec \quad \dfrac{\dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma', A \vdash B'\end{array} \quad \begin{array}{c}\vdots\, d_2\\ \Delta \vdash A\end{array}}{\Gamma', \Delta \vdash B'}\,(cut)}{\Gamma, \Delta \vdash B}\,(R) \tag{2.1.13}$$

where (R) is any rule distinct from the left rule for the principal connective of A.

$(commR)$

$$\dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma, A \vdash B\end{array} \quad \dfrac{\begin{array}{c}\vdots\, d_2\\ \Delta' \vdash A\end{array}}{\Delta \vdash A}\,(R)}{\Gamma, \Delta \vdash B}\,(cut) \quad \prec \quad \dfrac{\dfrac{\begin{array}{c}\vdots\, d_1\\ \Gamma, A \vdash B\end{array} \quad \begin{array}{c}\vdots\, d_2\\ \Delta' \vdash A\end{array}}{\Gamma, \Delta' \vdash B}\,(cut)}{\Gamma, \Delta \vdash B}\,(R) \tag{2.1.14}$$

where (R) is any rule distinct from the right rule for the principal connective of A.

As it is well-known, all other connectives of second order logic can be defined in the language $\mathcal{L}$:

$$\perp/\mathbf{1} := \forall X X/\forall X(X \Rightarrow X) \tag{2.1.15}$$
$$t = u := \forall X(X(t) \Rightarrow X(u)) \tag{2.1.16}$$
$$A \wedge B := \forall X((A \Rightarrow B \Rightarrow X) \Rightarrow X) \tag{2.1.17}$$
$$A \vee B := \forall X((A \Rightarrow X) \Rightarrow (B \Rightarrow X) \Rightarrow X) \tag{2.1.18}$$
$$\exists x A := \forall Y(\forall x(A \Rightarrow Y) \Rightarrow Y) \tag{2.1.19}$$
$$\exists X A := \forall Y(\forall X(A \Rightarrow X) \Rightarrow Y) \tag{2.1.20}$$

Second order *classical* logic $\mathbf{LK}^2$ is obtained by extending the notion of sequent to $\Gamma \vdash \Delta$, where Δ is another multiset of formulae and by considering the following rules (with $\neg A := A \Rightarrow \perp$):

$$\boxed{\begin{array}{cc}
\dfrac{}{A \vdash A}\ (ax) & \dfrac{\Gamma, A \vdash \Delta \quad \Gamma' \vdash A, \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ (cut) \\
\dfrac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ (\perp L) & \dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\ (\perp R) \\
\dfrac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ (WL) & \dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ (WR) \\
\dfrac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ (CL) & \dfrac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ (CR) \\
\dfrac{\Gamma \vdash A, \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \Rightarrow B \vdash \Delta, \Delta'}\ (\Rightarrow L) & \dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}\ (\Rightarrow R) \\
\dfrac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ (\forall L)_x & \dfrac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x A, \Delta}\ (\forall R)_x, x \notin FV(\Gamma) \\
\dfrac{\Gamma, A[P/X] \vdash \Delta}{\Gamma, \forall X A \vdash \Delta}\ (\forall L)_X & \dfrac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall X A, \Delta}\ (\forall R)_X, X \notin FV(\Gamma)
\end{array}} \tag{2.1.21}$$

It is clear from the rules above that any derivation in $\mathbf{LJ}^2$ is also a derivation in $\mathbf{LK}^2$. We do not list the reduction rules for $\mathbf{LK}^2$ (see for instance [ST00]), since in the following we will just consider those of $\mathbf{LJ}^2$.

Second order arithmetics We describe now intuitionistic second order arithmetics, or *Heyting Arithmetics* $\mathbf{HA}^2$. Remark that, whereas $\mathbf{HA}^2$ is usually presented as a *theory* over the language of second order minimal logic, here we present it under the form of a "logic"; in particular, this "logic" is an extension of $\mathbf{LJ}^2$ by some axioms, which can be seen, from the categorial viewpoint, as new morphisms.

Definition 2.1.3 (Language of arithmetics)**.** *The language of arithmetics $\mathcal{L}_A$ is defined as the language $\mathcal{L}$, but first-order variables $x_1, x_2, \ldots$ are replaced by* number variables $n_1, n_2, \ldots$ *and the first-order quantifier $\forall x$ is replaced by the* number quantifier $\forall n$.

Definition 2.1.4 (Heyting Arithmetics)**.** *Heyting arithmetics* $\mathbf{HA}^2$ *is defined as follows:*

Formulae *The formulae of* $\mathbf{HA}^2$ *are those of* $\mathcal{L}_A$*;*

Derivations *The derivations of* $\mathbf{HA}^2$ *are built up by the rules of* $\mathbf{LJ}^2$ *(where individual variables* $x, y, \dots$ *are replaced by number variables* $n, m, \dots$ *) plus the following axioms*

$$\underline{0} = \underline{s}(n) \vdash \tag{$PA1$}$$

$$\underline{s}(n) = \underline{s}(m) \vdash n = m \tag{$PA2$}$$

$$\vdash \forall X(\forall m(X(m) \Rightarrow X(\underline{s}(m))) \Rightarrow (X(\underline{0}) \Rightarrow \forall n X(n))) \tag{$PA3$}$$

$$\vdash n \underline{+} \underline{0} = n \qquad \vdash n \underline{+} \underline{s}(m) = \underline{s}(n \underline{+} m) \tag{$PA+_{1,2}$}$$

$$\vdash n \underline{\times} \underline{0} = \underline{0} \qquad \vdash n \underline{\times} \underline{s}(m) = (n \underline{\times} m) \underline{+} m \tag{$PA\times_{1,2}$}$$

Transformations *The reduction rules of* $\mathbf{HA}^2$ *are just the reduction rules of* $\mathbf{LJ}^2$*.*

Again, all other connectives can be defined as above for the language of $\mathbf{HA}^2$ (with number variable replacing individual variables). *Second order Peano Arithmetics* $\mathbf{PA}^2$ is the system obtained by replacing $\mathbf{LJ}^2$ by $\mathbf{LK}^2$ in the definition of $\mathbf{HA}^2$.

2.1.3 System F

The second order typed λ-calculus was introduced independently by Girard in [Gir72] (under the name of *System F*) and by Reynolds in [Rey74] (under the name of *polymorphic λ-calculus*). In the following we retain Girard's terminology for simplicity.

System F has a second order language for types made of type variables $\alpha, \beta, \dots$, a constant $\rightarrow$ to build implication types and a universal quantifier $\forall$. Hence the set of types **Typ** can be defined by the grammar below:

$$\sigma, \tau := \alpha | \sigma \rightarrow \tau | \forall \alpha \sigma \tag{2.1.22}$$

The original formulations of System F are *à la Church*: this means that the λ-terms (that we note $M, N, \dots$) are defined with type superscripts. For any type σ, one has a countable set of variables $x^\sigma, y^\sigma, \dots$ of type σ. One has the usual rules for building simply typed λ-terms (see [BAGM92]):

abstraction given a term M^τ and a variable x^σ one can form the term $(\lambda x^\sigma . M^\tau)^{\sigma \rightarrow \tau}$;

application given two terms of the form $M^{\sigma \rightarrow \tau}, N^\sigma$ one can form the term $\big((M^{\sigma \rightarrow \tau}) N^\sigma\big)^\tau$.

The novelty introduced with System F is the possibility to abstract over type variables, given by the following rules

type abstraction given a term M^σ and a type variable α one can form the term $(\Lambda \alpha . M^\sigma)^{\forall \alpha \sigma}$;

type extraction given a term $M^{\forall \alpha \sigma}$ and a type τ one can form the term $(M^{\forall \alpha \sigma}\{\tau\})^{\sigma[\tau/\alpha]}$.

The extraction construction tells that from a term of type $\forall \alpha \sigma$ a term of type $\sigma[\tau/\alpha]$, for *any* type τ (included $\forall \alpha \sigma$), can be extracted. This is what introduces in this typed λ-calculus the circularity which is typical of second order logic (also known as impredicativity, see chapter (4)). Moreover, this construction allows to type λ-terms containing variables applying to themselves: a variable $x^{\forall \alpha \alpha}$ can be extracted on the two types $\alpha \rightarrow \alpha$ and α, so that the term below, which is not typable in simple type theory, can be correctly typed in System F:

$$\lambda x^{\forall \alpha \alpha} . (x\{\alpha \rightarrow \alpha\}) x\{\alpha\} \tag{2.1.23}$$

Hence, the rules of type abstraction and type extraction introduce a type discipline which is very far from Russell's original motivations for introducing types (that is, avoiding auto-applications).

We introduce below in more detail a version *à la Curry* of System F, that will be used throughout the text: this means that one takes as terms the terms of pure, or untyped, λ-calculus and defines the rules of System F as *typing rules*, i.e. rules for assigning a type to such terms.

The presentation *à la Curry* highlights the *polymorphic* (etymologically, having many forms, many types) nature of the typed terms of System F: if M is a (pure) λ-term which has type $\forall\alpha\sigma$, then the same term M must have type $\sigma[\tau/\alpha]$, for any type τ. The acutal nature of this polymorphism, and the paradoxes related with it, are discussed in chapters (5) and (6).

The basic objects of *á la Curry* systems are λ-terms and *typing judgements*, i.e. sequents of the form $\Gamma \vdash M : \sigma$, which intuitively assert that M is a term of type σ under the assumptions Γ.

Definition 2.1.5 (System F). • *We define the "language" of system F by introducing* terms, types *and* judgements*:*

terms *The terms of system F are usual pure lambda terms, generated by the grammar*

$$M, N := x|\lambda x.M|(M)N \tag{2.1.24}$$

given a countable set of term variables $x, y, z, ...$[5] *and considered up to α-equivalence. For a detailed introduction to the λ-calculus see for instance [Bar85].*

types *the types of F are given by the set* **Typ***; the sets $FV(\sigma)$ and $BV(\sigma)$ of, respectively,* free *and* bound *type variables of a type σ are defined as follows:*

$$\begin{array}{cc} FV(\alpha) = \{\alpha\} & BV(\alpha) = \emptyset \\ FV(\sigma \to \tau) = FV(\sigma) \cup FV(\tau) & BV(\sigma \to \tau) = BV(\sigma) \cup BV(\tau) \\ FV(\forall\alpha\sigma) = FV(\sigma) - \{\alpha\} & BV(\forall\alpha\sigma) = BV(\sigma) \cup \{\alpha\} \end{array} \tag{2.1.25}$$

A substitution *operation over types is defined by*

$$\alpha[\sigma/\beta] = \begin{cases} \sigma & \textit{if } \alpha = \beta \\ \alpha & \textit{else} \end{cases} \tag{2.1.26}$$

$$\tau \to \rho[\sigma/\beta] = \tau[\sigma/\beta] \to \rho[\sigma\beta] \tag{2.1.27}$$

$$(\forall\alpha\tau)[\sigma/\beta] := \forall\alpha(\tau[\sigma/\beta]) \tag{2.1.28}$$

where substitution is defined, as in λ-calculus, as to avoid variable bindings.

declarations *a* type declaration *is an expression of the form $(x : \sigma)$, where x is a term variable and σ is a type. A* context Γ *is a finite set of type declarations*[6].

judgements *a* judgement *is an expression of the form $\Gamma \vdash M : \sigma$, where Γ is a context, M a term and σ a type.*

• *The typing derivations of system F are generated by the following rules:*

$$\boxed{\begin{array}{cc} \dfrac{}{\Gamma, (x : \sigma) \vdash x : \sigma}\ (id) & \\ \dfrac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (M)N : \tau}\ (\to E) & \dfrac{\Gamma, (x : \sigma) \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\to I) \\ \dfrac{\Gamma \vdash M : \forall\alpha\sigma}{\Gamma \vdash M : \sigma[\tau/\alpha]}\ (\forall E) & \dfrac{\Gamma \vdash M : \sigma \quad \alpha \textit{ bindable in } \Gamma}{\Gamma \vdash M : \forall\alpha\sigma}\ (\forall I) \end{array}} \tag{2.1.29}$$

[5] We adopt the same notation x, y, z for term variables and individual variables, unless confusing; this abuse will be indeed exploited in section (2.3).

[6] Unless confusing, we use the same notation Γ for contexts of type declaration and contexts of sequent calculus. Remark anyway that, whereas contexts of formulae are *multisets*, context of type declarations are *sets*.

where α is bindable in Γ if, for all type declaration $(x : \sigma) \in \Gamma$, α is not free in σ.

We distinguish two equality relations over types: $\sigma \equiv \tau$ denotes syntactic equality whereas $\sigma = \tau$ denotes α-equivalence.

We introduce an order relation over types, $\sigma \preceq \tau$, which is the reflexive transitive closure of the relation

$$\forall\alpha\sigma \prec \tau \qquad \Leftrightarrow \qquad \tau = \sigma[\rho/\alpha] \tag{2.1.30}$$

We recall some simple properties (whose proof can be found for instance in [BAGM92]):

Proposition 2.1.1 (basic properties)**.** *i. If $\Gamma \vdash M : \sigma$ is derivable, then $\Gamma' \vdash M : \sigma$, with $\Gamma \subseteq \Gamma'$, is derivable;*

ii. If $\Gamma \vdash M : \sigma$ is derivable, then, if $x \in FV(M)$, $(x : \tau) \in \Gamma$, for some type σ;

iii. If $\Gamma \vdash x : \sigma$ is derivable, then $(x : \sigma') \in \Gamma$ for some σ' such that $\sigma' \preceq \sigma$;

iv. If $\Gamma \vdash M : \sigma$ is derivable and M' is a subterm of M, then $\Gamma \vdash M' : \tau$ is derivable for some τ.

In system F we do not have a reduction relation over typing derivations, but only over lambda terms: *reduction* $M \to_\beta N$ is defined (as in pure λ-calculus) as the reflexive transitive closure of the relation $\to_1$ defined by

$$(\lambda x.M)N \to_1 M[N/x] \tag{2.1.31}$$

We recall two important lemmas that related the type structure with the reduction of the λ-terms. The first lemma tells that a type declaration $(x : \sigma)$ in a typing of a term M can always be replaced with the typing of a term N of type σ, by replacing every occurrence of x in M by the term N.

Lemma 2.1.1 (substitution lemma)**.** *If $\Gamma, (x : \sigma) \vdash M : \tau$ and $\Gamma \vdash N : \sigma$ are derivable, then $\Gamma \vdash M[N/x] : \tau$ is derivable.*

Proof. See [BAGM92]. □

The lemma below shows that the typing derivations are preserved under term reduction:

Lemma 2.1.2 (subject reduction)**.** *If $\Gamma \vdash M : \sigma$ is derivable in F and $M \to M'$, then $\Gamma \vdash M' : \sigma$ is derivable in F.*

Proof. See [BAGM92]. □

Remark that the subject reduction property is, in a certain sense, the equivalent in type theory of Prawitz's *inversion principle*: in the same way in which the latter provides a way to define a transformation over a derivation containing a cut, the former enables the reduction of a redex preserving the type structure of the term.

2.2 The Dedekind functor

2.2.1 "Was sind und was sollen die zahlen"

The logicist dream was that of a purely logical definition of arithmetical (and analytical) concepts. In his famous 1888 paper [Ded96], Dedekind explicitly writes:

> In speaking of arithmetics (algebra, analysis) as merely a part of logic I imply that I consider the number-concept entirely independent of the notions or intuitions of space and time - that I rather consider it an immediate product of the pure laws of thought. [Ded96]

In that paper the logical definition of the natural numbers made indeed its first appearance: Dedekind defined an "object" to be a natural number if it belongs to the intersection of *all* the "chains", i.e. of all the sets A containing an element $\underline{0}$ and closed under an injective function $\underline{s}(x)$. Once translated in the common language of second order logic, Dedekind's definition amounts to the introduction of the second order predicate $N(x)$ below:

$$N(x) := \forall X(\forall y(X(y) \Rightarrow X(\underline{s}(y))) \Rightarrow (X(\underline{0}) \Rightarrow X(x))) \tag{2.2.1}$$

The purely logical nature of its definition comes from the fact that it does not depend on an intended interpretation of the symbols $\underline{0}$ and $\underline{s}$:

> If in the consideration of a simply infinite system N ordered by a map ϕ we entirely neglect the special character of the elements, simply retaining their distinguishability and taking into account only the relations to one another in which they are placed by the ordering mapping ϕ, then these elements are called *natural numbers* or *ordinal numbers* or simply *numbers*, and the base element 1 is called the *base-number* of the *number-series* N. [Ded96]

In particular Dedekind was able to show that all instances of the induction schema were derivable from his definition and to prove the *isomorphism theorem*, which basically asserts that, provided that $\underline{0}$ is interpreted as a base element and $\underline{s}$ as an injective function, then all possible interpretations of the set $\{x|N(x)\}$ are isomorphic to $\mathbb{N}$, the set of natural numbers (what we call today a *categoricity* theorem - see [BBJ07]).

Far from the philosophical ambitions of the logicist program, in this section we develop Dedekind's idea of translating arithmetics into second order logic under the form of a functorial translation $\mathbb{D}$ (that we abusively call *Dedekind functor*). The idea of this translation is quite standard in the literature and amounts to relativize quantification in second order logic to Dedekind's predicate: for instance, arithmetical formulae of the form $\forall nA$ are translated as $\forall x(N(x) \Rightarrow A')$ and formulae of the form $\exists nA$ are translated as $\exists x(N(x) \wedge A')$.

The essence of Dedekind's translation is that all derivations in arithmetics of a sequent $\Gamma \vdash A$ can be translated into derivations in second order logic of the sequent $PA_1, PA_2, \Gamma^{\mathbb{D}} \vdash A^{\mathbb{D}}$, where PA_1, PA_2 are the two sentence expressing respectively the fact that $\underline{0}$ is a base element and that $\underline{s}(x)$ is injective (corresponding indeed to the first two axioms of *Peano Arithmetics*):

$$\forall x(\underline{0} \neq \underline{s}(x)) \tag{PA_1}$$
$$\forall x \forall y(\underline{s}(x) = \underline{s}(y) \Rightarrow x = y) \tag{PA_2}$$

The presentation we give of this translation allows to show the preservation of Gentzen's transformations. The interest of this aspect is twofold: on the one hand it allows, as it will be shown in the next section, to devise a complete cut-elimination procedure for arithmetics, since we no more need to make use of induction axioms (which are replaced by occurrences of Dedekind's predicate). Indeed, it is well-known (see for instance [Pra71b]) that cut-elimination for arithmetics fails when induction axioms are applied to terms containing parameters: the translation in second order logic makes it possible to remove those cuts.

On the other hand, as it will be recalled in section (2.3), the translation of arithmetics into second order logic allows a direct implementation of the *Curry-Howard correspondence* to arithmetics, with very elegant results: a derivation of $N(\underline{n})$ is translated into a program corresponding to the *Church's numeral* $\lambda f.\lambda x.(f)^n x$, and a derivation making use of an induction axiom is translated into a term implementing primitive recursion over a certain (not necessarily finite) type .

2.2.2 The functor $\mathbb{D}$

We describe here a functorial translation of second order (intuitionistic) arithmetics into second order (intuitionistc)[7] logic arising from Dedekind's intuition of a second order treatment of arithmetical concepts.

This idea of a purely logical treatment of arithmetical concepts can be described as follows (this idea was developed in many places, for instance in [Lei83]): given an arithmetical formula A derivable in arithmetics, let us consider its *signature* Σ, i.e. the set of all the constant and function symbols which occur in the derivation; Σ will contain the symbols $\underline{0}$ and $\underline{s}$, so as a finite number of function symbols $f_1, \dots, f_k$.

The "meaning" of those symbols is characterized by a finite set of formulae $\Delta_\Sigma = \{E_1, \dots, E_n\}$; for instance, the "meaning" of the symbols $\underline{0}$ and $\underline{s}$ is fixed by formulae in the axioms PA_1[8] and PA_2, and the meaning of the symbols $\underline{+}$ and $\underline{\times}$ is fixed by the axioms PA_+1-2 and $PA_\times 1-2$; more generally, since any recursive function f can be defined by a finite set of equations (by the so-called *Herbrand-Gödel-Kleene computability* [Kle52]), we let pure logic talk about f by introducing in its language function symbols $f, g_1, \dots, g_k$ for the functions which occur in the equations defining f and by putting such equations in the antecedent of each sequent.

A second element to be considered is the free occurrence of *parameters*: if a free variable x occurs in a formula, then, in logic, we have to make explicit the assumption that the variable x stands for an (unknown) natural number. As a consequence, in our translation we'll have to add assumptions declaring all freely occurring variables to stand for natural numbers.

The logical translation of a derivation of $\Gamma \vdash A$ in $\mathbf{HA}^2$ is a derivation in second order logic of the sequent $N(x_1), \dots, N(x_n), \Delta_\Sigma, \Gamma^\mathbb{D} \vdash A^\mathbb{D}$, where $x_1, \dots, x_n$ are the free parameters occurring in A and $[\_]^\mathbb{D}$ indicates the Dedekind translation of arithmetical formulae.

Formulas Let $N(x)$ be Dedekind's predicate

$$\forall X(\forall y(X(y) \Rightarrow X(\underline{s}(y))) \Rightarrow (X(\underline{0}) \Rightarrow X(x))) \tag{2.2.2}$$

Dedekind's translation from the formulas of $\mathbf{HA}^2$ to the formulas of LM^2 is given by the relativization of the universal quantifier $\forall n$ to Dedekind's predicate: let, for all term $t \in \mathbf{T}_A$, $t^\mathbb{D}$ be the result of replacing in t every occurrence of a number variable n_i with the individual variable x_i. We put then:

$$(X(t_1, \dots, t_n))^\mathbb{D} := X(t_1^\mathbb{D}, \dots, t_n^\mathbb{D}) \qquad (A \Rightarrow B)^\mathbb{D} := A^\mathbb{D} \Rightarrow B^\mathbb{D} \tag{2.2.3}$$

$$(\forall n_i A)^\mathbb{D} := \forall x_i(N(x_i) \Rightarrow A^\mathbb{D}) \qquad (\forall X A)^\mathbb{D} := \forall X A^\mathbb{D} \tag{2.2.4}$$

Dedekind's isomorphism theorem can now be restated in the following form:

Theorem 2.2.1. *Let A be an arithmetical formula. Then $A^\mathbb{D}$ is valid if and only if A is true in the standard model.*

Proof. See for instance [BBJ07]. □

[7] The reader will be easily convinced that this translation is actually independent from the choice of an intuitionistic or classical frame.

[8] Actually by the formula $\neg(\underline{0} = \underline{s}(x))$. This is indeed the only formula in Δ_Σ which is not an equation. The occurrence of negation in this formula has some delicate consequences for the translation in type theory, see subsection (2.3.2).

Derivations We show how to translate a derivation d of $\Gamma \vdash A$ in $\mathbf{HA}^2$ into a derivation $d^{\mathbb{D}}$ of $\Delta, \Gamma^{\mathbb{D}} \vdash A^{\mathbb{D}}$ in LM^2, where Δ may contain the axioms PA_1, PA_2, equations defining the function symbols occurring in $d^{\mathbb{D}}$ and assumptions of the form $N(x)$ for the free variables occurring in Γ, A.

The only cases to consider are the identity rules, the rules for the number-theoretic quantifiers and the axioms $(PA1)$.

(id) The axiom $\forall n_i A \vdash \forall n_i A$ is translated into the axiom $\forall x_i(N(x_i) \Rightarrow A^{\mathbb{D}}) \vdash \forall x_i(N(x_i) \Rightarrow A^{\mathbb{D}})$.

$(\forall R)_n$ Let d be the derivation

$$\dfrac{\begin{array}{c}\vdots\, d' \\ \Gamma \vdash A\end{array}}{\Gamma \vdash \forall n_i A}\,(\forall R)_{n_i} \tag{2.2.5}$$

then, by applying the induction hypothesis to the subderivation d', we define $d^{\mathbb{D}}$ as

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\, (d'^{\mathbb{D}}) \\ \Delta, \Gamma^{\mathbb{D}} \vdash A^{\mathbb{D}}\end{array}}{\Delta, \Gamma^{\mathbb{D}}, N(x_i) \vdash A^{\mathbb{D}}}\,(W)}{\Delta, \Gamma^{\mathbb{D}} \vdash N(x_i) \Rightarrow A^{\mathbb{D}}}}{\Delta, \Gamma^{\mathbb{D}} \vdash \forall x_i(N(x_i) \Rightarrow A^{\mathbb{D}})}\,(\forall R)_{x_i} \tag{2.2.6}$$

$(\forall L)_n$ Before defining the translation we describe, for all term $t \in \mathcal{T}$, its *number derivation* d_t, of conclusion $\Gamma, N(x_1), \ldots, N(x_m) \vdash N(t)$, where $x_1, \ldots, x_m$ are the free variables of t and Γ contains the equational axioms defining the function symbols occurring in t. We build d_t by induction on t:

- if $t = x_i$, then d_t is the axiom $N(x_i) \vdash N(x_i)$;
- if $t = \underline{0}$, then d_t is

$$\dfrac{\dfrac{\dfrac{X(\underline{0}) \vdash X(\underline{0})}{\forall y(X(y) \Rightarrow X(\underline{s}(y))), X(\underline{0}) \vdash X(\underline{0})}\,(W)}{\vdash \forall y(X(y) \Rightarrow X(\underline{s}(y))) \Rightarrow (X(\underline{0}) \Rightarrow X(\underline{0}))}\,(\Rightarrow R)}{\vdash N(\underline{0})}\,(\forall R)_X \tag{2.2.7}$$

- if $t = \underline{s}(t')$, then $FV(t) = FV(t')$ and d_t is (the cut-free derivation obtained from)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\, d_{t'} \\ \Delta, N(x_1), \ldots, N(x_m) \vdash N(t')\end{array} \quad \dfrac{\dfrac{\dfrac{X(\underline{0}) \vdash X(\underline{0}) \quad X(t') \vdash X(t')}{X(\underline{0}) \Rightarrow X(t'), X(\underline{0}) \vdash X(t')}\,(\Rightarrow L) \quad \forall y(X(y) \Rightarrow X(\underline{s}(y)) \vdash \forall y(X(y) \Rightarrow X(\underline{s}(y))}{\forall y(X(y) \Rightarrow X(\underline{s}(y)) \Rightarrow (X(\underline{0}) \vdash X(t')), \forall y(X(y) \Rightarrow X(\underline{s}(y)), X(\underline{0}) \vdash X(t')}\,(\Rightarrow L)}{N(t'), \forall y(X(y) \Rightarrow X(\underline{s}(y)), X(\underline{0}) \vdash X(t'))}\,(\forall L)_X}{\Delta, N(x_1), \ldots, N(x_m), \forall y(X(y) \Rightarrow X(\underline{s}(y)), X(\underline{0}) \vdash X(t'))}\,(cut) \quad \dfrac{X(t) \vdash X(t) \quad X(\underline{s}(t)) \vdash X(\underline{s}(t))}{X(t) \Rightarrow X(\underline{s}(t)), X(t) \vdash X(\underline{s}(t))}\,(\Rightarrow L)}{\Delta, N(x_1), \ldots, N(x_m), \forall y(X(y) \Rightarrow X(\underline{s}(y))), X(t') \Rightarrow X(\underline{s}(t)), X(\underline{0}) \vdash X(\underline{s}(t'))}\,(cut)}{\Delta, N(x_1), \ldots, N(x_m), \forall y(X(y) \Rightarrow X(\underline{s}(y))), \forall y(X(y) \Rightarrow X(\underline{s}(y))), X(\underline{0}) \vdash X(\underline{s}(t'))}\,(\forall L)_y}{\Delta, N(x_1), \ldots, N(x_m), \forall y(X(y) \Rightarrow X(\underline{s}(y))), X(\underline{0}) \vdash X(\underline{s}(t'))}\,(C)}{\Delta, N(x_1), \ldots, N(x_m) \vdash \forall y(X(y) \Rightarrow X(\underline{s}(y))) \Rightarrow (X(\underline{0}) \Rightarrow X(\underline{s}(t')))}\,(\Rightarrow R)}{\Delta, N(x_1), \ldots, N(x_m) \vdash N(\underline{s}(t))}\,(\forall R)_X \tag{2.2.8}$$

- if $t = t_1 \underline{+} t_2$, then $FV(t) = FV(t_1) \cup FV(t_2)$ and d_t is (the cut-free derivation obtained from)

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\, d_{+} \\ \Delta_{+} \vdash \forall y \forall z(N(y) \Rightarrow N(z) \Rightarrow N(y \underline{+} z))\end{array} \quad \dfrac{\dfrac{N(t_1) \vdash N(t_1) \quad N(t_2) \vdash N(t_2) \quad N(t_1 \underline{+} t_2) \vdash N(t_1 \underline{+} t_2)}{N(t_1) \Rightarrow N(t_2) \Rightarrow N(t_1 \underline{+} t_2), N(t_1), N(t_2) \vdash N(t_1 \underline{+} t_2)}\,(\Rightarrow L)}{\forall y \forall z(N(y) \Rightarrow N(z) \Rightarrow N(y \underline{+} z)), N(t_1), N(t_2) \vdash N(t_1 \underline{+} t_2)}\,(\forall L)_{y,z}}{\Delta_{+}, N(t_1), N(t_2) \vdash N(t_1 \underline{+} t_2)}\,(cut) \quad \begin{array}{c}\vdots\, d_{t_1} \\ N(x_1), \ldots, N(x_m) \vdash N(t_1)\end{array}}{\Delta_{+}, N(x_1), \ldots, N(x_m), N(t_2) \vdash N(t_1 \underline{+} t_2)}\,(cut) \quad \begin{array}{c}\vdots\, d_{t_2} \\ N(x_{m+1}), \ldots, N(x_{m+p}) \vdash N(t_2)\end{array}}{\Delta_{+}, N(x_1), \ldots, N(x_{m+p}) \vdash N(t_1 \underline{+} t_2)} \tag{2.2.9}$$

where Δ_+ contains PA_1, PA_2 and the equality axioms defining addition

$$x\underline{+}\underline{0} = x \qquad\qquad x\underline{+}\underline{s}(y) = \underline{s}(x\underline{+}y) \tag{2.2.10}$$

and d_+ is a derivation of the totality of the sum (see the next subsection for a discussion).

- if $t = t_1\underline{\times}t_2$, then $FV(t) = FV(t_1) \cup FV(t_2)$ and d_t is built as in the case above, with $d_\times$ replacing d_+, where $d_\times$ is a derivation of the totality of the product (again, see the next subsection), with context $\Delta_\times$ made of PA_1, PA_2, the equality axioms of addition and the equality axioms below

$$x\underline{\times}\underline{0} = \underline{0} \qquad\qquad x\underline{\times}\underline{s}(y) = (x\underline{\times}y)\underline{+}x \tag{2.2.11}$$

We can now describe the translation of the $(\forall L)_n$ rule: let d be the derivation

$$\dfrac{\begin{matrix}\vdots\, d' \\ \Gamma, A(t) \vdash B\end{matrix}}{\Gamma, \forall n_i A \vdash B}\,(\forall L)_{n_i} \tag{2.2.12}$$

then, by applying the induction hypothesis to the subderivation d', we define $d^{\mathbb{D}}$ as

$$\dfrac{\dfrac{\begin{matrix}\vdots\, d'^{\mathbb{D}} \\ \Delta_2, \Gamma^{\mathbb{D}}, A^{\mathbb{D}}(t) \vdash B^{\mathbb{D}}\end{matrix} \quad \begin{matrix}\vdots\, d_t \\ \Delta_1, N(x_1), \dots, N(x_m) \vdash N(t)\end{matrix}}{\Delta, N(x_1), \dots, N(x_m), \Gamma^{\mathbb{D}}, N(t) \Rightarrow A^{\mathbb{D}}(t) \vdash B^{\mathbb{D}}}\,(\Rightarrow L)}{\Delta, N(x_1), \dots, N(x_m), \Gamma^{\mathbb{D}}, \forall x_i(N(x_i) \Rightarrow A^{\mathbb{D}}(x_i)) \vdash B^{\mathbb{D}}}\,(\forall L)_{x_i} \tag{2.2.13}$$

(PA_1/PA_2) The axioms PA_1 and PA_2 are translated into the trivial derivations of $PA_1, PA_2 \vdash PA_1$ and $PA_1, PA_2 \vdash PA_2$.

(PA_3) The axiom PA_3 is translated into the derivation d_{IND} below

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall y(X(y) \Rightarrow X(\underline{s}(y))) \vdash \forall y(X(y) \Rightarrow X(\underline{s}(y))) \quad X(\underline{0}) \vdash X(\underline{0}) \quad X(x) \vdash X(x)}{\forall y(X(y) \Rightarrow X(\underline{s}(y))), X(\underline{0}), (\forall y(X(y) \Rightarrow X(\underline{s}(y)))) \Rightarrow (X(\underline{0}) \Rightarrow X(x)) \vdash X(x)}\,(\Rightarrow L)}{\forall y(X(y) \Rightarrow X(\underline{s}(y))), X(\underline{0}), N(x) \vdash X(x)}\,(\forall L)_X}{(\forall y(X(y) \Rightarrow X(\underline{s}(y)))) \Rightarrow (X(\underline{0}) \Rightarrow \forall x(N(x) \Rightarrow X(x)))}\,(\Rightarrow W)}{\vdash \forall X((\forall y(X(y) \Rightarrow X(\underline{s}(y)))) \Rightarrow (X(\underline{0}) \Rightarrow \forall x(N(x) \Rightarrow X(x))))}\,(\forall R)_X}{PA_1, PA_2 \vdash \forall X((\forall y(X(y) \Rightarrow X(\underline{s}(y)))) \Rightarrow (X(\underline{0}) \Rightarrow \forall x(N(x) \Rightarrow X(x))))}\,(W) \tag{2.2.14}$$

Reductions We show now that, for all derivations d, d' in $\mathbf{HA}^2$, if d reduces to d', then $d^{\mathbb{D}}$ reduces to $d'^{\mathbb{D}}$ in LM^2. We show this by induction on the translation of rules defined above. Since the case of the identity rule is trivial, we discuss the case of a cut $(\forall L)_n/(\forall R)_n$; moreover, we must add the case of the irreducible cut $(\forall L)_n/PA_3$: since the axiom PA_3 is translated into a derivation, it follows that this irreducible cut is translated into a reducible one.

$(\forall L)/(\forall R)$ let d be the following derivation

$$\dfrac{\dfrac{\begin{matrix}\vdots\, d_1 \\ \Gamma, A(t) \vdash B\end{matrix}}{\Gamma, \forall n_i A \vdash B}\,(\forall L)_{n_i} \qquad \dfrac{\begin{matrix}\vdots\, d_2 \\ \Gamma' \vdash A\end{matrix}}{\Gamma' \vdash \forall n_i A}\,(\forall R)_{n_i}}{\Gamma, \Gamma' \vdash B}\,(cut) \tag{2.2.15}$$

which reduces in one step to

$$\dfrac{\begin{matrix}\vdots\; d_1 \\ \Gamma, A(t) \vdash B\end{matrix} \quad \begin{matrix}\vdots\; d_2\{t/x\} \\ \Gamma' \vdash A(t)\end{matrix}}{\Gamma, \Gamma' \vdash B}\,(cut) \tag{2.2.16}$$

The derivation $d^{\mathbb{D}}$ is the following:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\; d_1^{\mathbb{D}}\\ \Delta_{11},\Gamma^{\mathbb{D}},A^{\mathbb{D}}(t^{\mathbb{D}})\vdash B^{\mathbb{D}}\end{matrix}\quad \begin{matrix}\vdots\; d_{t^{\mathbb{D}}}\\ \Delta_{12},N(x_1),\dots,N(x_m)\vdash N(t^{\mathbb{D}})\end{matrix}}{\Delta_1,N(x_1),\dots,N(x_m),\Gamma^{\mathbb{D}},N(t^{\mathbb{D}})\Rightarrow A^{\mathbb{D}}(t^{\mathbb{D}})\vdash B^{\mathbb{D}}}\,(\Rightarrow L)}{\Delta_1,N(x_1),\dots,N(x_m),\Gamma^{\mathbb{D}},\forall x_i(N(x_i)\Rightarrow A^{\mathbb{D}}(x_i))\vdash B^{\mathbb{D}}}\,(\forall L)_{x_i} \quad \dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\; (d_2^{\mathbb{D}})\\ \Delta_2,\Gamma'^{\mathbb{D}}\vdash A^{\mathbb{D}}\end{matrix}}{\Delta_2,\Gamma'^{\mathbb{D}},N(x_i)\vdash A^{\mathbb{D}}}\,(W)}{\Delta_2,\Gamma'^{\mathbb{D}}\vdash N(x_i)\Rightarrow A^{\mathbb{D}}}}{\Delta_2,\Gamma'^{\mathbb{D}}\vdash\forall x_i(N(x_i)\Rightarrow A^{\mathbb{D}})}\,(\forall R)_{x_i}}{\Delta,N(x_1),\dots,N(x_m),\Gamma^{\mathbb{D}},\Gamma'^{\mathbb{D}}\vdash B^{\mathbb{D}}}\,(cut) \tag{2.2.17}$$

which reduces in two steps to

$$\dfrac{\begin{matrix}\vdots\; d_1^{\mathbb{D}}\\ \Delta_{11},\Gamma^{\mathbb{D}},A^{\mathbb{D}}(t^{\mathbb{D}})\vdash B^{\mathbb{D}}\end{matrix} \quad \dfrac{\begin{matrix}\vdots\; d_{t^{\mathbb{D}}}\\ \Delta_{12},N(x_1),\dots,N(x_m)\vdash N(t^{\mathbb{D}})\end{matrix} \quad \dfrac{\begin{matrix}\vdots\; (d_2^{\mathbb{D}}\{t^{\mathbb{D}}/x_i\})\\ \Delta_2,\Gamma'^{\mathbb{D}}\vdash A^{\mathbb{D}}\end{matrix}}{\Delta_2,\Gamma'^{\mathbb{D}},N(t^{\mathbb{D}})\vdash A^{\mathbb{D}}(t^{\mathbb{D}})}\,(W)}{\Delta_{12},\Delta_2,N(x_1),\dots,N(x_m)\vdash A^{\mathbb{D}}(t^{\mathbb{D}})}\,(cut)}{\Delta,N(x_1),\dots,N(x_m),\Gamma^{\mathbb{D}}\vdash B^{\mathbb{D}}}\,(cut) \tag{2.2.18}$$

and successively to

$$\dfrac{\begin{matrix}\vdots\; d_1^{\mathbb{D}}\\ \Delta_{11},\Gamma^{\mathbb{D}},A^{\mathbb{D}}(t^{\mathbb{D}})\vdash B^{\mathbb{D}}\end{matrix} \quad \dfrac{\begin{matrix}\vdots\; (d'^{\mathbb{D}}\{t^{\mathbb{D}}/x_i\})\\ \Delta_2,\Gamma'^{\mathbb{D}}\vdash A^{\mathbb{D}}\end{matrix}}{\Delta_2,N(x_1),\dots,N(x_m),\Gamma'^{\mathbb{D}}\vdash A^{\mathbb{D}}(t^{\mathbb{D}})}\,(W)}{\Delta,N(x_1),\dots,N(x_m),\Gamma^{\mathbb{D}},\Gamma'^{\mathbb{D}},\vdash B^{\mathbb{D}}}\,(cut) \tag{2.2.19}$$

$(\forall L)/PA_3$ let d be the following irreducible derivation

$$\dfrac{\dfrac{\begin{matrix}\vdots\; d_1\\ \Gamma, A(t)\vdash B\end{matrix}}{\Gamma,\forall n_i A\vdash B}\,(\forall L)_{n_i} \quad \dfrac{\vdash PA_3 \quad \begin{matrix}\vdots\\ PA_3,\forall y(A(y)\Rightarrow A(\underline{s}(y))),A(\underline{0})\vdash\forall n_i A\end{matrix}}{\forall y(A(y)\Rightarrow A(\underline{s}(y))),A(\underline{0})\vdash\forall n_i A}\,(cut)}{\forall y(A(y)\Rightarrow A(\underline{s}(y))),A(\underline{0}),\Gamma\vdash B}\,(cut) \tag{2.2.20}$$

The derivation $d^{\mathbb{D}}$, after some reduction step, is the following:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\; d'^{\mathbb{D}}\\ \Gamma^{\mathbb{D}},A^{\mathbb{D}}(t^{\mathbb{D}})\vdash B^{\mathbb{D}}\end{matrix}\quad\begin{matrix}\vdots\; d_{t^{\mathbb{D}}}\\ N(x_1),\dots,N(x_m)\vdash N(t^{\mathbb{D}})\end{matrix}}{N(x_1),\dots,N(x_m),\Gamma^{\mathbb{D}},N(t^{\mathbb{D}})\Rightarrow A^{\mathbb{D}}(t^{\mathbb{D}})\vdash B^{\mathbb{D}}}\,(\Rightarrow L)}{N(x_1),\dots,N(x_m),\Gamma^{\mathbb{D}},\forall x_i(N(x_i)\Rightarrow A^{\mathbb{D}})(x_i)\vdash B^{\mathbb{D}}}\,(\forall L)_{x_i} \quad \dfrac{\dfrac{\dfrac{\dfrac{\forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y)))\vdash\forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y))) \quad A^{\mathbb{D}}(\underline{0})\vdash A^{\mathbb{D}}(\underline{0}) \quad A^{\mathbb{D}}(x)\vdash A^{\mathbb{D}}(x)}{\forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y))),A^{\mathbb{D}}(\underline{0}),\forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y)))\Rightarrow(A^{\mathbb{D}}(\underline{0})\Rightarrow A^{\mathbb{D}}(x))\vdash A^{\mathbb{D}}(x)}\,(\Rightarrow L)}{\forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y))),A^{\mathbb{D}}(\underline{0}),N(x)\vdash A^{\mathbb{D}}(x)}\,(\forall L)_X}{\forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y))),A^{\mathbb{D}}(\underline{0})\vdash N(x)\Rightarrow A^{\mathbb{D}}(x)}\,(\Rightarrow R)}{\forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y))),A^{\mathbb{D}}(\underline{0})\vdash\forall x(N(x)\Rightarrow A^{\mathbb{D}}(x))}\,(\forall R)_x}{\forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y))),A^{\mathbb{D}}(\underline{0}),N(x_1),\dots,N(x_m),\Gamma^{\mathbb{D}}\vdash B^{\mathbb{D}}}\,(cut) \tag{2.2.21}$$

which reduces to the derivation

$$\dfrac{\begin{matrix}\vdots\; d'^{\mathbb{D}}\\ \Gamma^{\mathbb{D}},A^{\mathbb{D}}(t^{\mathbb{D}})\vdash B^{\mathbb{D}}\end{matrix}\quad\begin{matrix}\vdots\; d_{t^{\mathbb{D}}}\{A^{\mathbb{D}}/X\}\\ \forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y))),A^{\mathbb{D}}(\underline{0}),N(x_1),\dots,N(x_m)\vdash A^{\mathbb{D}}(t^{\mathbb{D}})\end{matrix}}{\forall y(A^{\mathbb{D}}(y)\Rightarrow A^{\mathbb{D}}(\underline{s}(y))),A^{\mathbb{D}}(\underline{0}),N(x_1),\dots,N(x_m),\Gamma^{\mathbb{D}}\vdash B^{\mathbb{D}}}\,(cut) \tag{2.2.22}$$

Remark that in this case all the reductions are applied to the parts of the derivation introduced by the translation: the derivation $d_{t^{\mathbb{D}}}$ and the negative occurrence of $N(x)$ in the right-hand derivation (this is why the reduction is not "visible" in $\mathbf{HA}^2$).

2.2.3 Arithmetics and logic

Dedekind's translation provides a proof-theoretical bridge between arithmetics and logic. We recall here some applications, in particular the translation of some well-known theorems on arithmetics in the frame of second order (classical) logic.

The comparison of hierarchies The translation of arithmetics into second order logic provides an interesting proof-theoretical viewpoint over some results which are usually connected with arithmetics. Let us introduce two hierarchies for, respectively, arithmetical and second order logical closed formulae[9].

The *arithmetical hierarchy* is defined recursively as follows

Definition 2.2.1. *Let A be a closed arithmetical formula.*

- *A is Σ^0_0 or, equivalently Π^0_0, if it is classically equivalent to a formula without number quantifiers;*
- *A is Σ^0_{n+1} if it is classically equivalent to a formula of the form $\exists n_1 \dots \exists n_k\ B$, where B is Π^0_n;*
- *A is Π^0_{n+1} if it is classically equivalent to a formula of the form $\forall n_1 \dots \forall n_k\ B$, where B is Σ^0_n;*

Of particular interest for arithmetics are the two classes Σ^0_1 and Π^0_1. The first one is indeed connected with a completeness theorem:

Theorem 2.2.2 (Σ^0_1-completeness)**.** *Let A be a Σ^0_1 formula. If A is true in the standard model, then A is derivable in PA.*

Proof. see [BBJ07]. □

The second class is connected with Gödel's well-known incompleteness theorems, that can be reformulated as follows:

Theorem 2.2.3 (Π^0_1-incompleteness)**.** *There exists a Π^0_1 formula G which is true in the standard model but is not derivable in PA (if PA is coherent).*

Proof. This is just Gödel's first incompleteness theorem, along with the remark that the formula G is of the form $\forall n \neg\ prf_{PA}(n, \underline{k})$ (where $\underline{k}$ is the code of G) is Π^0_1. □

The logical hierarchy is defined recursively as follows

Definition 2.2.2. *Let A be a closed[10] formula of second order logic.*

- *A is Σ^0 or, equivalently Π^0, if it is classically equivalent to a formula without second order quantifiers;*
- *A is Σ^{n+1} if it is classically equivalent to a formula of the form $\exists X_1 \dots, \exists X_n\ B$, where B is Π^n;*
- *A is Π^{n+1} if it is classically equivalent to a formula of the form $\forall X_1 \dots \forall X_n\ B$, where B is Σ^n.*

With the aid of Dedekind's functor we can now restate theorems (2.2.2) and (2.2.3) as theorems concerning *classical* second order logic rather than arithmetics (in the following two paragraphs by second order logic we will implicitly mean *classical* second order logic $\mathbf{LK}^2$).

[9] Here by closed formula we mean a formula with no free first-order or number variable. Hence a closed formula can have free second order variables.

[10] Same remark that in the footnote above.

Π^1-completeness Let us first consider the class Π^1: it contains all formulas of the form $B = \forall X_1 \dots \forall X_n A$, where A is first-order. Typical examples of Π^1 formulae are those of the form $N(t)$.

Remark that a cut-free derivation of B still satisfies the subformula property: such a derivation must consist in a cut-free derivation of $\vdash A, \dots, A$, which satisfies subformula since A is first order, followed by instances of the $(\forall - R)$ rule and the contraction rule, which still satisfies subformula. A consequence of this remark is that we can extend the Schütte proof-search algorithm discussed above to Π^1 formula, obtaining the following result:

Theorem 2.2.4 (Π^1-completeness)**.** *Let A be a Π^1 logical formula. If A is not derivable (in* classical *second order logic), then it has a counter-model.*

Dedekind translation turns a Σ^0_1 formula A into a Π^1 one: if A is $\exists n_i A$, i.e. $\forall Y(\forall n_i(A(n_i) \Rightarrow Y) \Rightarrow Y)$, then, $A^{\mathbb{D}}$ is $\forall Y(\forall x_i(N(x_i) \Rightarrow (A^{\mathbb{D}}(x_i) \Rightarrow Y)) \Rightarrow Y)$ which is classically equivalent to $\exists x_i(N(x_i) \wedge A^{\mathbb{F}})$[11]. By applying theorem (2.2.1) we can thus derive the theorem (2.2.2) from the completeness theorem for Π^1 formulae.

Incompleteness and the comprehension schema Let us now consider the class Σ^1: it contains all formulae of the form $B = \exists X_1 \dots \exists X_n A$, where A is first-order. Remark that a cut-free derivations of a Σ^1 formulae might not satisfy the subformula property, since the premiss of the second order $(\exists R)$ rule may contain formulae of arbitrary logical complexity.

This fact has striking consequences, that we will explore in the next chapters: indeed the rule $(\exists R)$ can be equivalently reformulated by means of a *comprehension schema*:

$$\forall x_1 \dots x_n \exists X(A(x_1, \dots, x_n, y_1, \dots, y_m) \Leftrightarrow X(x_1, \dots, x_n)) \tag{2.2.23}$$

It is a well-known fact in the proof-theory of second order logic (see for instance [Poh89]) that the "strength" of second order systems depends on the complexity of their comprehension schemas.

As a consequence, when devising a proof-search for a Σ^1 formula, one can no more limit himself to a *finite* set of possible premisses for every rule: given a formula $\exists X A$, he must take into account all possible instances $A[P/X]$, for any predicate P: so to say, one is not only in search for the proof, but also in search for the predicates to use in the proof.

Dedekind translation turns a Π^0_1 formula into a Σ^1 formula: indeed, if A is $\forall n_i B$, then $A^{\mathbb{D}}$ is $\forall x_i(N(x_i) \Rightarrow B^{\mathbb{D}})$, which is classically equivalent to the Σ^1 formula $\exists X \forall x_i\big((\forall y(X(y) \Rightarrow X(\underline{s}(y))) \Rightarrow (X(\underline{0}) \Rightarrow X(x))\big) \Rightarrow B^{\mathbb{D}})$[12].

By applying theorem (2.2.1), theorem (2.2.3) can be reformulated as a theorem asserting that, as soon as subformula is lost, completeness is too:

Theorem 2.2.5 (Σ^1-incompleteness)**.** *There exists a valid Σ^1 formula which is not derivable in second order logic.*

Proof. One has to formulate a variant of Gödel's argument with a formula $G' := \forall n \neg prf_{\mathbf{LK}^2}(n, \underline{k})$, where $\underline{k} = \ulcorner G' \urcorner$ and the predicate $prf_{\mathbf{LK}^2}(n, m)$ codes derivability in second order logic. □

[11] Indeed, the converse also holds, that is, if $B := \forall X_1 \dots \forall X_n A$ is a Π^1 formula, by means of the Π^1-completeness theorem, it is equivalent to the validity of the first-order formula A, i.e. B is equivalent to the Σ^0_1 formula $\exists n(prf_{\mathbf{LK}}(n, \ulcorner A \urcorner))$, where $prf_{KL}(n, m)$ is the recursive predicate which codes derivability in first-order logic.

[12] Indeed, the converse also holds: it can be shown that a second order existentially closed Σ^1 formula $\exists X_1 \dots \exists X_n A$ is equivalent to the satisfiability of A which, by the completeness theorem for first order logic, is equivalent in turn to the Π^0_1 formula $\forall n(\neg prf_{\mathbf{LK}}(n, \ulcorner A \Rightarrow \bot \urcorner))$, where $prf_{\mathbf{LK}}(n, m)$.

Since the class Π^0_1 contains all those formulae that one can prove by means of an induction axiom, this means that such proofs contain a hidden comprehension: the Dedekind translation of a proof by induction corresponds exactly to a derivation in which the second order $(\forall L)$ rule occurs. So to say, Dedekind translation can be used to *extract* the comprehensions implicit in arithmetical proofs.

An interesting example can be found in Gentzen's 1943 paper [Gen69]: in order to show that transfinite induction up to ω_n ($TI(\omega_n)$[13]), for every integer n, can be derived in first-order Peano Arithmetics PA, he defines a series of predicates of growing complexity $TI_n(x)$ as

$$\begin{aligned}
&TI_1(x) := \forall y(TI(y) \Rightarrow TI(y+\omega^x))\\
&TI_2(x) := \forall z(\forall y(TI(y) \Rightarrow TI(y+\omega^z)) \Rightarrow \forall y(TI(y) \Rightarrow TI(y+\omega^{z+\omega^x})))\\
&TI_3(x) := \forall u(\forall z(\forall y(TI(y) \Rightarrow TI(y+\omega^z)) \Rightarrow \forall y(TI(y) \Rightarrow TI(y+\omega^{z+\omega^u}))) \Rightarrow\\
&\qquad\qquad \forall z(\forall y(TI(y) \Rightarrow TI(y+\omega^z)) \Rightarrow \forall y(TI(y) \Rightarrow TI(y+\omega^{z+\omega^{(u+\omega^x)}}))))\\
&\qquad\vdots
\end{aligned} \tag{2.2.24}$$

and constructs, by applying induction on the predicates $TI_n(x)$, cut-free derivations in PA of $TI(\omega_n)$, for all $n \in \mathbb{N}$. If we apply Dedekind translation to such derivations, we obtain derivations of formulae $TI(\omega_n)^{\mathbb{D}}$ of a *fixed* logical complexity containing comprehensions over predicates $TI_n(x)^{\mathbb{D}}$ whose logical complexity grows exponentially in n. In other words, we can use the translation to show the failure of the subformula property already in *first-order* Peano Arithmetics (see for instance [ST00]). This perspective is developed in detail in [Lei01], where the second order translation of arithmetics is applied to obtain a subsystem of $\mathbf{LK}^2$ which corresponds exactly to first order Peano Arithmetics.

2.3 The forgetful functor

In this section we recall some of the technical tools of the *Curry-Howard correspondence* between intuitionistic second order logic and polymorphic type theory.

First we associate with any formula A a type $A^{\mathbb{F}}$ and with any context of formulae Γ a context $\Gamma^{\mathbb{F}}$ of type declarations. Then, with any derivation d of a sequent $\Gamma \vdash A$ we associate a lambda term $\mathbb{F}(d)$ and a typing derivation $d^{\mathbb{F}}$ of the judgement $\Gamma^{\mathbb{F}} \vdash \mathbb{F}(d) : A^{\mathbb{F}}$.

This functorial translation has been called *forgetful* (as in [Gir11]) to stress the fact that it deletes all first order information: for instance, the translation of Dedekind's predicate is the type $\mathbf{N} = \forall\alpha((\alpha \to \alpha) \to (\alpha \to \alpha))$. In particular, the functoriality of the translation implies that the behavior of the rules for first order quantifiers under Gentzen's transformations has a void computational content: the reduction of a cut between first-order quantifiers implies no reduction of the corresponding programs (see [Lei90] for a discussion).

The payoff of this translation is at least threefold: firstly, since a normalizable term of the form $\mathbb{F}(d)$ must come from a derivation which reduces into a cut-free one, the *Hauptsatz* for intuitionistic second order logic can be directly inferred from the *weak normalization theorem* for System F (that will be shown and widely discussed in the next chapter), i.e. the theorem which asserts that every term typable in System F has a normal form.

[13] Corresponding to the formula $\forall x(\forall y(\forall z(z \prec y \Rightarrow A(z)) \Rightarrow A(y)) \Rightarrow (x \prec \omega_n \Rightarrow A(x)))$, where ω_n refers to a recursive coding of Cantor ordinal notation and $\prec$ is a recursive coding of the order relation on Cantor ordinals (see [ST00]).

A second consequence is at the level of derivability: since the existence of a proof of A corresponds to the existence of a λ-term of type $A^{\mathbb{F}}$, derivability in second order logic can be investigated from the viewpoint of typability in System F. This will be indeed the perspective developed in chapter (6) and discussed in chapter (7).

A third consequence is at the level of the structure of the derivations: one of the main fruitful directions within the *Curry-Howard* paradigm is to investigate the structure of proofs of certain classes of formulae through the behavior of their associated programs (see subsection (3.2.3) about Krivine's program). For instance, the derivations of the sequents $\vdash N(t)$ induce programs M which behave as *iterators*: given a base program N_0 and a functional program N_s, $(M)N_sN_0$ reduces to the k-th iteration of N_s over N_0, i.e. to the term $(N_s)^kN_0$: the computational content of Dedekind's predicate is thus expressed by iteration. A second important case is represented by derivations of the totality of recursive functions, whose associated λ-terms behave as programs computing those functions.

2.3.1 The functor $\mathbb{F}$

Formulas The translation of formulae and predicates into types is relatively straightforward: all we do is systematically erase first-order information from formulae.

$$\begin{array}{cc} (X_i(t_1,\dots,t_n))^{\mathbb{F}} := \alpha_i & (A \Rightarrow B)^{\mathbb{F}} := A^{\mathbb{F}} \to B^{\mathbb{F}} \\ (\forall x_i A)^{\mathbb{F}} := A^{\mathbb{F}} & (\forall X_i A)^{\mathbb{F}} := \forall \alpha_i A^{\mathbb{F}} \\ (\lambda x_1.\dots.\lambda x_n.A)^{\mathbb{F}} := A^{\mathbb{F}} & \end{array} \tag{2.3.1}$$

We translate contexts $\Gamma = \{A_1,\dots,A_n\}$ as follows: let $x_1,\dots,x_n$ be variables of the lambda calculus; then $\Gamma^{\mathbb{F}}$ is the *set* made of the type declarations $(x_1 : A_1^{\mathbb{F}}),\dots,(x_n : A_n^{\mathbb{F}})$.

Derivations We define now a map which associates with every derivation d of a sequent $\Gamma \vdash A$, a lambda term $\mathbb{F}(d)$ and a derivation $d^{\mathbb{F}}$ of the typing judgement $\Gamma^{\mathbb{F}} \vdash \mathbb{F}(d) : A^{\mathbb{F}}$.

We consider all cases:

(id) if $d = \dfrac{}{A \vdash A}\,(Ax)$, then $\mathbb{F}(d) := y$ and $d^{\mathbb{F}} := \dfrac{}{(y : A^{\mathbb{F}}) \vdash y : A^{\mathbb{F}}}$;

(cut) if $d = \dfrac{\begin{array}{c}\vdots\, d_1 \\ \Gamma, B \vdash A\end{array} \quad \begin{array}{c}\vdots\, d_2 \\ \Delta \vdash B\end{array}}{\Gamma, \Delta \vdash A}\,(cut)$, then $\mathbb{F}(d) := (\lambda x.\mathbb{F}(d_1))\mathbb{F}(d_2)$ and $d^{\mathbb{F}}$ is

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, d_1^{\mathbb{F}} \\ \Gamma^{\mathbb{F}}, (x : B^{\mathbb{F}}) \vdash \mathbb{F}(d_1) : A^{\mathbb{F}}\end{array}}{\Gamma^{\mathbb{F}} \vdash \lambda x.\mathbb{F}(d_1) : B^{\mathbb{F}} \to A^{\mathbb{F}}} \quad \begin{array}{c}\vdots\, d_2^{\mathbb{F}} \\ \Delta^{\mathbb{F}} \vdash \mathbb{F}(d_2) : B^{\mathbb{F}}\end{array}}{\Gamma^{\mathbb{F}}, \Delta^{\mathbb{F}} \vdash \mathbb{F}(d) : A^{\mathbb{F}}} \tag{2.3.2}$$

(W) if $d = \dfrac{\begin{array}{c}\vdots\, d' \\ \Gamma \vdash B\end{array}}{\Gamma, A \vdash B}\,(W)$ then $\mathbb{F}(d) := \mathbb{F}(d')$ and $d^{\mathbb{F}}$ is just $d'^{\mathbb{F}}$, where all contexts Δ have been replaced by $\Delta \cup (x : A^{\mathbb{F}})$ for a fresh variable x (use proposition (2.1.1) *i.*));

(C) if $d = \dfrac{\begin{array}{c}\vdots\, d' \\ \Gamma, A, A \vdash B\end{array}}{\Gamma, A \vdash B}\,(C)$, let x, y be respectively the variables associated to the two type declarations $(x : A^{\mathbb{F}})$ and $(y : A^{\mathbb{F}})$ occurring in $d'^{\mathbb{F}}$; then $\mathbb{F}(d) := \mathbb{F}(d')[x/y]$ and $d^{\mathbb{F}}$ is obtained from $d'^{\mathbb{F}}$ by replacing all occurrences of the declaration $(y : A^{\mathbb{F}})$ by the declaration $(x : A^{\mathbb{F}})$ (and remembering that contexts are *sets* of declarations).

$(\Rightarrow L)$ if $d = \dfrac{\begin{array}{cc} \vdots\, d_1 & \vdots\, d_2 \\ \Gamma, B \vdash C & \Delta \vdash A \end{array}}{\Gamma, \Delta, A \Rightarrow B \vdash C}\,(\Rightarrow L)$, then $\mathbb{F}(d) := \mathbb{F}(d_1)[y\mathbb{F}(d_2)/x]$, where x is the variable declared of type $B^{\mathbb{F}}$ in $d_1^{\mathbb{F}}$ and y is a fresh variable; one has the following two derivations

$$\begin{array}{c} \vdots\, d_1^{\mathbb{F}} \\ \Gamma^{\mathbb{F}}, (x : B^{\mathbb{F}}) \vdash \mathbb{F}(d_1) : C^{\mathbb{F}} \end{array} \qquad \dfrac{\Delta^{\mathbb{F}}, (y : A^{\mathbb{F}} \to B^{\mathbb{F}}) \vdash y : A^{\mathbb{F}} \to B^{\mathbb{F}} \quad \begin{array}{c} \vdots\, d_2^{\mathbb{F}} \\ \Delta^{\mathbb{F}} \vdash M_2 : A^{\mathbb{F}} \end{array}}{\Delta^{\mathbb{F}}, (y : A^{\mathbb{F}} \to B^{\mathbb{F}}) \vdash y\mathbb{F}(d_2) : B^{\mathbb{F}}} \tag{2.3.3}$$

and $d^{\mathbb{F}}$ is obtained by applying proposition (2.1.1) *i.* and the substitution lemma (2.1.1).

$(\Rightarrow R)$ if $d = \dfrac{\begin{array}{c} \vdots\, d' \\ \Gamma, A \vdash B \end{array}}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow R)$ then $\mathbb{F}(d) := \lambda x.\mathbb{F}(d')$, where x is the variable declared of type $A^{\mathbb{F}}$ in $d'^{\mathbb{F}}$, and $d^{\mathbb{F}}$ is

$$\dfrac{\begin{array}{c} \vdots\, d'^{\mathbb{F}} \\ \Gamma^{\mathbb{F}}, (x : A^{\mathbb{F}}) \vdash \mathbb{F}(d') : B^{\mathbb{F}} \end{array}}{\Gamma^{\mathbb{F}} \vdash \mathbb{F}(d) : A^{\mathbb{F}} \to B^{\mathbb{F}}} \tag{2.3.4}$$

$(\forall L)_x$ if $d = \dfrac{\begin{array}{c} \vdots\, d' \\ \Gamma, A(t) \vdash B \end{array}}{\Gamma, \forall x A \vdash B}\,(\forall L)_x$, then $\mathbb{F}(d) := \mathbb{F}(d')$ and $d^{\mathbb{F}} := d'^{\mathbb{F}}$;

$(\forall R)_x$ if $d = \dfrac{\begin{array}{c} \vdots\, d' \\ \Gamma \vdash A \end{array}}{\Gamma \vdash \forall x A}\,(\forall R)_x$, then $\mathbb{F}(d) := \mathbb{F}(d')$ and $d^{\mathbb{F}} := d'^{\mathbb{F}}$;

$(\forall L)_X$ if $d = \dfrac{\begin{array}{c} \vdots\, d' \\ \Gamma, A[P/X_i] \vdash B \end{array}}{\Gamma, \forall X_i A \vdash B}\,(\forall L)_x$, then $\mathbb{F}(d) = \mathbb{F}(d')$ we have the following two derivations:

$$\dfrac{\Gamma^{\mathbb{F}}, (x : \forall \alpha_i A^{\mathbb{F}}) \vdash x : \forall \alpha_i A^{\mathbb{F}}}{\Gamma^{\mathbb{F}}, (x : \forall \alpha_i A^{\mathbb{F}}) \vdash x : A^{\mathbb{F}}[P^{\mathbb{F}}/\alpha_i]} \qquad \begin{array}{c} \vdots\, d'^{\mathbb{F}} \\ \Gamma^{\mathbb{F}}, (x : A^{\mathbb{F}}[P^{\mathbb{F}}/\alpha_i]) \vdash \mathbb{F}(d') : B^{\mathbb{F}} \end{array} \tag{2.3.5}$$

one easily verifies by induction that $A^{\mathbb{F}}[P^{\mathbb{F}}/\alpha^i] = (A[P/X_i])^{\mathbb{F}}$ and $d^{\mathbb{F}}$ is obtained by means of the substitution lemma (2.1.1).

$(\forall R)_X$ if $d = \dfrac{\begin{array}{c} \vdots\, d' \\ \Gamma \vdash A \end{array}}{\Gamma \vdash \forall X_i A}\,(\forall R)_X$, then $\mathbb{F}(d) := \mathbb{F}(d')$ ad $d^{\mathbb{F}}$ is

$$\dfrac{\begin{array}{c} \vdots\, d'^{\mathbb{F}} \\ \Gamma^{\mathbb{F}} \vdash \mathbb{F}(d) : A^{\mathbb{F}} \end{array}}{\Gamma^{\mathbb{F}} \vdash \mathbb{F}(d) : \forall \alpha_i A^{\mathbb{F}}} \tag{2.3.6}$$

remark that the requirement $X \notin FV(\Gamma)$ implies that α is bindable in $\Gamma^{\mathbb{F}}$.

Remark 2.3.1. *Equalities $t = u$ are translated by $\mathbb{F}$ as the unity $(t = u)^{\mathbb{F}} = \mathbf{1}^{\mathbb{F}} = \forall\alpha(\alpha \to \alpha)$. This implies that no computational content is assigned to equalities: indeed the two rules of equality*

$$\frac{\Gamma \vdash A(t)}{\Gamma, t = u \vdash A(u)}\ (= L) \qquad\qquad \frac{}{\vdash t = t}\ (= R) \tag{2.3.7}$$

which are immediately derivable from the second order definition of equality $t = u := \forall X(X(t) \Rightarrow X(u))$, are translated into dummy terms by the forgetful functor:

$(= L)$ *if* $d = \dfrac{\begin{array}{c}\vdots\ d' \\ \Gamma \vdash A(t)\end{array}}{\Gamma, t = u \vdash A(u)}\ (= L)$, *then $\mathbb{F}(d) := \mathbb{F}(d')$ and $d^{\mathbb{F}} := d'^{\mathbb{F}}$;*

$(= R)$ *if* $d = \dfrac{}{\vdash t = t}\ (= R)$, *then $\mathbb{F}(d) := \lambda x.x$ and $d^{\mathbb{F}}$ is*

$$\dfrac{\dfrac{(x : \alpha) \vdash x : \alpha}{\vdash \mathbb{F}(d) : \alpha \to \alpha}}{\vdash \mathbb{F}(d) : \forall\alpha(\alpha \to \alpha)} \tag{2.3.8}$$

Reductions We pass now to show that if a derivation d reduces to d' by cut-elimination, then the lambda term $\mathbb{F}(d)$ and the lambda term $\mathbb{F}(d')$ are β-equivalent[14]. We limit ourselves to the cases of identity and implication:

(id) Let d be the derivation

$$\frac{A \vdash A \quad \begin{array}{c}\vdots\ d' \\ \Gamma \vdash A\end{array}}{\Gamma \vdash C}\ (cut) \tag{2.3.9}$$

which reduces in one step to d'. The derivation $d^{\mathbb{F}}$ is

$$\frac{\dfrac{(x : A^{\mathbb{F}}) \vdash x : A^{\mathbb{F}}}{\vdash \lambda x.x : A^{\mathbb{F}} \to A^{\mathbb{F}}} \quad \begin{array}{c}\vdots\ d'^{\mathbb{F}} \\ \Gamma^{\mathbb{F}} \vdash \mathbb{F}(d') : A^{\mathbb{F}}\end{array}}{\Gamma^{\mathbb{F}} \vdash (\lambda x.x)\mathbb{F}(d') : A^{\mathbb{F}}} \tag{2.3.10}$$

and clearly $\mathbb{F}(d)$ reduces in one step to $\mathbb{F}(d')$.

$(\Rightarrow L)/(\Rightarrow R)$ let d be the derivation

$$\frac{\dfrac{\begin{array}{c}\vdots\ d_1 \\ \Gamma_{11} \vdash A\end{array} \quad \begin{array}{c}\vdots\ d_2 \\ \Gamma_{12}, B \vdash C\end{array}}{\Gamma_1, A \Rightarrow B \vdash C}\ (\Rightarrow L) \quad \dfrac{\begin{array}{c}\vdots\ d_3 \\ \Gamma_2, A \vdash B\end{array}}{\Gamma_2 \vdash A \Rightarrow B}\ (\Rightarrow R)}{\Gamma \vdash A}\ (cut) \tag{2.3.11}$$

which reduces in one step to d' below

$$\frac{\dfrac{\begin{array}{c}\vdots\ d_1 \\ \Gamma_{11} \vdash A\end{array} \quad \begin{array}{c}\vdots\ d_3 \\ \Gamma_2, A \vdash B\end{array}}{\Gamma_{11}, \Gamma_2 \vdash B}\ (cut) \quad \begin{array}{c}\vdots\ d_2 \\ \Gamma_{12}, B \vdash C\end{array}}{\Gamma \vdash A}\ (cut) \tag{2.3.12}$$

[14]We recall that the relation $=_\beta$ of β-equivalence over pure λ-terms is the symmetric closure of the reduction relation $\to$.

The typing derivations $d^{\mathbb{F}}, d'^{\mathbb{F}}$ have respectively the shape below:

$$\frac{\dfrac{\vdots}{\Gamma^{\mathbb{F}} \vdash \lambda y.\mathbb{F}(d_2)[y\mathbb{F}(d_1)/x] : (A^{\mathbb{F}} \to B^{\mathbb{F}}) \to C^{\mathbb{F}}} \quad \dfrac{\vdots}{\Gamma^{\mathbb{F}} \vdash \lambda z.\mathbb{F}(d_3) : A^{\mathbb{F}} \to B^{\mathbb{F}}}}{\Gamma^{\mathbb{F}} \vdash (\lambda y.\mathbb{F}(d_2)[y\mathbb{F}(d_1)/x])\lambda z.\mathbb{F}(d_3) : A^{\mathbb{F}}} \tag{2.3.13}$$

$$\frac{\dfrac{\begin{array}{c}\vdots\ d_2^{\mathbb{F}} \\ \Gamma^{\mathbb{F}}, (x : B^{\mathbb{F}}) \vdash C^{\mathbb{F}}\end{array}}{\Gamma^{\mathbb{F}} \vdash \lambda x.\mathbb{F}(d_2) : B^{\mathbb{F}} \to C^{\mathbb{F}}} \quad \dfrac{\dfrac{\begin{array}{c}\vdots\ d_3^{\mathbb{F}} \\ \Gamma^{\mathbb{F}}, (z : A^{\mathbb{F}}) \vdash \mathbb{F}(d_3) : B^{\mathbb{F}}\end{array}}{\Gamma^{\mathbb{F}} \vdash \lambda z.\mathbb{F}(d_3)} \quad \begin{array}{c}\vdots\ d_1^{\mathbb{F}} \\ \Gamma^{\mathbb{F}} \vdash \mathbb{F}(d_1) : A^{\mathbb{F}}\end{array}}{\Gamma^{\mathbb{F}} \vdash (\lambda z.\mathbb{F}(d_3))\mathbb{F}(d_1) : B^{\mathbb{F}}}}{\Gamma^{\mathbb{F}} \vdash (\lambda x.\mathbb{F}(d_2))(\lambda z.\mathbb{F}(d_3))\mathbb{F}(d_1) : C^{\mathbb{F}}} \tag{2.3.14}$$

now $(\lambda y.\mathbb{F}(d_2)[y\mathbb{F}(d_1)/x])\lambda z.\mathbb{F}(d_3)$ and $(\lambda x.\mathbb{F}(d_2))(\lambda z.\mathbb{F}(d_3))\mathbb{F}(d_1)$ both reduce to the term $\mathbb{F}(d_2)[\mathbb{F}(d_3)[\mathbb{F}(d_1)/z]/x]$.

Remark 2.3.2. *We can also consider the derived case of equality, as it will be explicitly used in the next section:*

Let d be the derivation

$$\frac{\dfrac{\begin{array}{c}\vdots\ d' \\ \Gamma \vdash A\end{array}}{\Gamma, t = t \vdash A}\,(= L) \qquad \dfrac{}{\vdash t = t}\,(= R)}{\Gamma \vdash A}\,(cut) \tag{2.3.15}$$

which reduces in one step to d'; the derivation $d^{\mathbb{F}}$ is

$$\frac{\begin{array}{c}\vdots\ d'^{\mathbb{F}} \cup (z : \forall\alpha(\alpha \to \alpha)) \\ \Gamma^{\mathbb{F}}, (z : \forall\alpha(\alpha \to \alpha)) \vdash \mathbb{F}(d') : A^{\mathbb{F}}\end{array} \qquad \dfrac{\dfrac{(x : \alpha) \vdash x : \alpha}{\vdash \lambda x.x : \alpha \to \alpha}}{\vdash \lambda x.x : \forall\alpha(\alpha \to \alpha)}}{\Gamma^{\mathbb{F}} \vdash (\lambda z.\mathbb{F}(d'))\lambda x.x : A^{\mathbb{F}}} \tag{2.3.16}$$

and clearly $\mathbb{F}(d)$ reduces in one step to $\mathbb{F}(d')$, since z is fresh.

We end this subsection by recalling a result (called *faithfulness* in [Kre70]) which shows that the forgetful functor can be inverted: typed programs are exactly those that are the image, under the forgetful translation, of actual derivations in sequent calculus

Theorem 2.3.1 (faithfulness)**.** *If $(x_1 : A_1^{\mathbb{F}}), \ldots, (x_k : A_n^{\mathbb{F}}) \vdash M : A^{\mathbb{F}}$ is derivable in simple type theory, then there exists a sequent calculus derivation d of conclusion $A_1, \ldots, A_n \vdash A$ such that $\mathbb{F}(d) = M$.*

Proof. We argue by induction on construction of M:

($M = x_i$) The typing derivation of M is just the axiom $(x_1 : A_1^{\mathbb{F}}), \ldots, (x_k : A_k^{\mathbb{F}}) \vdash x_i : A_i^{\mathbb{F}}$, and d is obtained by an axiom followed by several weakenings:

$$\frac{A_i \vdash A_i}{\overline{A_1, \ldots, A_k \vdash A_i}} \tag{2.3.17}$$

($M = \lambda x.M'$) Then $A^{\mathbb{F}} = B^{\mathbb{F}} \to C^{\mathbb{F}}$ and the typing derivation of M has the form

$$\frac{(x_1 : A_1^{\mathbb{F}}), \ldots, (x_k : A_k^{\mathbb{F}}), (x : B^{\mathbb{F}}) \vdash M' : C^{\mathbb{F}}}{(x_1 : A_1^{\mathbb{F}}), \ldots, (x_k : A_k^{\mathbb{F}}) \vdash \lambda x.M' : A^{\mathbb{F}}} \tag{2.3.18}$$

then, by induction hypothesis there exists a derivation d' of $A_1, \ldots, A_k, B \vdash C$ such that $\mathbb{F}(d') = M'$ and we can obtain d with a $(\Rightarrow R)$ rule:

$$\dfrac{\begin{array}{c} \vdots\ d' \\ A_1, \ldots, A_k, B \vdash C \end{array}}{A_1, \ldots, A_k \vdash B \Rightarrow C} \tag{2.3.19}$$

($M = (x)M_1 \ldots M_h$) Then the typing derivation of M has the form:

$$\dfrac{\begin{array}{c} \dfrac{\Gamma \vdash x_i : B_1^{\mathbb{F}} \to \cdots \to B_h^{\mathbb{F}} \to A^{\mathbb{F}} \quad \begin{array}{c} \vdots\ d_1 \\ \Gamma \vdash M_1 : B_1^{\mathbb{F}} \end{array}}{\Gamma \vdash (x_i)M_1 : B_2^{\mathbb{F}} \to \cdots \to B_h^{\mathbb{F}} \to A^{\mathbb{F}}} \\ \vdots \\ \Gamma \vdash (x_i)M_1 \ldots M_{h-1} : B_h^{\mathbb{F}} \to A^{\mathbb{F}} \end{array} \qquad \begin{array}{c} \vdots\ d_h \\ \Gamma \vdash M_h : B_h^{\mathbb{F}} \end{array}}{\Gamma \vdash (x_i)M_1 \ldots M_h : A^{\mathbb{F}}} \tag{2.3.20}$$

where $A_i^{\mathbb{F}} \equiv B_1^{\mathbb{F}} \to \cdots \to B_h^{\mathbb{F}} \to A^{\mathbb{F}}$ and Γ is the context $(x_1 : A_1), \ldots, (x_k : A_k)$. Then the derivation d is the following:

$$\dfrac{\dfrac{\begin{array}{c} \dfrac{A_1, \ldots, A, \ldots, A_k \vdash A \quad \begin{array}{c} \vdots\ d_h^* \\ \Delta \vdash B_h \end{array}}{\Delta, A_1, \ldots, B_h \Rightarrow A, \ldots, A_k \Rightarrow A}\ (\Rightarrow L) \\ \vdots \\ \Delta, \ldots, \Delta, A_1, \ldots, B_2 \Rightarrow \ldots \Rightarrow B_h \Rightarrow A, \ldots, A_k \vdash A \end{array} \quad \begin{array}{c} \vdots\ d_1^* \\ \Delta \vdash B_1 \end{array}}{\Delta, \ldots, \Delta, A_1, \ldots, B_1 \Rightarrow \ldots \Rightarrow B_h \Rightarrow A, \ldots, A_k \vdash A}\ (\Rightarrow L)}{\Delta \vdash A}\ (C) \tag{2.3.21}$$

where Δ is the context $A_1, \ldots, B_1 \Rightarrow \ldots \Rightarrow B_h \to A, \ldots, A_k$ and d_j^*, for $1 \leq j \leq h$, exists by induction hypothesis and is such that $\mathbb{F}(d_j^*) = d_j$. Remark that the order of appearance of the d_j^* is inverted with respect to the order of appearance of the d_j.

($M = (\lambda x.M_1)M_2$) Then the typing derivation of M has the form:

$$\dfrac{\dfrac{\begin{array}{c} \vdots\ d_1 \\ \Gamma, (x : B^{\mathbb{F}}) \vdash M_1 : A^{\mathbb{F}} \end{array}}{\Gamma \vdash \lambda x.M_1 : B^{\mathbb{F}} \to A^{\mathbb{F}}} \quad \begin{array}{c} \vdots\ d_2 \\ \Gamma \vdash M_2 : B^{\mathbb{F}} \end{array}}{\Gamma \vdash M : A^{\mathbb{F}}} \tag{2.3.22}$$

where Γ is as above. Then the derivation d is the following:

$$\dfrac{\dfrac{\begin{array}{c} \vdots\ d_1^* \\ \Delta, B \vdash A \end{array} \quad \begin{array}{c} \vdots\ d_2^* \\ \Delta \vdash B \end{array}}{\Delta, \Delta \vdash A}\ (cut)}{\Delta \vdash A}\ (C) \tag{2.3.23}$$

where Δ is the context $A_1, \ldots, A_k$ and d_1^*, d_2^* exist by induction hypothesis and are such that $\mathbb{F}(d_1^*) = d_1$ and $\mathbb{F}(d_2^*) = d_2$.

□

2.3.2 Arithmetics in type theory

The composition of the two functors yields a type-theoretic interpretation of arithmetics, that we briefly recall.

Composing $\mathbb{D}$ and $\mathbb{F}$ We present here some well-known results on the interpretation of arithmetics within System F (see [GLT89]). This translation can now be presented as the composition of the Dedekind and the forgetful translation.

Let us introduce the type $\mathbf{N} := N(x)^{\mathbb{F}}$, which is the standard type for the iterators:

$$\mathbf{N} := \forall\alpha((\alpha \to \alpha) \to (\alpha \to \alpha)) \tag{2.3.24}$$

Let $t \in \mathcal{T}$ and d_t be the *number derivation* of $\Gamma, N(x_1), \dots, N(x_k) \vdash N(t)$. By applying $\mathbb{F}$ we obtain a (normal) program $\mathbb{F}(d_t)$ and a derivation $d_t^{\mathbb{F}}$ of the judgement $\Gamma^{\mathbb{F}}, (x_1 : \mathbf{N}), \dots, (x_k : \mathbf{N}) \vdash \mathbb{F}(d_t) : \mathbf{N}$ (remark the abuse of notation). In particular, if $t = \underline{n}$, then the context of $d_{\underline{n}}$ is empty and thus $\mathbb{F}(d_{\underline{n}})$ is a normal term of type $\mathbf{N}$. One easily shows then by induction that $\mathbb{F}(d_{\underline{n}})$ corresponds to the n-th *Church numeral* $\mathbf{n} := \lambda f.\lambda x.(f)^n x$. In other words, a derivation of $N(\underline{n})$ corresponds to a program which behaves like a n-times iterator.

One of the most significative examples of composition of $\mathbb{D}$ and $\mathbb{F}$, that we use through-out this text, concerns the *provably recursive functions*: a k-ary recursive function f is said *provably recursive* (or *provably total*) if it is derivable in $\mathbf{PA}^2$ that

$$\forall n_1 \dots \forall n_k \exists m(\mathbf{f}(n_1, \dots, n_k) = m) \tag{2.3.25}$$

where $\mathbf{f}$ is a function symbol introduced along with a set of equational axioms.

Now, if a function is provably recursive then its totality can be derived already in $\mathbf{HA}^2$: this follows from a well-known theorem by [Fri78] which says that $\mathbf{HA}^2$ and $\mathbf{PA}^2$ prove exactly the same Π^0_2 statements[15].

Let us say that a k-ary recursive function f is *representable* in System F if there exists a λ-term M such that, for all $n_1, \dots, n_k$, $(M)\mathbf{n}_1, \dots, \mathbf{n}_k$ reduces to $\mathbf{m}$ if and only if $f(n_1, \dots, n_k) = m$ and moreover the judgement $\vdash M : \mathbf{N} \to \mathbf{N}$ is derivable in System F. A classic result is the following:

Theorem 2.3.2 ([Gir72, GLT89])**.** *The provably recursive functions of second order Peano arithmetics are exactly those which are representable in System F.*

Proof. We limit ourselves to sketch the first part of the proof, in order to highight the role of the two functorial translations. The second part, which involves the notion of reducibility which we introduce in chapter (4), will be sketched in section (4.3.1) and can be found in [Gir72, GLT89].

Let f be provably recursive and let d be a derivation of $\Gamma \vdash \forall n_1 \dots \forall n_k \exists m(\mathbf{f}(n) = m)$ (where Γ contains equations expressing the "meaning" of the function symbols defining $\mathbf{f}$). By applying the Dedekind functor to d we obtain a derivation $d^{\mathbb{D}}$ of the sequent $\Gamma' \vdash B$, where B is the formula below:

$$\forall x_1 \dots \forall x_k \exists y(N(x_1) \Rightarrow \dots \Rightarrow N(x_k) \Rightarrow N(y) \wedge \mathbf{f}(x_1, \dots, x_k) = y) \tag{2.3.26}$$

and Γ' contains the equational axioms of the function f plus the axioms $PA1$ and $PA2$.

[15] The idea of this theorem is that of using the $\neg\neg$-translation from classical to intuitionistic logic: in particular the translation of a Π^0_2 formula $\forall n \exists m A$ is $\forall n \neg\neg\exists m A$. Now it can be shown by standard proof-theoretic techniques that the latter formula is derivable in $\mathbf{HA}^2$ if and only if the former is derivable in $\mathbf{PA}^2$.

A simple manipulation turns the derivation $d^{\mathbb{D}}$ into a derivation d' of the sequent $\Gamma' \vdash \mathbf{Tot}(f)$, where $\mathbf{Tot}(f)$ is the formula below (which is intuitionistically equivalent to B)

$$\forall x_1 \dots \forall x_k (N(x_1) \Rightarrow \dots \Rightarrow N(x_k) \Rightarrow N(\mathbf{f}(x_1, \dots, x_k))) \tag{2.3.27}$$

Now we can apply the forgetful functor to the derivation d': this produces a program M_f and a derivation of the judgement $(z_0 : \forall\alpha(\alpha \to \alpha) \to \forall\alpha\alpha), (z_1 : \forall\alpha(\alpha \to \alpha)), \dots, (z_h : \forall\alpha(\alpha \to \alpha)) \vdash M_f : \mathbf{N} \to \mathbf{N}$. Indeed all axioms are interpreted by $\mathbb{F}$ as unities except $PA1$, which is interpreted as the negation of the unity $\forall\alpha(\alpha \to \alpha) \to \forall\alpha\alpha \equiv \forall\alpha\alpha$. Let then $M'_f := M_f[\lambda z.z/z_1, \dots, \lambda z.z/z_h]$; since $\lambda z.z$ has type $\forall\alpha(\alpha \to \alpha)$, it follows by the substitution lemma (2.1.1) that $(z_0 : \forall\alpha(\alpha \to \alpha) \to \forall\alpha\alpha) \vdash M'_f : \mathbf{N} \to \mathbf{N}$ is derivable in F.

It remains then to get rid of the free variable $z_0 : \forall\alpha(\alpha \to \alpha) \to \forall\alpha\alpha$: a first solution (discussed in [GLT89]) would be to add a junk term Ω of type $\forall\alpha\alpha$ to System F, so that $\lambda z.\Omega$ can be given type $\forall\alpha(\alpha \to \alpha) \to \forall\alpha\alpha$. The argument we develop below would suffice indeed to show that the term Ω disappears during the normalization process. A more elegant solution requires a slight modification of the forgetful interpretation, but for all details we address the reader to [GLT89].

By applying one of the two mentioned strategies, we get, in definitive, a *closed* term M^*_f and a derivation of $\vdash M^*_f : \mathbf{N} \to \mathbf{N}$.

We want now to show that the program M^*_f effectively computes the function f; to do this, we will have to rely on the *Hauptsatz* for second order logic (that will be proved in the next chapter). Indeed, by applying the *Hauptsatz* we get that, for all $k_1, \dots, k_h \in \mathbb{N}$, the derivation below

$$\dfrac{\dfrac{\vdots\, d'}{\Gamma \vdash \forall x_1 \dots \forall x_h (N(x_1) \Rightarrow \dots \Rightarrow N(x_h) \Rightarrow N(\mathbf{f}(x_1, \dots, x_h)))} \qquad \dfrac{\dfrac{\begin{array}{c} \dfrac{\dfrac{\vdots}{\forall x_1 \dots \forall x_h (N(x_1) \Rightarrow \dots \Rightarrow N(x_h) \Rightarrow N(\mathbf{f}(x_1, \dots, x_h))), N(\underline{k}_1), \dots, N(\underline{k}_h) \vdash N(\mathbf{f}()\underline{k}_1, \dots, \underline{k}_h)} \quad \dfrac{\vdots\, d_{k_1}}{\vdash N(\underline{k}_1)}}{\forall x_1 \dots \forall x_h (N(x_1) \Rightarrow \dots \Rightarrow N(x_h) \Rightarrow N(\mathbf{f}(x_1, \dots, x_h))), N(\underline{k}_2), \dots, N(\underline{k}_h) \vdash N(\mathbf{f}(\underline{k}_1, \dots, \underline{k}_h))}\,(cut) \\ \vdots \end{array}}{\forall x_1 \dots \forall x_h (N(x_1) \Rightarrow \dots \Rightarrow N(x_h) \Rightarrow N(\mathbf{f}(x_1, \dots, x_h))), N(k_h) \vdash N(\mathbf{f}(\underline{k}_1, \dots, \underline{k}_h))} \quad \dfrac{\vdots\, d_{k_h}}{\vdash N(\underline{k}_h)}}{\forall x_1 \dots \forall x_h (N(x_1) \Rightarrow \dots \Rightarrow N(x_h) \Rightarrow N(\mathbf{f}(x_1, \dots, x_h))) \vdash N(\mathbf{f}(\underline{k}_1, \dots, \underline{k}_h))}\,(cut)}{\Gamma \vdash N(\mathbf{f}(\underline{k}_1, \dots, \underline{k}_h))}\,(cut) \tag{2.3.28}$$

reduces into a cut-free derivation $e_{k_1,\dots,k_h}$ of $\Gamma \vdash N(\mathbf{f}(\underline{k}_1, \dots, \underline{k}_h))$; now, since no parameters occur in the formulae in the sequent, it follows that $e_{k_1,\dots,k_h}$ contains a derivation of $\vdash N(\underline{p})$, for a certain $p \in \mathbb{N}$ followed by several application of the $(= L)$ rule. From the soundness of $\mathbf{HA}^2$ we get indeed $f(k_1, \dots, k_h) = p$.

We can now rely on the functorial nature of both $\mathbb{D}$ and $\mathbb{F}$ and verify that $(M_f)\mathbf{k}_1 \dots \mathbf{k}_h$ reduces indeed to $(e^{\mathbb{D}})^{\mathbb{F}}$, which must be of the form $(z_{i_1})(z_{i_2}) \dots (z_{i_q})\mathbf{p}$, for a certain $q \in \mathbb{N}$, where the z_{i_j} are the variables corresponding to the equality axioms of the function f. As an immediate consequence we get that $(M^*_f)\mathbf{k}_1 \dots \mathbf{k}_h$ must reduce to $\mathbf{p}$. In other words, we have shown that for all $k_1, \dots, k_h$, the application of M_f to the *Church numerals* $\mathbf{k}_1, \dots, \mathbf{k}_h$ reduces to the *Church numeral* corresponding to $f(k_1, \dots, k_h)$.

$\square$

A simple application of the theorem above is provided by the standard exercise of constructing derivations of totality d_+ and $d_\times$ in $\mathbf{HA}^2$, respectively for the sum and the product of natural numbers in such a way that the application of the Dedekind and the forgetful translation to such produces the two terms Add and $Mult$ below

$$Add := \lambda x.\lambda y.\lambda f.\lambda z.(x)f((y)fz) \tag{2.3.29}$$

$$Mult := \lambda x.\lambda y.\lambda f.\lambda z.x(yf)z \tag{2.3.30}$$

which correspond to the usual programs to code sum and product of *Church numerals* in λ-calculus.

Type inference and the type hierarchy We introduce a hierarchy of types which allows to extend the comparison of hierarchies between logic and arithmetics to type theory.

Definition 2.3.1. *Let σ be a type of System F.*

- *σ is $\mathbf{\Sigma}^0$ or, equivalently $\mathbf{\Pi}^0$, if it is quantifier-free;*
- *σ is $\mathbf{\Sigma}^{n+1}$ if it is of the form $\tau \to \rho$, where τ and ρ are $\mathbf{\Pi}^n$;*
- *σ is $\mathbf{\Pi}^{n+1}$ if it is of the form $\forall\alpha_1 \dots \forall\alpha_n\tau$, where τ is $\mathbf{\Sigma}^n$.*

In the next chapter we will derive theorems which correspond, in type theory, to the Π^1-completeness and the Σ^1-incompleteness theorems (2.2.3) and (2.2.2) of second order logic: we introduce a predicate of *reducibility* Red_σ (or *validity* or *realizability*) for programs with respect to a type σ and we will prove the following:

- if σ is $\mathbf{\Pi}^1$ and M is a normal λ-term such that $Red_\sigma(M)$, then $\vdash M : \sigma$ is derivable in System F;
- there exists a normal term M and such that $Red_{\mathbf{N}\to\mathbf{N}}(M)$ but $\vdash M : \mathbf{N} \to \mathbf{N}$ is not derivable in System F (remark that $\mathbf{N} \to \mathbf{N}$ is a $\mathbf{\Sigma}^1$ type).

In chapter (6) we will discuss the *type inference* problem for $\mathbf{\Pi}^1$ and $\mathbf{\Sigma}^1$ types: when is $\vdash M : \sigma$ derivable in System F?

Indeed, in the $\mathbf{\Pi}^1$ case, this can formulated as a problem of *first-order unification* and shown to be decidable (see [MD82]); in the $\mathbf{\Sigma}^1$ this can be formulated as a problem of *second-order unification* and is known to be indecidable (see [Wel98]).

2.4 Beyond System F

The following pages contain a brief presentation of the systems F^ω, U^-, U, N, which are higher order extensions of System F and which can be seen as more and more powerful Curry-Howard formalisms for higher-order logic.

The generalization of the polymorphic type discipline of System F poses some delicate theoretical challenges. In particular, the identification of propositions (or formulae, see footnotes 16 and 17) and types, which is apparently at work in the Curry-Howard correspondence, seems incompatible with a completely uniform treatment of quantification over types.

As these theoretical questions involve many technical notions and ideas that will be presented later in this text, this section can be read as a sketch of some issues that will be developed in more detail in the next chapters, or simply skipped and postponed to a later reading.

2.4.1 From Curry's type theory to System F^ω

The type *prop* Historically, the task of generalizing the polymorphic type discipline of System F led to some difficulties which are very similar to the ones faced at the very beginning of the history of type theory.

FIrst observe that quantification over arbitrary propositions[16], along with Russell's principle (**RUS**) (discussed in subsection (3.2.3))

[16] In the literature on type theory and typed λ-calculi it is standard to talk of *propositions* rather than *formulae*; since the literature we are confronted with in this chapter is essentially type-theoretic we follow this terminology in the following pages, in order to avoid confusion in the description of type systems.

(**RUS**) *The range of significance of a propositional function forms a type*

implies that there must be a type *prop* of all propositions.

Church's original version of the simply typed λ-calculus in [Chu40] (that we will call CTT for *Church's type theory*) was indeed thought as a representation of Russell's doctrine of types. CTT contains a type *prop* of all propositions (intended *á la Frege* as truth-values), a type ι for individuals and several constants, among which the constants $\rightarrow$ of type $prop \rightarrow prop \rightarrow prop$ and Π^{σ} of type $(\sigma \rightarrow prop) \rightarrow prop$ (for every type σ).

A n-ary *predicate* $P(x_1^{\sigma_1}, \dots, x_n^{\sigma_n})$ in CTT is represented by a λ-term of the form $\lambda x_1. \dots .\lambda x_n.M$ of type $\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow prop$. An atomic proposition $P(t_1, \dots, t_k)$ is obtained then by the application of the term M representing the predicate $P(x_1^{\sigma_1}, \dots, x_n^{\sigma_n})$ to the terms $N_1, \dots, N_k$ representing the individuals $t_1, \dots, t_k$. Complex propositions are constructed by means of the constants $\rightarrow$ and Π^{σ}:

- given two propositions A, B, represented by terms M, N, the proposition $A \Rightarrow B$ is represented by the term $(\rightarrow)MN$;
- given a proposition A depending a free variable x^{σ}, represented by the term M, the proposition $\forall x^{\sigma} A$ is represented by the term $\Pi^{\sigma}(\lambda x^{\sigma}.M)$.

Thus, in Church's type theory, a proposition is represented by a typed λ-term (with constants). The reader should not confuse between Church's identification of propositions with typed λ-terms and the Curry-Howard correspondence between proofs and typed λ-terms. The latter is indeed based on the principle **PasT** (discussed in subsection (3.2.3))

(**PasT**) *Propositions should be identified with types*

which asserts the identification of propositions[17] and types.

As it is observed in [Coq90], the conjunction of the principle **RUS** and the principle **PasT** is incompatible with quantification over all propositions: since, as we already remarked, quantification over all propositions and **RUS** imply the existence of a type of all propositions, the identification of types and proposition implies that this type must be a type of all types, an hypothesis which is inconsistent, as it will be shown in subsection (4.3.2). This idea was indeed one of the main motivations for Martin-Löf's introduction of the type ν of all types in his original type theory [ML70b], shown to be inconsistent in [Gir72] (see subsection (4.3.2) and appendix (B) for more details).

Indeed, one of the main features of Martin-Löf's type theory is the identification of two *prima facie* distinct forms of typing: the typing of terms, where the latter are seen as (the interpretation of) proofs, and the typing of propositions, where the latter are seen as (the interpretation of) formulae and predicates.

Hence, if one wishes to extend polymorphic type theory in the style of Church's type theory the identification of propositions and types must be rejected: a distinction must be made between the types for the terms and the types for the propositions; this solution is at the basis of systems like F^{ω} (see [Gir72, Urz97]), the *calculus of constructions* [Coq90] and the *pure type systems* (see [Ber88]). Though these systems do not follow the identification of propositions and types, they can still be considered "Curry-Howard" as they can be related to higher order intuitionistic sequent calculi by means of rather straightforward extensions of the forgetful translation described above (see for instance [Lei94]).

[17] A terminological ambiguity, which seems to persist in the literature, must be here stressed: Curry [CF58] originally noticed a correspondence between logical *propositions* and *types*; Howard's [How80] presents a correspondence between *formulae* and *types*; still, one reads about *propositions-as-types* in [ML84, Coq90], and about *formula-as-types* in the classical notes [SU06] and in [GLT89].

In order to avoid confusions around the word "type", we will talk of *propositions* when referring to the expressions used to type proof-like terms, and of *kinds* or *universes* when referring to the expressions used to type constructors, i.e. terms used to build propositions (hence, *prop* will be considered as a universe).

We can define the grammar of *pure* (i.e. untyped) *constructors* as follows (we use $X, Y, \dots$ to indicate constructor variables):

$$C, D := X | C \to D | \forall X C | \lambda X.C \tag{2.4.1}$$

Hence, by a proposition we will mean a pure constructor C such that $\Gamma \vdash C : \mathit{prop}$ is derivable in the type system.

System F^ω The Curry-Howard version of Church's type theory is an extension of System F called System F^ω. In System F^ω one has three levels of objects: "proof-like" terms, notation $M, N, \dots$, type constructors, notation $C, D, \dots$, and universes, notation $\kappa, \kappa', \dots$.

Universes are defined similarly to simple types: one has a constant *prop*, and a constructor $\to$ of type $\mathit{prop} \to \mathit{prop} \to \mathit{prop}$, with the following rules

$$\boxed{\begin{array}{cc} \dfrac{}{\Gamma, (\gamma : \kappa) \vdash \gamma : \kappa}\ (id) & \\ \dfrac{\Gamma \vdash C : \mathit{prop} \quad \Gamma \vdash D : \mathit{prop}}{\Gamma \vdash C \to D : \mathit{prop}}\ (\to) & \dfrac{\Gamma, (\alpha : \kappa) \vdash C : \mathit{prop}}{\Gamma \vdash \forall^\kappa \alpha C : \mathit{prop}}\ (\forall^\kappa) \\ \dfrac{\Gamma \vdash C : \kappa \to \kappa' \quad \Gamma \vdash D : \kappa}{\Gamma \vdash (C)D : \kappa'}\ (@) & \dfrac{\Gamma, (\gamma : \kappa) \vdash C : \kappa'}{\Gamma \vdash \lambda\gamma.C : \kappa \to \kappa'}\ (\lambda) \end{array}} \tag{2.4.2}$$

Once defined universes, one can call *types* those constructors C such that $\Gamma \vdash C : \mathit{prop}$ is derivable in the system above. The rules for typing "proof-like" terms are then the following:

$$\boxed{\begin{array}{cc} \dfrac{}{\Gamma, (x : \sigma) \vdash x : \sigma}\ (id) & \dfrac{\Gamma \vdash M : \sigma \quad \sigma =_\beta \tau}{\Gamma \vdash M : \tau}\ (\beta) \\ \dfrac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash M : \sigma}{\Gamma \vdash MN : \tau}\ (@) & \dfrac{\Gamma, (x : \sigma) \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\lambda) \\ \dfrac{\Gamma \vdash N : \forall^\kappa X \sigma \quad \Gamma \vdash C : \kappa}{\Gamma \vdash M : \sigma[C/X]}\ (\forall^\kappa E) & \dfrac{\Gamma, (X : \kappa) \vdash M : \sigma \quad X \text{ bindable in } \Gamma}{\Gamma \vdash M : \forall^\kappa X \sigma}\ (\forall^\kappa I) \end{array}} \tag{2.4.3}$$

Where X is bindable in Γ if it does not occur free in any of the constructors occurring in Γ.

Remark the rule (β), which accounts for the possibility that a type containing a redex be reduced. On the other hand, since all types in F^ω are strongly normalizing (as a consequence of the reducibility theorem for simple type theory (3.2.1)), one can eliminate rule (β) and replace the rule $(\forall^\kappa E)$ by the rule $(\forall^\kappa E)'$ below

$$\dfrac{\Gamma \vdash M : \forall^\kappa X \sigma \quad \Gamma \vdash C : \kappa}{\Gamma \vdash M : nf(\sigma[C/X])}\ (\forall^\kappa E)' \tag{2.4.4}$$

where $nf(\sigma)$, for a type σ, denotes its normal form.

A constructor of universe *prop* will be called a *proposition* and noted, as usual, by small greek letters $\sigma, \tau, \dots$. A constructor $\lambda\gamma.C$ of universe $\kappa \to \mathit{prop}$ will be called a *set* over κ and noted in set notation as $\{\gamma : \kappa | C\}$. Moreover, if C is a set over κ and D is in κ, then we will note the application CD in set notation as $D \in C$. Thus, we can see the type theory F^ω as a set theory.

The System F^ω is quite well-studied in the literature (see [Urz97, Mal97]); here we recall some well-known facts about the reducibility of System F^ω. The remarks that follow make reference to reducibility and its connected technical aspects that will be introduced in the next chapters (chapter (3), (4) and (5)), so the reader not familiar with these topics may want to postpone the reading of the following lines after the reading of those chapters.

The reducibility technique for System F (presented in chapter (4)) can be straightforwardly extended to prove strong normalization for system F^ω. The idea of the extension is indeed contained in the proof sketched in section (4.3.2) of normalization for Martin-Löf's type theory; in particular, one interprets universes as sets as follows: the interpretation of the universe *prop* is set CR of all reducibility candidates (remind that $CR \subseteq \wp(\Lambda)$); the interpretation of the universe $\kappa \to \kappa'$ is then the set of all functions from the interpretation of κ to the interpretation of κ'. As a consequence, propositions are interpreted by means of reducibility candidates (as types in System F), and general constructors are interpreted by functions in the appropriate function space.

The reducibility interpretation of F^ω has many similarities with Reynolds' set-theoretic interpretation of type theory (sketched in subsection (5.1.1)): one interprets implication universes by means of function spaces. That is, the reducibility interpretation of higher-order type theory mimics the set-theoretic interpretation of simple type theory.

2.4.2 The systems U and U^-

It seems then quite natural to expect the worse to happen if one tries to extend the hierarchy of universes by means of impredicative quantifiers in the style of System F. If we denote universe variables as $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \dots$, we can introduce a quantifier over universes: if κ is a universe and $\mathcal{X}$ a variable, then $\forall \mathcal{X}\kappa$ is a universe, intuitively the "intersection" of all universes $\kappa[\kappa'/\mathcal{X}]$.

The system U^- is obtained by extending System F^ω by means of polymorphic universes, i.e. by adding to F^ω the following rules:

$$\frac{\Gamma \vdash C : \forall \mathcal{X}\kappa}{\Gamma \vdash C : \kappa[\kappa'/\mathcal{X}]}\ (\forall E) \qquad \frac{\Gamma \vdash C : \kappa \quad \mathcal{X} \text{ bindable in } \Gamma}{\Gamma \vdash C : \forall \mathcal{X}\kappa}\ (\forall I) \tag{2.4.5}$$

Clearly System U^- contains much more sets than System F^ω: in particular, one can construct in System U^- "paradoxical universes" ([Hur95]) of the form

$$\mathcal{U} := \forall \mathcal{X}((\wp\wp\mathcal{X} \to \mathcal{X}) \to \mathcal{X}) \tag{2.4.6}$$

where $\wp\kappa := \kappa \to \textit{prop}$ is the universe of sets over κ. One can in particular reproduce Reynolds' argument (section (5.1.1)) within the reducibility interpretation of System U^-.

System U (first formulated in [Gir72]) is just System U^- extended with quantification over universes, i.e. by adding the rules below:

$$\frac{\Gamma \vdash N : \forall \mathcal{X}\sigma[\kappa/\mathcal{X}]}{\Gamma \vdash M : \sigma}\ (\forall^U E) \qquad \frac{\Gamma \vdash M : \sigma \quad \mathcal{X} \text{ bindable in } \Gamma}{\Gamma \vdash M : \forall \mathcal{X}\sigma}\ (\forall^U I) \tag{2.4.7}$$

The Systems U and U^- can be easily interpreted in Martin-Löf's impredicative type theory [ML70b] (section (4.3.2)). Historically, Girard found the paradox that bears his name (appendix (B)) in System U and was then able to reproduce it in Martin-Löf's type theory. The connection between the two system is not *prima facie* evident, because in Martin-Löf's type theory there is no distinction between propositions and types, nor between types and universes: indeed an object of type ν can be either a proposition, either a universe.

Remark that, as a consequence of the reducibility theorem of System F, one has a reducibility theorem for the propositions of System U and U^- of the form: every proposition has a (unique) normal form.

However one cannot extend reducibility to the terms typable in such systems: Girard's paradox ([Gir72], see appendix (B)) provides an example of a non reducible though typable λ-term. The question of the consistency of the apparently weaker System U^- was solved negatively in [Coq94], where a paradox (i.e. a non normalizing typable term) is described for that system. [Coq94] also contains a Curry-Howard presentation of System U^- in connection with a system called *Polymorphic Higher Order Logic*, an extension of Curry's type theory with polymorphic types.

The analysis of these paradoxes constituted for the author the main source of intuitions and ideas for the investigations pursued in chapter (6). The reader will find in appendix (B) an analysis of Girard's paradox, which follows essentially [Hur95], from the viewpoint of typability; this analysis provides at the same time an insight into the typing properties of these violently impredicative type systems and an introductory example to the perspective developed in chapter (6).

2.4.3 A naïve type theory

Church's type theory introduced the idea that propositions can be constructed as typed λ-terms. In order to describe the type disciplines for propositions, in the last subsection we introduced pure constructors and associated, with each type system, a set of typing rules for constructors.

It is natural then to consider the possibility of a "naïve" type system, whose constructors are not typed. This means that every pure constructor can be seen as an element of the universe *prop*. This type system bears some analogies with naïve set theory: as we did for System F^ω and System U we can call a constructor of the form $\lambda\gamma.C$ a set, and write the application of a set C to a constructor D as $D \in C$; then we can write the usual rules of β-expansion and β-reduction as

$$\frac{C[D/\gamma]}{D \in \lambda\gamma.C}\ (\beta - exp) \qquad\qquad \frac{D \in \lambda\gamma.C}{C[D/\gamma]}\ (\beta - red) \tag{2.4.8}$$

The rules above closely resemble Prawitz's rules for naïve set theory (see subsection (3.1.2)):

$$\frac{A[t/x]}{t \in \{x|A\}}\ (set - I) \qquad\qquad \frac{t \in \{x|A\}}{A[t/x]}\ (set - E) \tag{2.4.9}$$

This is why we chose to call such a system System N, where N stands for "naïve".

The rules of System N are very simple, since there are no rules for universes: they are indeed just the rules of System F plus the (β) rule (already present in F^ω).

$$\boxed{\begin{array}{cc}
\dfrac{}{\Gamma, (x:\sigma) \vdash x : \sigma}\ (id) & \dfrac{\Gamma \vdash M : \sigma \quad \sigma =_\beta \tau}{\Gamma \vdash M : \tau}\ (\beta) \\
\dfrac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\ (@) & \dfrac{\Gamma, (x:\sigma) \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\lambda) \\
\dfrac{\Gamma \vdash M : \forall\alpha\sigma}{\Gamma \vdash M : \sigma[\tau/\alpha]}\ (\forall E) & \dfrac{\Gamma \vdash M : \sigma \quad \alpha \text{ bindable in } \Gamma}{\Gamma \vdash M : \forall\alpha\sigma}\ (\forall I)
\end{array}} \tag{2.4.10}$$

The structural properties of System N closely resemble those of System F (except for normalization, obviously), but one has to take into account the existence of a not normalizing reduction relation over types. In particular one can prove the following two properties (whose proofs can be found in (A)):

Proposition 2.4.1 (subject reduction lemma in [BAGM92])**.** *Let $\Gamma \vdash M : \sigma$ be derivable in N and let $M \rightsquigarrow M'$. Then $\Gamma \vdash M' : \sigma^*$ is derivable for some σ^* such that $\sigma \rightsquigarrow \sigma^*$.*

This proposition says that the reduction relation over terms is preserved by the type systems.

Proposition 2.4.2. *Let M be a normal term and $\vdash M : \mathbf{N}$ be derivable in N; then there exists a positive integer $n \in \mathbb{N}$ such that $M \equiv \lambda f.\lambda x.(f)^n x$, where $\equiv$ denotes syntactic equality.*

A simple corollary of the propositions above ensures that normal terms of type $\mathbf{N} \to \mathbf{N}$ in system N can still be considered as codes for recursive functions (though we can no more be sure that those functions are actually total ones).

Corollary 2.4.1. *Let M be a normal term and $\vdash M : \mathbf{N} \to \mathbf{N}$ be derivable in N; then, for all* Church *integer* $\mathbf{n}$, $(M)\mathbf{n}$ *is either not normalizable either it reduces to a* $\mathbf{m}$, *for a positive integer* $m \in \mathbb{N}$.

Since the reduction behavior of M can be coded by a recursive function, it follows that there exists a partial recursive function f such that $f(n)$ is defined and equal to m iff $M\mathbf{n}$ is weakly normalizable and has normal form $\mathbf{m}$.

Fixpoint types System N allows the definition of types by means of fixpoint operators: the combinator $(\delta)\delta$ of λ-calculus, seen as a pure constructor, is a set such that, if $\sigma[\alpha]$ is a type with a free variable α, then $fix_\sigma := (\Delta)\lambda\alpha.\sigma$ is a type which satisfies

$$fix_\sigma = \sigma[fix_\sigma/\alpha] \tag{2.4.11}$$

There exists a quite vast literature on types satisfying equations like the one above: for instance in [ML86] and [Pal90] one finds the analysis of extensions of Martin-Löf's type theory by means of fixpoint operations. In the computer science literature several extensions of simple type theory or System F with fixed point types (usually called *recursive types*) are investigated (see for instance [CC91, Men87]).

All type systems containing a fixpoint operator are inconsistent and, then, not normalizing. For instance, Russell's paradox can be typed in N by using the type **Rus** below

$$\mathbf{Rus} := \big(\forall\alpha((\alpha)\alpha \to \bot)\big)\forall\beta((\beta)\beta \to \bot) \tag{2.4.12}$$

where we may take $\bot$ as $\forall\gamma.\gamma$. Since **Rus** is β-equivalent to $\mathbf{Rus} \to \bot$ one has that $\lambda x.(x)x$ can be given type $\mathbf{Rus} \to \bot$; again, since **Rus** is β-equivalent to $\mathbf{Rus} \to \bot$, $\lambda x.(x)x$ can be given type $(\mathbf{Rus} \to \bot) \to \bot$ and thus it can be applied to an isomorphic copy of $\lambda x.(x)x$. As a consequence, we succeed in typing the not normalizing term $(\lambda x.(x)x)\lambda y.(y)y$ of type $\bot$.

Remark that the type **Rus** used to type the λ-term $(\lambda x.(x)x)\lambda y.(y)y$ is not normalizing. In chapter (6) it will be shown (lemma (6.3.6)) that if $(\lambda x.(x)x)\lambda y.(y)y$ is typable, then its types cannot be in normal form.

The expressive power of System N is *prima facie* extremely big: if one takes as $C^\sigma(\alpha)$ the constructor $\alpha \to \sigma$, for an arbitrary type σ, then the type $\Delta_{C^\sigma} =_\beta \Delta_C \to \sigma$ allows to type every λ-term (indeed the type **Rus** is of the form $\Delta_{C^\bot}$). This impression will be indeed disproven at the end of chapter (6), where it is shown that, if we exclude fixpoint types (which can always be used to type not normalizing λ-terms), then the typability in System N essentially corresponds to the one of the Systems U and U^-.

Part II

Explaining why

Chapter 3

Inferentialist and interactionist interpretations of proofs

An interpretation of proofs is obtained by associating derivations, in a suitable formal system, with certain "constructions", which might be informal entities or concrete mathematical objects. The interest of an interpretation of proofs is twofold: first, it can be used to attach meaning to formulae and to the logical constants occurring in them[1]. This proof-theoretic meaning is given by stipulating the conditions under which a "construction" can be considered as an *evidence* for, or a *realizer* of the formula. Second, it can be used to provide a proof-theoretic notion of validity for derivations and to derive *soundness* theorems of the form: if d is a derivation of a formula A, then its associated "construction" is an evidence for, or a realizer of A.

In this chapter we present two quite distinct, though historically and conceptually related, approaches to the interpretation of proofs and the connected notions of proof-theoretic meaning and validity. On the one hand, we recall some of the main ideas coming from the proof-theoretic semantics tradition, arising from Prawitz's work on natural deduction and Dummett's program of a philosophical foundation of deductive inference; on the other hand, we recall some of the ideas connected with Kleene's realizability interpretation and, more recently, with the Tait-Girard reducibility technique, and try to reconstruct from those ideas a coherent proof-theoretic approach. The exposition will be limited to the case of first order logic; the more controversial situation of second order logic will be discussed in detail in the next chapter.

In addition to constituting a background for the next chapters, this chapter contains an attempt at confronting two traditions which, though sharing a common origin in Gentzen's transformational proof-theory and constructivism, developed in a quite independent way.

3.1 Proof-theoretic validity

In a series of papers ([Pra71a, Pra71b, Pra74]) Prawitz laid down the foundations of a proof-theoretical approach to the notions of validity and logical consequence, i.e. an approach which takes the notion of proof (and its transformations) as central rather than the notion of truth and the connected notion of model.

[1] In the following pages we'll refer to the tasks of providing meaning to logical formulae and of providing meaning to the logical constants as essentially equivalent tasks, since the meaning of a logical formulae is stipulated on the basis of the logical constant which occurs in it as its *principal* operator.

At the basis of Prawitz's project was a criticism of the standard model-theoretical approach to validity:

> Whether e.g. a sentence $\exists x \neg P(x)$ follows logically from a sentence $\neg \forall x P(x)$ depends according to this definition on whether $\exists x \neg P(x)$ is true in each model (D, S) in which $\neg \forall x P(x)$ is true. And this again is the same as to ask whether there is an element e in D that does not belong to S whenever it is not the case that every e in D belongs to S, i.e. we are essentially back to the question whether $\exists x \neg P(x)$ follows from $\neg \forall x P(x)$. [Pra74]

In definitive, Prawitz's criticism amounted to the claim that Tarski's definition of logical consequence, though extensionally correct, does not provide any clue as to why a certain sentence should be taken as a consequence of another one, or to why a certain sentence should be taken as valid while another should not: indeed, the model-theoretic explanation relies on those rules whose meaning it is supposed to explain (see section (4.3.1)). By contrast, Prawitz proposed to redefine the usual semantical notions starting from a definition of *valid argument* and, in particular, an interpretation of proofs. It is not among the aims of this chapter to evaluate this contraposition; we will limit ourselves to reconstruct Prawitz's notion of validity; by the way, in the next chapter, we'll find forms of explanatory circularity very similar to the one ascribed to model-theoretic semantics, when dealing with second order extensions of proof-theoretic validity.

Prawitz's papers and ideas constituted the starting point for the *proof-theoretic semantics* program (see [SH91, SH12]): this is a program in the philosophy of logic, arising from the works by Dummett and Prawitz himself in the 70s, which aims at showing how deductive inference can be justified by relying on the meanings assigned to the logical constants by means of the interpretation of proofs.

Proof-theoretic semantics is not a direct consequence of the acceptance of a proof-theoretical notion of validity, since it relies on the thesis (usually called the *verificationist thesis*, see below), vaguely inspired by some remarks by Gentzen, that the meaning of a logical constant is determined by its introduction rules. In particular, the technique of *computability* predicates in proofs of normalization in type theory (which will be presented in the next section) is historically and conceptually tied to Prawitz's notion of validity, but is not in accordance with the verificationist thesis (section (3.2.2)).

In this section we briefly present and discuss some of the motivations for a proof-theoretic approach to validity and we recall the basic ideas of proof-theoretic semantics.

3.1.1 Meaning and implicit definitions

Before entering into the details of the interpretation of proofs which is usually referred to as proof-theoretic semantics, something must be said about the conception of meaning (and thus, of semantics) which underlies this perspective.

Meaning as use: first interpretation A characteristic aspect of the proof-theoretic approaches is the idea that the meaning of the logical constant lies in the concrete conditions of the their use (as occurring as principal operators in logical sentences): if a natural deduction frame is adopted (as it is often the case in this tradition) then such conditions are identified with the introduction and elimination rules associated to the logical constants. This idea was already contained in some remarks by Gentzen (see below) in his 1934 thesis [Gen64], and i usually associated with a well-known remark by Wittgenstein in [Wit09, Wit78] (see below)

> For a *large* class of cases of the employment of the word "meaning" - though not for all - this way can be explained in this way: the meaning of a word is its use in the language. [Wit09] §43

A second interpretation of the Wittgenstein's"meaning as use" motto will be sketched in subsection (3.2.3).

Such a conception of meaning has to be contrasted with the view which takes truth-conditions (for instance, truth-tables) as determining the meaning of the logical constants and which considers deductive inference justified as it preserves truth from premisses to conclusion: the usual model-theoretic notions of validity and logical consequence are usually applied to devise a formal frame for this view [Tar83].

In definitive, in contrast with the model-theoretic conception of meaning (charged by Prawitz of running into a form of explanatory circularity), the proof-theoretic conception aims at a vindication of logic within the description of the practice of proving and deriving consequences from assertions (as far as this practice can be formalized within a suitable proof-system).

Self-justifying rules Opposed to the idea that the justification of logical rules comes from the preservation of model-theoretic truth, and in accord with the "meaning as use" motto, stands the thesis that (at least some of) the logical rules must be taken as self-justifying, i.e. as demanding for no justification; in [Dum91b] Dummett describes a self-justifying rule as simply a rule that we *treat* as immediately valid. Dummett takes the admission of some rules as self-justifying as a condition for the possibility itself of a proof-theoretical justification of logic:

> [...] we cannot have a proof theory unless we have some means of proof. If, then, there is to be a general proof-theoretic procedure for justifying logical laws, uncontaminated by any ideas foreign to proof theory, there must be some logical laws that can be stipulated outright initially, without the need for justification, to serve as a base for the proof-theoretic justification of other laws. [Dum91b]

The link with the "meaning as use" view is that a rule (for the introduction or elimination of a logical constant) which is taken as self-justifying, is part of an implicit definition of that constant, i.e. as *meaning-constitutive* for that operator: understanding its meaning corresponds then to accepting the rule as valid. As Boghossian explains

> It is by arbitrarily stipulating that [...] certain inferences are to be valid that we attach a meaning to the logical constants. [Bog96]

This conception stands in open contrast with the model-theoretic view, according to which the meaning of a sentence is given by the conditions which determine it as true and a rule is valid when it preserves the truth from the premisses to the conclusion. The roots of this opposition can be traced back to a well-known debate occurred at the end of the 19th century between Frege and Hilbert: the latter, in his *Grundlagen der Geometrie*, was explicitly advancing the idea that the axioms of a certain geometry constitute an implicit definition of the geometrical notions involved. Frege replied to Hilbert in a letter in 1899, fiercely opposing the view that it is up to definitions to fix the meaning of sentences and the denotation of terms, and that axioms should express truths.

> [Axioms and theorems] must not contain a word or sign whose sense and meaning, or whose contribution to the expression of a thought, was not already completely laid down, so that there is no doubt about the sense of the proposition and the thought it expresses. The only question can be whether this thought is true and what its truth rests on. Thus axioms and theorems can never try to lay down the meaning of a sign or a word that occurs in them, but it must be already laid down. [Fre80]

Reading the *Grundlagen* under this perspective, Frege observed that

> [...] the meanings of the words "point", "line", "between" are not given, but are assumed to be known in advance. [Fre50]

In his answer to Frege, Hilbert strongly rejected Frege's reading:

> I do not want to assume anything as known in advance. I regard my explanation [...] as a definition of the concepts point, line, plane [...] If one is looking for other definitions of a "point", [...] one is looking for something one can never find because there is nothing there. [Fre50]

In [Cof91] Coffa describes the view defended by Hilbert in Kantian terms as one of the first steps towards a "Copernican turn in semantics":

> Meanings are constituted roughly in the way in which Kantians used to think that we constitute experience or its objects, through the employment of rules or maxims whose adoption is prior to and the source of the meanings in question. [Cof91]

The mature development of such a semantical turn, in Coffa's reconstruction, can be found in the writings by Carnap and Wittgenstein in the 1930's: the first, in [Car37], defended the view that axioms and rules of a formal system implicitly define the meaning of the logical symbols.

> Let any postulates and any rules of inference be chosen arbitrarily; then this choice, whatever it may be, will determine what meaning is to be assigned to the fundamental logical symbols. [Car37]

In particular Carnap's conception allowed to retrieve the ancient notion of *analyticity*, or "truth by virtue of meaning": since the meanings of the logical sentences are determined by the rules and axioms involving them, all theorems of a formal logical system shuld be taken as analytically true.

In the same years Wittgenstein was defending a similar position (in contrast with the ideas made popular with the *Tractatus* [Wit01]): he held that the sole vindication of logical inference lied in the practice of accepting its defining rules: in a word, the rules of logic would not be infallible because of some property they enjoy ("In what sense is logic something sublime?" [Wit09], ¶89), but just because we have been learned to *treat* them as infallible.

> But doesn't e.g. 'fa' have to follow from '$(x)fx$', if '$(x)fx$' is meant in the way we mean it?" - And how does the way we mean it come out? Doesn't it come out in the constant practice of its use? [...] One learns the meaning of '(x)' by learning that 'fa' follows from '$(x)fx$'. [Wit78]

Wittgenstein's "meaning as use" doctrine has here the consequence of inverting the direction of explanation of the role of logic with respect to language: logic would not have an exceptional, normative role in language because of its nature, but rather the nature of logic would be given by the exceptional, normative role that it plays in linguistic practices.

Inference and analyticity As it is well-known Quine in the 1950s had presented a series of arguments (contained in [Qui53] and [Qui76]) against the use of the notion of analyticity in the explanation and justification of logical rules, with an explicit reference to Carnap's doctrine of implicit definitions. The development of the proof-theoretic semantic conception between the 1970s and the 1980s had, among its consequences, the one of revitalizing the debate in the philosophy of logic over analyticity.

Indeed, in proof-theoretic semantics the meaning of a logical constant is given by the set of self-justifying rules involving that operator. From an epistemological point of view, this implies that the knowledge of the meaning of a logical constant is enough to be justified in taking its meaning-constitutive rules as valid.

In [Bog96], Boghossian acknowledges that Quine's arguments lead to a rejection of a *metaphysical* notion of analyticity: he calls a sentence metaphysically analytic when its truth-value is determined by its meaning. Similarly we can call an inference metaphysically analytic if its truth-preservation is determined by the meaning of the premisses and the conclusion. The rejection of the metaphysical notion undermines a semantic justification of logical inference based on the idea that the meaning of a logical sentence is determined by its truth-conditions.

At the same time Boghossian tries to defend the view that Quine's rejection can be escaped if one endorses an inferentialist conception of meaning, as the one involved in the thesis that rules work as implicit definitions of the logical constants. In particular, Boghossian claims that Quine's argument leaves room for the development of an *epistemic* notion of analyticity (see [Bog96, Bog03]): a sentence is epistemically analytic if mere grasp of its meaning suffices for being justified in holding it true; an inference is epistemically analytic if mere grasp of the meaning of the premisses and the conclusion suffices for being justified in holding it valid. On this reading the self-justifying rules for a logical constant turn out to be epistemically analytic.

3.1.2 Consistency and the inversion principle

An obvious objection to the implicit definition conception is that, by admitting that whatever rule can be taken as implicitly defining a logical constant, one runs into serious problems of justification: for instance, if a contradiction can be derived from a given system of rules or axioms, in what sense can the use of those rules and axioms be considered justified (or self-justified)?

The advocate of proof-theoretic semantics would answer that the rules of logic are not purely arbitrary, as the enjoy some structural properties (arising from Prawitz's inversion principle - see subsection (2.1.1)) which allow to reject some pathological examples (as the one notoriously proposed by Prior in [Pri67]).

However, in chapter (1) we remarked that a consequence of Gödel's incompleteness theorems is that a sharp distinction must be made between properties that can be established combinatorially or recursively ("how proof theory") and properties, like consistency, which demand for logically complex arguments ("why proof theory").

Hence Prawitz's inversion principle, which is a *local*, combinatorial, criterion, must be distinguished from the *Hauptsatz*, a *global* criterion, which implies consistency.

Implicit definitions and contradictions The conceptions of Hilbert, Carnap and Wittgenstein sketched above diverge on the problem of contradictions: in [Wit78] Wittgenstein, as it is well-known, defended the idea that all rules gain their legitimacy from the concrete practice of language, and in particular logical rules gain their epistemological status (of deductively valid ones) from the role attributed to them in the use of language. As a consequence, he considered all matters as to the justification of logical rules as devoid of sense. In [Wit89] he even tries to argue for the substantial harmlessness of contradictions (as those arising from Russell's paradox).

By contrast, as it is well-known, in the formalist program developed by Hilbert, a set of axioms can be taken as an implicit definition of a mathematical entity only when satisfying a *criterion of non-contradiction*:

> If contradictory attributes be assigned to a concept, I say, that mathematically the concept does not exist. So, for example, a real number whose square is -1 does not exist mathematically. But if it can be proved that the attributes assigned to the concept can never lead to a contradiction by the application of a finite number of logical inferences, I say that the mathematical existence of the concept (for example, of a number or a function which satisfies certain conditions) is thereby proved. [Hil96b]

In definitive, if we do not want to admit as valid an inference which can be used to derive a contradiction, it appears that implicit definitions should be supplemented with some form of warrant that they won't lead to a contradiction. But, since at least one of the purposes of a definition of validity for sentences and inferences is to have a warrant that they do not lead to contradiction, this seems tantamount to say that we can define validity by means of implicitly defining inferences, provided that the latter are valid inferences: a viciously circular explanation.

A similar objection is often advocated against defenders of an epistemic conception of analyticity: since the reason for judging an inference analytic is that this inference must be in a sense compelling, an implicitly defining inference should be supplemented with a warrant that a speaker is actually entitled to draw the its conclusion from its premisses (for instance, as Peacocke agues in [Pea93], by the warrant that the inference is truth-preserving).

In this context Carnap's position is of some interest: in [Car37] he adopts a liberalist position as to logical rules:

> No question of justification arises at all, but only the question of the syntactical consequences to which one or other of the choices leads, including the question of non-contradiction. [Car37]

In the same text he remarks that the evaluation of a formal system on the basis of its syntactical properties (like non-contradiction) is made on a purely pragmatic basis. It must be remarked here how Carnap seems to consider the question of non-contradiction as a finite, combinatorial matter (a "syntactical consequence"), devoid of a genuine epistemological interest.

By contrast it should be remarked that, by Gödel's second incompleteness theorem, the question of the non-contradiction has a deep epistemological content: it was just the fact that the argument for the satisfaction of such a criterion for an arithmetical theory could not be formalized within the theory itself which was at the origin of the failure of Hilbert's program (see subsection (4.3.1)).

The inversion principle In a famous paper ([Pri67]) Arthur Prior, in order to argue against the implicit definition conception, presented a weird connective, *tonk*, whose implicitly defining rules are listed below

$$\frac{A}{AtonkB}\ (tonk-I)_1 \qquad \frac{B}{AtonkB}\ (tonk-I)_2 \qquad \frac{AtonkB}{A}\ (tonk-E)_1 \qquad \frac{AtonkB}{B}\ (tonk-E)_2 \tag{3.1.1}$$

Since, by successively introducing and eliminating *tonk*, every formula can be derived, the acceptance of the *tonk* connective as a meaningful logical constant leads to contradictions.

Prior's example provoked a vast debate over the legitimacy of a purely conventionalist interpretation of logic. The by now "standard" proof-theoretical response to Prior is the remark that the rules of logic are not purely conventional, since they are supposed to satisfy some structural properties. In order to describe such properties, we have to get back to Gentzen's transformational approach.

When defining Gentzen transformations over derivations, we have to consider cuts whose premisses are respectively obtained by means of right and of a left rule for the same logical constant (see chapter (2)). In such cases the transformation consists in deleting the two rules introducing the logical constant on the two sides of the sequents and introducing cuts between the remaining subderivations.

The translation of this operation in the language of natural deduction leads to a *normalization procedure* for derivations (see [Pra65]): by a cut it is meant the occurrence of an introduction rule for a logical constant immediately followed by an elimination rule for the same logical constant; the Gentzen transformation in this case applies to the derivation in order to produce a derivation in which the two rules are deleted. For instance, in the case of implication, a cut corresponds to the occurrence of the following situation in a derivation d:

$$\cfrac{\cfrac{\begin{array}{c} A \\ \vdots \\ B \end{array}}{A \Rightarrow B}\ (\Rightarrow I) \qquad \begin{array}{c} \vdots \\ A \end{array}}{\begin{array}{c} B \\ \vdots \end{array}}\ (\Rightarrow E) \tag{3.1.2}$$

which can be reduced to the derivation d below, where the occurrences of the rules $(\Rightarrow I)$ and $(\Rightarrow E)$ have been eliminated:

$$\begin{array}{c} \vdots \\ A \\ \vdots \\ B \\ \vdots \end{array} \tag{3.1.3}$$

Prawitz's *inversion principle* (subsection (2.1.1)) states indeed that such transformations must always be performable, if a cut occurs in a derivation. This principle can indeed be seen as a principle for the justification of a logical constant: it says that the conditions which allow for the assertion of a sentence in which a logical constant occurs as principal operator must be enough for justifying the assertion of an immediate consequence of this sentence.

We can use the inversion principle to reject Prior's connective *tonk*: in order to derive a contradiction one has to use a *tonk*-introduction (given a derivation of an arbitrary formula A) immediately followed by a *tonk*-elimination, as below:

$$\cfrac{\cfrac{\begin{array}{c} \vdots \\ A \end{array}}{A tonk \bot}\ (tonk - I)_1}{\bot}\ (tonk - E)_2 \tag{3.1.4}$$

now, since the two rules $(tonk - I)_1$ and $(tonk - E)_2$ do not satisfy an inversion principle, the derivation above cannot be normalized.

By the way, the inversion principle does not constitute a sufficient criterion for avoiding contradictions from arbitrarily stipulated rules. A counterexample can be found already in Prawitz's book [Pra65]: there he defines a natural deduction version of naïve set theory, made of the following two rules (corresponding to the naïve comprehension principle):

$$\frac{A[t/x]}{t \in \{x|A\}}\ (set - I) \qquad\qquad \frac{t \in \{x|A\}}{A[t/x]}\ (set - E) \tag{3.1.5}$$

the rules above satisfy the inversion principle, as

$$\cfrac{\cfrac{\begin{array}{c} \vdots \\ A[t/x] \end{array}}{t \in \{x|A\}}\ (set - I)}{\begin{array}{c} A[t/x] \\ \vdots \end{array}}\ (set - E) \tag{3.1.6}$$

can be reduced to

$$\begin{array}{c}\vdots\\ A[t/x]\\ \vdots\end{array}\tag{3.1.7}$$

At the same time, Russell's paradox can be reproduced within this system: in particular, by letting t be the set $\{x|x \in x \Rightarrow A\}$, the derivation d_{Rus} below can be built

$$\dfrac{\dfrac{\dfrac{\dfrac{[t\in t]^x}{t\in t\Rightarrow A}\,(set-E) \qquad [t\in t]^x}{A}\,(\Rightarrow -E)}{t\in t\Rightarrow A}\,(\Rightarrow I)^x \qquad \dfrac{\dfrac{\dfrac{\dfrac{[t\in t]^y}{t\in t\Rightarrow A}\,(set-E) \qquad [t\in t]^y}{A}\,(\Rightarrow -E)}{t\in t\Rightarrow A}\,(\Rightarrow I)^y}{t\in t}\,(set-I)}{A}\,(\Rightarrow -E)\tag{3.1.8}$$

for an arbitrary formula A (for instance $A = \bot$). d_{Rus} ends with a cut made of the rules $(\Rightarrow -I)$ and $(\Rightarrow -E)$, and one easily verifies that, by normalizing this cut, a derivation identical to d_{Rus} is produced, i.e. the normalization procedure diverges.

This example shows that *the existence of a well-defined reduction procedure over derivations is not sufficient for characterizing valid inferences*: the possibility to locally reduce proofs belongs to "how proof theory" (it can be entirely described in a recursive way), whereas the fact that all such reductions terminate producing a normal form belongs to "why proof theory" (as Gentzen's *Hauptsatz* is expressed by a Π^0_2 formula). This fact will appear more clear when we look at derivations from the "forgetful" viewpoint, i.e. as pure λ-terms, and at the normalization procedure as the execution of those terms (see (3.2)).

3.1.3 Proof-theoretic semantics

We start our short description of the perspective of proof-theoretic semantics by recalling the two main sources of the proof-theoretical interpretation: the BHK interpretation of intuitionistic proofs, and Gentzen's remarks on the role of introduction rules as implicit definitions of the meaning of the logical constants.

The BHK *interpretation* The idea of a semantics centered on the notion of proof has to be traced back to the so-called BHK *interpretation* of intuitionistic proofs. This is usually acknowledged as the first example of an (informal) interpretation of logic defined at the level of proofs. BHK is an informal semantics in which proofs are interpreted as certain "constructions" (we discuss this ambiguous notion in the next section). In particular, this interpretation is obtained by a series of clauses which state the conditions under which a certain formula can be asserted: the interpretation of proofs can then be seen also as an assignment of meaning to logical formulae, where the meaning of a formula is given by stating under which circumstances a "construction" can be seen as a proof of that formula.

The most well-known source for the BHK interpretation is [Hey56]:

> The *conjunction* $\wedge$ gives no difficulty: $p \wedge q$ can be asserted if and only if both p and q can be asserted.
>
> I have already spoken of the *disjunction* $\vee$. $p \vee q$ can be asserted if and only if at least one of the propositions p and q can be asserted.
>
> The *negation* $\neg$ [...] $\neg p$ can be asserted if and only if we possess a construction which from the supposition that a construction p were carried out, leads to a contradiction.

> [...]
> The *implication* $p \to q$ can be asserted if and only if we possess a construction r which, joined to any construction proving p (supposing that the latter be effected), would automatically effect a construction proving q. In other words, a proof of p, together with r, would form a proof of q. [Hey56].

This interpretation was indeed one of the first attempt towards a dynamical presentation of logic, since a proof of an implication was described in terms of how it could be used in order to transform other proofs; a precise connection with the *Curry-Howard correspondence* between proofs and programs will be discussed in the section (3.2), by exploiting the realizability interpretation.

Gentzen's remarks and verificationism Gentzen's doctoral thesis [Gen64] contains a series of brief remarks in which he states that the introduction rules of natural deduction calculus work as definitions of the "meaning" of the logical constants, and that the elimination rules are, in a sense, derived from the former.

> The introductions represent, as it were, the "definitions" of the symbols concerned, and the eliminations are no more, in the final analysis, than the consequences of these definitions. This fact may be expressed as follows: in eliminating a symbol, the formula, whose terminal symbol we are dealing with, may be used only "in the sense afforded it by the introduction of that symbol."
> [...]
> By making these ideas more precise, it should be possible to devise the E-inferences as single-valued functions of their corresponding I-inferences, on the basis of certain requirements. [Gen64]

The proponents of proof theoretic semantics (see for instance [SH12]) interpret the intuitions contained in these remarks by means of two theses: first, the thesis of *implicit definitions*, that is, the already discussed thesis that some rules of natural deduction can be considered as an implicit definition of the logical constants; second, the *verificationist thesis*, asserting that the meaning-constitutive rules are the introduction rules, and that the elimination rules are indeed "derived" or justified with respect to the meaning fixed by the former.

The verificationist thesis The explanation of the verificationist thesis comes from two remarks: firstly, it embodies the idea, coming from the BHK interpretation, that the meaning of a sentence is given by specifying the form of its proofs (or "verifications", in Dummett's terminology). Dummett opposes this idea to a "pragmatist" conception, for which the meaning of a sentence is given by specifying how to derive consequences from it. In this sense the verificationist thesis appears quite natural for an approach based on proofs.

By the way, the idea that the meaning of a logical constant C is given by what counts as a proof of a formula in which C occurs principally must be kept distinct from the old empiricist idea that the meaning of a sentence is given by its (experimental) verifications: as Prawitz points out

> [...] according to the verificationism of today, to know the meaning of a sentence it is sufficient to know what counts as a verification of the sentence, one does not need to know a method that in principle verifies or refutes the sentence. [Pra02].

As Dummett writes

> just this mistake was one of the two dogmas of empiricism repudiated by Quine. [Dum91b]

Indeed, it is already part of the BHK interpretation that the knowledge of the meaning of a formula does not provide a way to explicitly construct a proof, but involves the ability to recognize the form that such a proof, if any, should have. Thus, the verificationist thesis should be read as stating that the meaning of a sentence (in which a certain logical constant occurs principally) is given by the conditions under which this can be proved, where such conditions are stipulated by a recursive definition in the style of the BHK interpretation.

The second remark comes from the fact that admitting all forms of verification (i.e. of proofs) as meaning-constitutive would amount to admitting all rules as meaning-constitutive for the logical constants. Indeed, an arbitrary rule can potentially occur in a proof of an arbitrary formula (that is exactly what distinguishes a arbitrary derivation from a cut-free one, enjoying the subformula property).

By elaborating Gentzen's intuition that the introduction rules are the only meaning-constitutive ones, a distinction between two forms of proofs was then proposed: a *canonical* proof of a formula whose principal operator is C is a proof which ends with an introduction rule for C; this means that *the last step* of the proof is taken in accordance with the meaning attached to C (it is, in Boghossian's terminology, epistemically analytic). A non canonical proof is a proof which is not canonical.

The distinction between canonical and non canonical derivation constitutes the essential ingredient of the proof-theoretic definition of validity of proofs. First, the validity of a canonical derivation can be defined by an induction on the sum of the complexities of its premisses and its conclusion: since the last rule of the derivation is an introduction rule, and thus immediately valid, it is enough to verify that the sub-derivations which have those premisses as conclusion are valid; now, the premisses of the rule are subformulae of the conclusion, and are thus of smaller complexity.

Second, the definition of validity for non-canonical derivations requires the appeal to Gentzen's transformations, which allows to reduce the derivation in canonical form. Indeed, since the last rule of a non canonical derivation might not be an introduction, the inductive definition above does not work. One has then to rely on the inversion principle in order to transform the derivation into a canonical one: as Martin-Löf puts it in [ML84], a non-canonical proof can be seen as a "method which, when applied, produces a canonical proof".

In the definition of validity (that we sketch below) the normalization procedure (i.e. cut-elimination) assumes thus the role of a *vindication of meaning*: firstly, because the inversion principle can be restated as a semantical principle for the local justification of the elimination rules with respect to meaning. Dummett's *harmony requirement* is indeed a general reformulation of that principle:

> We say that harmony, in the general sense, obtains between the verification-conditions or application-conditions of a given expression and the consequences of applying it when we cannot [...] establish as true some statement which we should not have had other means of establishing [...] The analogue, within the restricted domain of logic, for an arbitrary logical constant c, is that it should not be possible, by
>
> first applying one of the introduction rules for c and then immediately drawing a consequence from the conclusion of that introduction rule by means of an elimination rule of which it is the major premiss, to derive form the premisses of the introduction rule a consequence that we could not otherwise have drawn.
>
> [...]
>
> The requirement that this criterion for harmony be satisfied conforms to our fundamental conception of what deductive inference accomplishes. An argument or proof convinces us because we construe it as showing that, given that the premisses hold good according to our

> ordinary criteria, the conclusion must also hold *according to the criteria we already have for its holding.* [Dum91b]

Secondly, since, as we saw in the preceding section, the inversion principle is not powerful enough to characterize validity, a stronger condition is required, namely that an arbitrary closed derivation can be transformed into a canonical one, what Dummett calls the *fundamental assumption*:

> But the justification depends heavily upon what we may call the "fundamental assumption": that, if we have a valid argument for a complex statement, we can construct a valid argument for it which finishes with an application of one of the introduction rules governing its principal operator. [Dum91b]

Such an assumption, at least in the $\vee, \exists$-free fragment of intuitionistic logic, can be proved as a simple corollary of the normalization theorem for first order intuitionistic natural deduction.

Remark that, trivially, there exists no canonical proof of the absurd, since the latter has no introduction rules. As a consequence, from the normalization theorem (or, from the fundamental assumption) it follows that no non canonical proof of the absurd exists: if it existed, it would reduce to a canonical one.

Hence, once again, we must distinguish between the inversion principle (a local, combinatorial, property) and the fundamental assumption (a global property, implying consistency).

A definition of validity We provide a brief sketch of the definition of proof-theoretic validity for the implicative fragment of intuitionistic logic. The definition below essentially follows Prawitz's definition of *strong validity* in [Pra71a]. For a detailed discussion of the several notions of proof-theoretic validity on the market, the reader can look at [SH06].

The definition is given by a generalized inductive definition: firstly, an induction on the complexity of the conclusion of the derivation; secondly, an induction on the reduction relation between derivations of the same conclusion.

Definition 3.1.1 (Validity for the $\Rightarrow$-fragment of intuitionistic logic)**.** *Let d be a natural deduction derivation of conclusion A. d is valid if either:*

(V1) *$A = B \Rightarrow C$ and d is canonical, i.e. of the form*

$$\dfrac{\begin{array}{c}[B]\\ \vdots \\ C\end{array}}{B \Rightarrow C}\ (\Rightarrow -E) \tag{3.1.9}$$

and for every valid derivation d' of conclusion B, the derivation

$$\begin{array}{c}\vdots\ d' \\ B \\ \vdots \\ C\end{array} \tag{3.1.10}$$

is valid;

(V2) *d is not canonical and normal;*

(V3) *d is not canonical and not normal, and for every derivation d' such that d reduces to d' in one step, d' is valid.*

Proof-theoretic validity, as it implies consistency, is a logically complex concept. In particular, the validity of a derivation with respect to a formula A is expressed by a formula which has at least the logical complexity of A: if A is $B \Rightarrow C$, then a derivation of conclusion A is valid if, for all derivation d', *if* d' is a valid derivation of conclusion B, *then* the composition of d and d' by means of the implication elimination rule is a valid derivation of conclusion C.

A simple consequence of this definition is the following lemma

Lemma 3.1.1. *Every valid derivation is strongly normalizable.*

Proof. We argue by induction on the sum of the complexities of the conclusion and the open assumptions of d (where $compl(A \Rightarrow B)$ is $compl(A) + compl(B) + 1$), with a sub-induction on the reduction relation over derivations.

Let A be $B \Rightarrow C$ and d be canonical. Hence d is of the form

$$\dfrac{\begin{array}{c}[B] \\ \vdots\, d_1 \\ C\end{array}}{B \Rightarrow C}\ (\Rightarrow -I) \tag{3.1.11}$$

Since, d_1 is valid, it follows by the induction hypothesis that it is strongly normalizing, hence d is too.

If d is non canonical and normal, then it is obviously strongly normalizing. If d is non canonical and non normal, then, since all its immediate reducts are valid, and by induction hypothesis, strongly normalizing, it follows that d is too.

□

A second consequence of the definition of validity concerns open derivations, i.e. derivations with open assumptions:

Lemma 3.1.2 (substitution lemma)**.** *Let d be an open derivation of the form*

$$\begin{array}{c}[A_1], \dots, [A_n] \\ \vdots\, d \\ B\end{array} \tag{3.1.12}$$

then d is valid if and only if, for every list of valid derivations $d_1, \dots, d_n$, respectively of conclusion $A_1, \dots, A_n$, the derivation

$$\begin{array}{c}\begin{array}{ccc}\vdots\, d_1 & & \vdots\, d_n \\ A_1 & \dots & A_n\end{array} \\ \vdots\, d \\ B\end{array} \tag{3.1.13}$$

is valid.

We omit the proofs of this lemma so as of the theorem below. The arguments are indeed very similar to the ones presented in the next section for reducibility in type theory (see subsection (3.2.2)).

By a more sophisticated argument (we'll sketch in the next section an argument for type theory which has the same structure), and by relying on the lemma above, it can finally be proved that all derivations are valid.

Theorem 3.1.1. *Every derivation of the $\Rightarrow$-fragment of intuitionistic natural deduction is valid.*

In the next section, we will discuss the *computability* or *reducibility* properties of λ-terms, in the context of type theory, which are similar to the property of validity for natural deduction derivations. Indeed, we will present in some more detail how these properties can be used to prove strong normalization theorems by techniques vary similar to the ones above. The main difference is that, when dealing with the computability of λ-terms, we drop the distinction between canonical and non canonical derivations, and with it the emphasis over introduction rules. As a result, proofs are easier to follow but proof-theoretic semantics is lost. Rather, we will try to propose an alternative to proof-theoretic semantics based on realizability semantics.

We conclude this presentation with some remarks. The proof-theoretic justification here sketched can be read at two distinct levels: at an epistemological level it provides a notion of validity for derivations and formulae by which the validity of all derivations can be reduced (by means of the manipulations arising from Gentzen's transformations) to the validity of derivations whose last step is taken as valid by definition (or "epistemically analytic", in the sense of [Bog96]). At a semantic level it provides a notion of meaning for the logical constants (given by introduction rules) which is preserved by all deductive constructions (i.e. canonical or non canonical derivations). On the one hand, then, cut-elimination is used to assure validity (and coherence); on the other hand, it provides a vindication of the meaning stipulated by means of introduction rules:

> The meanings of our assertoric sentences in general, and of the logical constants in particular, are given to us in such a way that the forms of deductive inference we admit as valid can be exhibited as faithful to, or licensed by, those meanings and involve no modification of them. [Dum91b]

3.2 Realizability and reducibility

In this section we present two different, though intimately related, approaches to the interpretation of proofs: we briefly recall realizability semantics, a quite vast domain of research inaugurated by Kleene's paper [Kle45] and we discuss the technique of reducibility (also known as convertibility or computability) predicates, used to build normalization proofs for type theories. In the last section we discuss some of the features shared by the two approaches, which prompt an alternative view with respect to proof-theoretic semantics.

3.2.1 Realizability semantics

Kleene's recursive realizability The history of realizability starts with Kleene's paper [Kle45], where he provides an interpretation of proofs for intuitionistic (first-order) arithmetics. Kleene's goal was to state a clear connection between the informal intuitionistic notion of construction (as stated for instance in the BHK interpretation[2]) and the notion of recursive computation that had been developed by Herbrand, Gödel and himself.

Kleene's main intuition, as he reports in [Kle73], was the following: intuitionistically, a proof of a Π^0_2 statement $\forall n \exists m A(n, m)$ is a constructive method μ producing, for each integer n, an integer $\mu(n)$ and a proof that $A(n, \mu(n))$ holds; now, on the basis of Church's thesis, which identifies the informal notion of computability with the rigorous one defined by means of the notion of general recursive function, he conjectured that the method μ could be coded by a general recursive function. Kleene's conjecture, as he reports, did not receive a great appreciation

[2]Kleene is indeed quite explicit that, in developing the definition of realizability, he was not really inspired by the BHK interpretation of proofs but rather by Brouwer's texts.

at the time [3]; by the way, it can surely be seen as one of the first intuitions in the direction of what we presented in chapter (2) by the notion of "forgetful functor". Through the realizability interpretation, Kleene was indeed able to express his conjecture in precise terms and finally to prove it.

The basic idea of Kleene's realizability is to define a *realizability relation* between codes and formulae: an intuitionistic proof of a formula A is then translated into a code realizing A. Realizability is defined by means of the following clauses:

i. e realizes $t = u$ if and only if $t = u$ is true;

ii. no e realizes $\perp$;

iii. e realizes $A \Rightarrow B$ if and only if, for all a which realizes A, $\{e\}a$ realizes B;

iv. e realizes $\forall nA$ if and only if, for all integer k, $\{e\}k$ realizes $A[\underline{k}/n]$.

where $\{e\}a$ denotes Kleene's brackets, i.e. the application of the recursive function whose code is e to the integer a. A code e realizing a formula A is called a *realizer* of A.

Kleene was able to show that, from a derivation d in Heyting Arithmetics **HA** of a formula A it is possible to extract a realizer e_d of A. In particular, among the several properties he could establish, he proved the following two:

1) If A is derivable in **HA**, then A is realizable;

2) if $\forall n \exists m A(n, m)$ is derivable in **HA**, then there exists a recursive function f such that $A(n, f(n))$ is realizable.

Kleene himself remarked that the realizability relation was akin to be explicitly formalized by a predicate ® in the language of arithmetics. A complete formalization was obtained by Troelstra in [Tro63], where Kleene's results were internalized within **HA** as follows:

1′) if A is derivable in **HA**, then there exists a code e such that e®A is derivable in **HA**;

2′) if $\forall n \exists m A(n, m)$ is derivable in **HA**, then there exists an e such that both $A(n, \{e\}n)$ and $\forall n \exists m(m = \{e\}n)$ (i.e. the totality of the function coded by e) are derivable in **HA**[4].

These results can actually be seen as the first hints towards the extraction of programs from formal derivations. In particular the theorem **2′**) goes in the direction of theorem (2.3.2), which shows how to extract a provably total recursive function from a derivation of a Π^0_2 formula.

Modified realizability and the forgetful functor Kreisel's approach to realizability in [Kre59] differed from Kleene's original one in that he took realizers not to be arbitrary codes, but rather typed programs (in his vocabulary, functionals of finite type). Kreisel's functionals where defined starting from typed variables $x^\sigma, y^\sigma, \dots$, combinators $\mathbf{k}^{\sigma\to\tau\to\sigma}$, $\mathbf{s}^{(\sigma\to\tau\to\rho)\to((\sigma\to\tau)\to(\sigma\to\rho))}$ (coming from combinatory logic [CF58]) and combinators $\mathbf{r}^{\sigma\to(\mathbf{N}\to\sigma\to\sigma)\to\mathbf{N}\to\sigma}$ for primitive recursion.

The idea of this "modified" version of the realizability interpretation is, first, to assign with each sentence A a type A^* (where A^* is essentially $A^{\mathbb{F}}$); second, to define a realizability relation under the form of a typing relation between programs and types. Kreisel's results can then be summarized as follows:

[3] "*That this plan was not altogether obvious in 1940 is illustrated by the reaction of a prominent logician to whom I explained it at a chance meeting early in 1940. He explained to me reasons why, in his view, the plan could not be expected to succeed. I did not succeed in understanding his reasons*" [Kle73].

[4] Given a suitable encoding of Kleene's brackets.

1″) If A is derivable in $\mathbf{HA}^\omega$, then there exists a program M of type $A^{\mathbb{F}}$ such that M®$A^{\mathbb{F}}$ is derivable in $\mathbf{HA}^\omega$;

2″) if $\forall n \exists m A(n,m)$ is derivable in $\mathbf{HA}$, then there exists a program M of type $\mathbf{N} \to (\mathbf{N} \wedge A^{\mathbb{F}}))$ such that M®$\mathbf{N} \to (\mathbf{N} \wedge A^{\mathbb{F}}))$ is derivable in $\mathbf{HA}^\omega$ and moreover, for all positive integers $k, h \in \mathbb{N}$, $A(\underline{k}, \underline{h})$ is derivable in $\mathbf{HA}$ if and only if $M\mathbf{k}$ reduces to $\mathbf{h}$.

where $\mathbf{HA}^\omega$ is Heyting Arithmetics enriched with finite types or, equivalently, Gödel's System T enriched with predicate logic (see [Kre59]). Remark that **2″** is very similar to the extraction of program given by theorem (2.3.2).

Kreisel's functional interpretation can be seen as one of the first concrete examples of the *formula-as-types* paradigm: he explicitly assigned types to closed formulae and showed how to extract typed programs from derivations. In this sense, it constitutes one of the most striking precursors of the *Curry-Howard correspondence*. As van Oosten remarks in his brief historical reconstruction [VO02]:

> This "typed realizability", defined by Kreisel in 1959 ([Kre59]), predates the slogan "formulae as types" ([How80]) by 10 years! [VO02]

The relationship between Kreisel's realizability and Kleene's can be expressed again through the forgetful translation: if we translate Kreisel's functionals in a typed λ-calculus (for instance, in System T) and then we erase all type information from them, we obtain an interpretation of proofs by means of *untyped* programs which is equivalent to Kleene's interpretation by means of numerical codes. More on this below.

What is realizability? Realizability has actually grown into a quite vast domain of research, so it would be pointless to try to make a list of all of its relevant developments. We can just recall two main axis of research: on the one hand the search for purely mathematical descriptions of the concept of realizability led to the development of a very rich categorial approach to the subject (for instance [Hyl82] is considered as a cornerstone of the categorical approach, see [VO02] for a brief reconstruction); on the other hand one should mention the development of classical realizability, i.e. the extension of realizability to the interpretation of classical proofs by means of control operators (see [Kri09]).

More relevant to the scopes of this very short summary is to try to highlight the main characteristics of the several, and quite different, approaches to the semantics of proofs which go under the name of realizability. A first property of all realizability interpretations is that they are based over a map $|\cdot|$, which associates, with each formula A, a set $|A|$ of programs, i.e. the set of the programs which realize A.

An interpretation presupposes then the choice of a class $\mathcal{P}$ of programs, such that, for each formula A, $|A| \subseteq \mathcal{P}$. The very general notion of *partial combinatory algebra*, or *pca*, (see [Sta73]) captures the ingredients needed to yield a realizability interpretation. A *pca* is essentially a set on which a notion of product $a * b$ is defined (where $a * b$ is to be read as the application of program a to input b), which contains variables and equivalent of the combinators $\mathbf{k}$ and $\mathbf{s}$. In particular, Kleene's codes for partial recursive functions, along with Kleene's bracket, form a partial combinatory algebra, so as Kreisel's functionals of finite type and pure λ-calculus.

Starting from the definition of the realizability relation, the map $|\cdot|$ can be essentially described as follows:

$$|t = u| = \begin{cases} \mathcal{P} & \text{if } t = u \text{ is true} \\ \emptyset & \text{else} \end{cases} \tag{3.2.1}$$

$$|\bot| = \emptyset \tag{3.2.2}$$

$$|A \Rightarrow B| = |A| \to |B| \tag{3.2.3}$$

$$|\forall n A| = \prod_{k \in \mathbb{N}} |A[\underline{k}/n]| \tag{3.2.4}$$

where, given two sets $a, b \subseteq \mathcal{P}$, $a \to b$ denotes the set of programs M such that, for all program $N \in a$, $M * N \in b$.

Thus, with the exception of Kreisel's modified realizability, all these interpretation associate formal derivations with *untyped* programs; in particular, the internal structure of the programs is never questioned in the definition of realizability: all that matters is how the program behaves in certain context. Kleene in particular seems already quite conscious of this aspect:

> A realization number by itself of course conveys no information; but given the form of statement of which it is a realization, we shall be able in the light of our definition to read from it the requisite information. [Kle45]

This is a major difference with respect to the *Curry-Howard correspondence*, where a derivation is usually translated into a typed λ-term; in realizability, derivations are interpreted by arbitrary programs, independently of how such programs are constructed. Here we can see a very strong difference with respect to the rule-based approach of proof-theoretic semantics: there the interpretation of a derivation is based upon the notion of a canonical proof, i.e. of a proof with a peculiar internal structure (connected with its last rule).

A crucial consequence of this untyped approach is that the same program can be a realizer of different sentences: trivially every program is a realizer of every true atomic formula; more interestingly, an untyped combinator for recursion (for instance a type-free version of Kreisel's combinator $\mathbf{r}$) is a realizer of *every* instance of an induction axiom. Such a *polymorphism* of programs will be indeed a crucial ingredient when discussing second order proof-theory.

A second remarkable feature concerns the treatment of atomic sentences; the definition implies indeed a form of *proof-irrelevance* of atomic sentences: if an atomic sentence is true, then whatever program can interpret a proof; if it is false, then no program can realize it. As a consequence, for instance, one can devise trivial realizers for the first two Peano axioms: if we chose λ-calculus as our *pca*, for the first one one simply takes the term $\lambda x.\lambda y.\lambda z.z$, for the second one one can take any term.

Finally, whereas theorem $\mathbf{1}$) and its variants $\mathbf{1}'), \mathbf{1}''$) all express the soundness of **HA** with respect to the realizability semantics, the converse result (i.e. completeness) is false: there exists many well-known cases of realizable sentences which are not intuitionistically derivable. A long list of remarkable examples can be provided as a corollary of a simple result stating that, for every formula A, either it or its negation is realizable. In particular, since the excluded middle $A \vee \neg A$ is not realizable, its negation $\neg(A \vee \neg A)$ is realizable (but not derivable). Again, as a consequence of the Halting problem one can show that the formula $\forall n(Halt(n) \vee \neg Halt(n))$, where $Halt(n)$ is a predicate expressing that the program coded by the integer n halts, is not realizable. Thus, its negation is realizable, but not derivable. These examples show that realizability, as we defined, is fundamentally incompatible with classical logic. The extension of realizability to a classical frame demands indeed for the introduction of several new ingredients (see [Kri09]).

We can produce other interesting examples of incompleteness which are indeed compatible with classical logic by applying Gödel's theorems: the latter allows to find Π^0_2 formulae which are not derivable in **HA**; in particular, to devise recursive functions whose totality cannot be proved in **HA**. Now, as a consequence of **2′**, the recursive function itself, as coded in a suitable *pca*, can be seen as a realizer of the Π^0_2 sentence expressing its totality.

Remark that, since a realizer is a constructive object in all respects, this means that in a sense we have constructive realizations of all (true) totality statements. By the way, due to the incompleteness theorems, to quote Kleene, we have no means to "read the requisite information" from these programs. In a word, we have the codes but we don't know how to decode them. This idea will be indeed at the basis of chapter (6) and discussed in chapter (7).

To sum up, the main features of the realizability interpretation are essentially three: the *polymorphism* of realizers, the *proof-irrelevance* of atomic sentences and the *incompleteness* with respect to realizable sentences.

3.2.2 Tait-Girard reducibility

In the literature on type theory the expression "Tait-Girard reducibility" refers to a family of techniques for proving normalization arguments which originates in a paper by Tait ([Tai67]) on the strong normalization of Gödel's System T and was successively developed and extended to higher order type theories by Girard in his thesis ([Gir72]). Several variants and further developments of this technique can be found in the literature (for instance, Krivine's technique of saturated sets [Kri93] or Mitchell's [Mit86]).

Furthermore, Prawitz's proof-theoretical validity in [Pra71a], so as Martin-Löf's computability in [ML70a] arose as extensions of the reducibility technique to natural deduction for, respectively, intuitionistic higher order logic and the intuitionistic theory of (iterated) inductive definitions. We briefly discuss below the (quite relevant) differences between these two related techniques.

The technique of reducibility predicates shares many ideas and features with the realizability interpretation: in particular [Tai75], soon after the publication of Girard's ideas, elaborated an untyped version of Girard's reducibility and showed that the normalization proof for System F could be restated in the form of a realizability argument; [Gal95] discusses in detail the relationship between realizability and reducibility. In a sense, it might be said that Tait-Girard reducibility is an application of the idea of realizability to type theory in order to prove normalization.

In particular, this technique can be seen as a semantics of programs which associates types with sets of programs behaving in a certain way; the definition of the behavior of programs strongly resembles the realizability interpretation. Moreover, whereas [Tai67] and [Gir72] are based on a typed frame (i.e. types are associated with sets of typed programs), in [Tai75] and later [Mit86] and [Gal90] reducibility is defined in an untyped frame (i.e. types are associated with sets of pure λ-terms). The latter will be the approach followed in the brief sketch below.

The reducibility of simple types We present here an untyped version of the reducibility technique which is essentially based on Tait's paper [Tai75] and on [Gir11] and [Gal90]. We limit ourself to the case of finite types (covering, by the forgetful translation, the case of intuitionistic first order logic), as this will be enough for a brief comparison with the notion of proof-theoretic validity presented in the preceding section. In the next section we discuss the second order case.

The first intuition for a normalization proof for type theory is to develop an argument by induction over the size of terms. The main difficulty arises in the case of a redex $(\lambda x.M)N$, since the reduced term $M[N/x]$ might have size strictly bigger than the former. That's why one looks indeed for an argument by induction over the types, with a subinduction over the reduction relation for each type (similarly to the definition of validity).

Tait's idea in [Tai67] is to define, for each type σ, a set of terms of type σ which are called *computable*, and which have the property of being strongly normalizing. In [Tai75] this idea is restated in an untyped frame: with each simple type σ, he associates a set Red_σ of untyped λ-terms by induction as follows:

- if σ is a variable, then $Red_\sigma = \mathcal{SN}$, the set of strongly normalizing λ-terms;
- if $\sigma = \tau \to \rho$, then $Red_\sigma = Red_\tau \to Red_\rho$, i.e. the set of λ-terms M such that, for all $N \in Red_\tau$, $MN \in Red_\sigma$.

[Tai75] explicitly states this definition in a realizability style: he defines indeed a realizability relation between terms and types given by $M \circledR \sigma$ if and only if $M \in Red_\sigma$.

> When $a \in \overline{A}$, we say that *a realizes A*. This is closely related to Kleene's 1945 recursive realizability interpretation, except that, instead of coding functions by their Gödel numbers, we use the corresponding term of $\mathcal{C}$ [i.e. pure λ-calculus]. [Tai75]

In [Gir72] Girard proves three crucial properties of this definition:

Lemma 3.2.1. *Reducibilitiy satisfies the following properties:*

(R1) Every reducible term is in $\mathcal{SN}$*;*

(R2) $M \in Red_\sigma$ and $M \to_\beta M'$ implies $M' \in Red_\sigma$ *;*

(R3) If M is *simple* and for all M' such that $M \to_1 M'$, $M' \in Red_\sigma$, then $M \in Red_\sigma$.

Proof. We argue by induction over the types. We just discuss the case of the implication $\sigma \to \tau$, since the variable case is obvious.

(**R1**) By **R3** applied to σ, the variable x is in Red_σ; if $M \in Red_{\sigma\to\tau}$, then $Mx \in Red_\tau$, hence, by **R1** for τ, $Mx \in \mathcal{SN}$, which implies that $M \in \mathcal{SN}$.

(**R2**) If $M \in Red_{\sigma\to\tau}$ and $M \to_\beta M'$, and if $N \in Red_\sigma$, then $MN \in Red_\tau$ and, since $MN \to_\beta M'N$, by **R2** $M'N \in Red_\tau$. Hence, $M' \in Red_{\sigma\to\tau}$.

(**R3**) Let $N \in Red_\sigma$; by **R1**, $N \in \mathcal{SN}$; by induction on the $|N|$, the supremum of the lengths all reduction sequences of N, one shows that $MN \in Red_\tau$ (and, a fortiori, by **R3**, that $M \in Red_{\sigma\to\tau}$; indeed, the immediate reducts of MN are of the form $M'N$, where $M \to_1 M'$, or MN', where $N \to_1 N'$ (here we use the fact that M is simple). Now, $M'N \in Red_\tau$ by hypothesis, whereas $MN' \in Red_\tau$ by induction hypothesis, since $|N'| < |N|$.

□

Property **R1** states that, for all type σ, Red_σ is a subset of $\mathcal{SN}$; property **R2** states that reducibilities are closed under β-reduction; property **R3** is the least intuitive: a *simple* term is a term which does not begin with a λ[5]; then the property states that reducibilities are closed under immediate anti-reduction.

One of Girard's ideas in [Gir72] was to use properties **R1** $-$ **3** in order to define an abstract notion of *reducibility candidate*, fundamental for the second order case (see section (4.1)):

Definition 3.2.1 (Reducibility candidate)**.** *A* reducibility candidate $\mathcal{C}$ *is a set of* λ*-terms satisfying* **R1** $-$ **3**.

[5]Remark that those terms, from a natural deduction perspective, essentially correspond to derivations which are not in canonical form.

Hence lemma (3.2.1) can be restated as saying that, for all σ, Red_σ is a reducibility candidate.

In order to prove strong normalization for all simply types terms, it remains then to show that, for every term M of type σ, M realizes σ (this is called indeed the *realizability theorem* in [Tai75]).

The idea is now to proceed by induction over the reduction of terms: the problematic case of an application MN now becomes easy: from $M \in Red_{\sigma\to\tau}$ and $N \in Red_\sigma$ one immediately gets $MN \in Red_\tau$, and hence $M \in \mathcal{SN}$. More delicate is the case of λ-abstraction: we have to show that, if $M \in Red_\tau$, then $\lambda x.M \in Red_{\sigma\to\tau}$, i.e. for every $N \in Red_\sigma$, $(\lambda x.M)N \in Red_\tau$. Remark that, by **R3**, it is enough to show that $M[N/x] \in Red_\tau$. To achieve this we need to strengthen the induction hypothesis: we will show by induction indeed that, if $M \in Red_\tau$, $x \in FV(M)$ is declared of type σ and $N \in Red_\sigma$, then $M[N/x] \in Red_\tau$.

Remark that this strengthened version essentially corresponds to what is proven, for natural deduction, by lemma (3.1.2).

To prove the final result, now, we need a lemma:

Lemma 3.2.2. *If $\Gamma, (x : \sigma) \vdash M : \tau$ is derivable in simple type theory and, for all $N \in Red_\sigma$, $M[N/x] \in Red_\tau$, then $\lambda x.M \in Red_{\sigma\to\tau}$.*

Proof. We have to show that $(\lambda x.M)N \in Red_\tau$. Remark that M is reducible (and hence strongly normalizing), since $My \in Red_\tau$. We argue then by induction on $|M| + |N|$; since $(\lambda x.M)N$ is a simple term, by **R3** it suffices to show the result for its immediate reducts; these are of the form $(\lambda x.M')N$, for $M \to_1 M'$ and hence $|M'| < |M|$, or $(\lambda x.M)N'$, with $N \to_1 N'$, $|N'| < |N|$, both reducible by induction hypothesis, or $M[N/x]$, reducible by hypothesis.

□

We can now state the main theorem, corresponding to lemma (3.1.2).

Theorem 3.2.1. *Let $(x_1 : \tau_1), \dots, (x_n : \tau_n) \vdash M : \sigma$ be derivable in simple type theorty. Then, for every choice of $N_1 \in Red_{\tau_1}, \dots, N_n \in Red_{\tau_n}$, $M[N_1/x_1, \dots, N_n/x_n] \in Red_\sigma$.*

Proof. Let $\overline{M}$ be $M[N_1/x_1, \dots, N_n/x_n]$. We argue by induction on the term M:

i. If M is a variable, then the result is immediate;

ii. If $M = \lambda x.M'$, then $\sigma = \tau \to \rho$ and, by induction hypothesis, $(x_1 : \tau_1), \dots, (x_n : \tau_n), (x : \tau) \vdash M' : \rho$ is derivable and for all $N \in Red_\tau$, $M[N/x] \in Red_\rho$. Hence, by lemma (3.2.2), $\lambda x.M \in Red_{\tau\to\rho}$;

iii. If $M = M'M''$, then, by induction hypothesis, $(x_1 : \tau_1), \dots, (x_n : \tau_n) \vdash M' : \tau \to \rho$ and $(x_1 : \tau_1), \dots, (x_n : \tau_n) \vdash M'' : \tau$ are derivable and $\overline{M}' \in Red_{\tau\to\rho}$ and $\overline{M}'' \in Red_\tau$ for certain types τ, ρ (this is easily proved by induction on the typing derivation), and the result immediately follows by the definition of reducibility.

□

We can thus finally state the "realizability theorem" as a corollary:

Corollary 3.2.1. *If $\vdash M : \sigma$ is derivable in simple type theory, then M is a realizer of σ (and hence strongly normalizing).*

Proof. By **R3** variables belong to Red_τ, for all τ, hence, if $FV(M) = \{x_1, \dots, x_n\}$, $M[x_1/x_1 \dots x_n/x_n] = M \in Red_\sigma$.

□

Reducibility and validity We won't enter here into the complex debate over the differences between a normalization argument and a semantic proof of validity (which constitutes for instance the subject of [SH06]), but we limit ourselves to highlight some important differences between the two historically related approaches of Tait-Girard reducibility and proof-theoretic validity.

Tait-Girard reducibility was a key ingredient for the development of the notions of proof-theoretic validity (Prawitz explicitly refers to Tait's and Girard's work in [Pra71a]). It is indeed possible to extend the reducibility definition into a definition of proof-theoretic validity, based on elimination rules rather than on introduction rules (as the clause defining $Red_{\sigma\to\tau}$ imposes). Such a definition is just sketched in [Pra71a] and discussed in [SH06].

Definition 3.2.2 (Validity based on elimination rules)**.** *Let d be a natural deduction derivation of conclusion A. d is valid if either*

(V′1) *A is atomic and d is strongly normalizable;*

(V′2) *$A = B \Rightarrow C$ and for every valid derivation d' of conclusion B, the derivation*

$$\dfrac{\begin{matrix}\vdots\, d \\ B \Rightarrow C\end{matrix} \quad \begin{matrix}\vdots\, d' \\ B\end{matrix}}{C}\ (\Rightarrow E) \tag{3.2.5}$$

is valid.

As a consequence of lemma (3.2.2), lemma (3.1.2) is derivable for this eliminative version of validity. The definition above looks much simpler than the one based on introduction rules, for at least two reasons: first, no distinction is made between canonical and non canonical derivations, since no reference is made to introduction rules; second, the complex and counterintuitive clause (**V3**) is absent, and is indeed recovered by the analogue property (**R3**), which is a consequence of the definition by means of lemma (3.2.1). In particular, the definition is a pure induction over formulae, in contrast with Prawitz's validity, which is defined by an iterated induction over formulae and over the reduction relation (such an iterated induction is then recovered in the proof of theorem (3.2.1).

The lack of a canonical/non canonical distinction makes the definitions easier, but has the consequence that the very idea of proof-theoretic semantics is lost: the validity of a derivation is not defined in terms of an ideal form that the derivation must have or must achieve, through reduction; it is indeed defined in terms of the potential behavior of the derivation in fixed contexts, i.e. following the paradigm of realizability.

In the preceding section, we identified the epistemological content of the definition of validity with the fact that valid derivations can be transformed into canonical ones, which are in a sense valid by definition. In the case above, no derivation is taken as valid by definition in virtue of its internal form. Rather, derivations are valid in virtue of the properties of their behavior (their interaction with other derivations in the case of a non-atomic conclusion). The normalization argument achieves then the following result: *if a λ-term can be assigned the type σ (equivalently, if a derivation has been constructed following the introduction* and *elimination rules of intuitionistic logic), then it will behave in a well-specified way; in particular, the term itself will be strongly normalizing and its interaction with other (well-behaving) terms will preserve validity.*

Though the reducibility approach does not consider the internal structure of terms, it inherits the main features of realizability semantics: first, the untyped and polymorphic frame, given by the fact that we associates types with sets of pure λ-terms; second, the proof-irrelevance of atomic types: all atomic types are assigned the set $\mathcal{SN}$ of strongly normalizing terms, which means that the behavior of their realizers is not decomposed, as it is the case for non atomic types. Finally, the incompleteness of simple type theory: from a closed term M being a realizer of a certain type σ, it does not follow that M can be given type σ in simple type theory (see next section).

3.2.3 Untyped semantics

By the expression "untyped semantics" it will be meant an approach to the interpretation of proofs which reflects the perspectives coming from realizability and reducibility interpretations. It must be said that, whereas the philosophical and epistemological development of proof-theoretic semantics is the object of a quite large literature, the literature on realizability and Tait-Girard reducibility arguments is quite confined to mathematics and computer science departments, with some few exceptions (notably Girard's many philosophical comments and intuitions, Joinet's work on the philosophy of computability - [Joi07, Joi09, Joi11] - and some other works like [NPS14]).

There are at least two active research programs in the logic panorama that explicitly pursue the idea of an untyped interpretation of proofs: one is Krivine's program ([Kri09, Kri11, Kri12, Kri], which aims at reconstructing the untyped programs (or machines) which lay beyond proofs of classical logic and mathematics, by extending the *Curry-Howard correspondence* to classical logic and set theories. The other one is Girard's *geometry of interaction* program ([Gir89c, Gir89a, Gir90a, Gir95, Gir06, Gir10, Gir13]), which aims at reconstructing untyped proofs from a purely geometrical perspective, provided by operator algebras ([Gir89a, Gir10]) and unification algebras ([Gir13]).

In trying to highlight the main features of these approaches we aim indeed at helping confronting two different traditions sharing the same origins (intuitionism and constructivist mathematics), but divided by a cultural and technical gap evolved through time (the aim of reconstructing the history of this bifurcation in the development of logic clearly exceeds the aims of this short presentation).

Untyped proofs and intuitionism Kleene's recursive realizability is often presented as a formalization of the intuitionistic (or BHK) explanation of the logical constants, thought Kleene explicitly rejected this connection (see [Kle45]). As it is argued in [Sun83], a major difference between the two approaches regards the different way in which the notion of proof (or, more generally, the notion of "construction") is considered: whereas in stating his conditions in [Hey56] Heyting was describing constructions, methods as informal (mental?) entities, not themselves subject to mathematical treatment, Kleene's interpretation of proofs pursues an explicit mathematical formalization of the (allegedly) intuitionistic notion of construction. At best, as Kleene himself writes in [Kle45], his interpretation can be seen as an explanation of intuitionistic constructions within classical mathematics.

One of the main features of the untyped approach is indeed the interpretation of proofs as elements of certain algebraic structures (*pca*): the informal notion of proof (or construction) is thus replaced by a well-defined mathematical notion, investigated with purely mathematical tools (λ-calculus, category theory and even operator algebra, in the case of geometry of interaction). This feature appears in sharp contrast with the intuitionistic credo that constructions are purely mental entities.

Historically, the reception of the intuitionist interpretation was strongly influenced by Kreisel's formalizations in [Kre60, Kre65]: there his aim was "*to set up a formal system, called "abstract theory of constructions" for the basic notions mentioned above, in terms of which the formal rules of Heyting's predicate calculus can be interpreted*". In particular Kreisel's reconstruction incorporated a recursive predicate formalizing the relation

$$\textit{the construction } c \textit{ proves the formula } A \tag{3.2.6}$$

so that the BHK clauses themselves could at the end be regarded as logical formulae. As Sundholm observes:

> The difference in aims between the early views of Heyting-Kolmogoroff and Kreisel now becomes clear. Heyting-Kolmogoroff do not give a reduction to any other theory, but try to explain what a proposition is, how it should be understood. For Kreisel, on the other hand, the aim was to formalize the properties of the "abstract constructions" in a theory and reduce the theory of logic to that. Kreisel is thus closer to [...] Gödel's *Dialectica* and the realizability interpretations. [Sun83]

As it is well-known, in his attempt Kreisel was led to slightly modify the clauses for implication and universal quantification, by adding the request of an effective "verification" that the construction actually does what it is supposed to: for instance, a proof of an implication $A \Rightarrow B$ is a construction c which assigns to each proof d of A a proof $c(d)$ of B, along with a verification that c satisfies this condition. As we discuss below this further clause is especially problematic in the case of Π^0_1 formulae. It is quite significative that Troelstra's 1968 presentation of intuitionism ([Tro69]) incorporates Kreisel's modifications and explicitly refers to his formalization.

Kreisel's theory of constructions was not the only attempt at formalizing the intuitionistic notion of construction: we can mention for instance Gödel's *Dialectica* interpretation [Gö58], Scott's theory of constructive validity [Sco68] and Martin-Löf's intuitionistic type theory [ML84]. In the latter, in particular, the identification of proofs with certain mathematical objects is a consequence of the *formulae-as-types* paradigm (that we discuss in the next paragraph). Martin-Löf draws a distinction between proofs as constructions, in the sense of mathematical objects, and proofs as derivations in tree-like form:

> To distinguish between proofs of judgements (usually in tree-like form) and proofs of propositions (here identified with elements, thus to the left of $\in$) we reserve the word *construction* for the latter and use it when confusion might occur. [ML84]

In the context of proof-theoretic semantics the question about the nature of constructions is debated: [Sun98] reports the skepticism by Prawitz about the legitimacy of the mathematical notion of construction; in particular, in [Pra12], Prawitz considers two opposite alternatives: the first one, attributed to Martin-Löf and Sundholm, takes proofs as construction in the mathematical sense:

> For instance, a "proof" of an implication $A \Rightarrow B$ is simply a function that applied to proofs of A yields a proof of B, and the "proofs" of A and B may again be just functions, which may make one doubt that the notion of proof is really an epistemic one. [Pra12]

In particular, Martin-Löf, in [ML98], claims that proofs are not to be considered as epistemic notions, but rather as mathematical "proof objects" which may enter in the stipulation of the proof-conditions for the logical constants (he explicitly draws a connection with Kleene's realizability).

The second alternative, inspired by the verificationist thesis, takes proofs as chain of inferences, where an inference is conceived as a piece of linguistic practice; this alternative can be found for instance in Dummett's treatment of derivations in sequent calculus or natural deduction as a formalization of linguistic practice, with no peculiar interest in their inner mathematical structure.

It is the opinion of the author that this divergence about the legitimacy of a purely mathematical treatment of constructions constitutes one of the main obstacles which keep the philosophical tradition of proof-theoretic semantics far from the the tradition of Kleene and Kreisel (and in particular from the most recent advances in the mathematical interpretation of proofs, as Krivine's *classical realizability* and Girard's *geometry of interaction*).

Russell's typing and *Curry-Howard* typing The principle by which Russell introduced his type discipline in [Rus08] was the following:

(**RUS**) *The range of significance of a propositional function forms a type*

by that he meant that, when considering a propositional function, i.e. a predicate $P(x)$ depending on a variable x, the objects to which the predicate can be applied must belong to a well-defined set. Syntactically, this implies that the terms that can be substituted for the variable x in $P(x)$ must be *of the same type as* the variable x. We can indeed rephrase the principle (**RUS**) by the following syntactic principle: variables occurring (free or bound) in predicates (and in proofs) must be typed. Variables are then written with a type index as x^σ, a propositional function $P(x^\sigma)$ being a function from σ to a certain family of propositions.

As it is widely known, the reason that led Russell to introduce the type discipline was that, by admitting unrestricted substitutions for the variables occurring in his propositional functions, it was possible to construct pathological propositions, leading to the well-known antinomies.

The *Curry-Howard* typing discipline can be obtained (as it is explained for instance in [Coq90]) from Russell's discipline by simply adding the principle below:

(**PasT**) *Propositions should be identified with types*

Indeed, principle (**PasT**), along with principle (**RUS**), yields the consequences that a *proof* of a proposition is an *object* of a certain type, and that a propositional function $P(x^\sigma)$ is a function from σ to a certain family of types. The identification of proofs with the objects of a type leads then to the interpretation of the former as programs (an example of the forgetful translation can be found clearly stated in [How80]). Moreover, if a propositional function is a function from a type to a family of types, it follows that a proof of a universal proposition $\forall x^\sigma A$ is indeed a typed program of type $\sigma \to \mathcal{A}$, where $\mathcal{A}$ is a family of types over the elements of σ (this idea is made explicit in Martin-Löf type theory [ML84], see also subsection (4.3.2)).

Now, admitting unrestricted substitutions for the variables occurring in the propositional function $P(x^\sigma)$ amounts, from the Curry-Howard perspective, to admitting unrestricted *inputs* for a program of type $\sigma \to \tau$ (for a certain type or family of types τ). This means that Russell's typing discipline is turned into a discipline for the interaction (styled "socialization" in [Joi11]) between programs: it forbids indeed to apply programs to certain other programs. Typically, if a program M has type $\sigma \to \tau$, it cannot have itself as an input.

This is indeed the paradigm which underlies the distinction between *pure* and *typed* programming languages; in the case of λ-calculus, the most studied and significative for type theory, one has an underlying space of programs, with no restriction as to possible interactions: any program can be applied to any other one. The rule-less society of untyped programs reveals itself indeed quite wild, since unrestricted interactions between programs give rise to pathological cases of non terminating computations. In a word, types discipline the contexts in which a program can be put.

On the contrary, once programs are typed, i.e. once the socialization is regulated, pathological cases are expelled from society; the typical example, again, is the one of *autoapplication*: if a variable x in a program is declared of type $\sigma \to \tau$, then it cannot be applied to a program of type $\sigma \to \tau$, and in particular it cannot be applied to itself. This is tantamount to say that the λ-term $\lambda x.(x)x$ is not typable in simple type theory (in chapter (6) we discuss these properties of typing from the abstract viewpoint of unification theory).

Remark that a peculiar feature of simple types is that the typing is in a sense rigid: terms are called *monomorphic*, which means that they have a unique type. On the contrary, if a variable can have, at the same time, different types, then one can find a way to correctly type an auto-application (this *polymorphism* is indeed the central feature of higher-order type theories, see chapters (5) and (6)).

Now, as the strong normalization theorem of the preceding section shows, the rigid discipline of simply types provides the following properties:

1. *All programs of variable type are strongly normalizing*;

2. *If M and N are strongly normalizing programs and MN is well-typed, then MN is strongly normalizing.*

Remark that the two properties above strongly resemble the realizability-reducibility clauses; in particular, property 2. implies that all programs are strongly normalizing: if M does not have variable type, then it has type $\sigma \to \tau$, and for $N = x$, MN is well-typed and is strongly normalizing, hence M is too. In a sense, the realizability-reducibility clauses express the norms of "socialization" of simple type theory. [Joi11] discusses in detail the "social" features that can be imported in logic from the experience on typed λ-calculi:

> Ce qu'on pourra appeler "la bonne socialité dynamique des processus" prévaut donc encore dans le fragment typé correspondant au système de déduction naturelle concerné, et le typage doit être vu non simplement comme découpant un sous-ensemble des termes (des programmes), mais au delà comme découpant un *fragment de la dynamique* (un sous-ensemble des Õvaluations, une sous-dynamique) [...] [Joi11]

Krivine's program pushes farther this reflection, since it aims at reconstructing the behavioral content of mathematical theorems:

> Nous avons écrit ce programme à partir d'une preuve d'une certaine formule Θ. Nous sommes confrontés à ce que j'appelle le *problème de la spécification*, qui est, sans doute, le problème le plus difficile mais aussi le plus fascinant posé par la correspondance de Curry-Howard:
>
> *Étant donné un théorème Θ (de la théorie des ensembles avec choix dépendant), quel est le comportement commun à tous les programmes obtenus à partir des preuves de Θ?*

In the following paragraph we try to sketch some of the features of an interpretation of proofs and their validity based on this untyped (wild) vs typed (civilized) paradigm.

Untyped proofs and rules The two basic principles shared by the realizability and the reducibility interpretation of derivations and programs are the following:

U1 *Proofs are interpreted as untyped programs*;

U2 *Rules are interpreted by clauses disciplining the "socialization" of proofs.*

Principle **U1** is indeed the starting point of Kleene's 1945 realizability interpretation, and principle **U2** is a consequence of the application of the realizability-reducibility clauses to the **PasT** condition.

An immediate consequence of **U1** is that the internal structure of proofs is not taken in consideration by interpretations of this form; as a limit case, a proof of an atomic (true) formula is interpreted by an *arbitrary* (strongly normalizing) program (this is the principle that we called of *proof-irrelevance*). This blindness to the internal structure of proofs implies that an untyped interpretation must assign a quite different role to rules with respect to proof-theoretic semantics. The latter indeed assigned a role to rules which can be called *constitutive*[6]: there is simply no

[6]The *constitutive/regulative* distinction traces back to Kant, and was more recently retrieved by Searle ([Sea69]. We take here Searle's definition: a rule is constitutive if the existence of the practice it disciplines depends on the acceptance of the rule itself. It is regulative if it disciplines an activity which might exists independently from the acceptance of that rule.

notion of proof without a definition of introduction and elimination rules (or left and right rules in the case of sequent calculus).

On the contrary, in the untyped interpretation, so as in the related notion of eliminative validity (see above), no mention is made to the rules of which a proof is made; in particular, there is no space for a *last rule condition*, i.e. for a distinction between canonical and non canonical proofs.

In the next chapter the question of the retrieval of a *last rule condition* within a reducibility interpretation will be briefly discussed on the basis of a *completeness* theorem for Π^1-reducibility.

A conception of the role of rules must indeed be developed in accordance with principle **U2**: indeed, as it results from the strong normalization theorem, the rules (intended as typing rules) internalize patterns of behavior, as they are described by means of the realizability-reducibility clauses. As Joinet writes,

> [...] chaque (type de) règle est moins une règle d'*inference* (règle de transition des énoncés vers les énoncés) qu'une règle d'*interaction*, règle déterminant une forme particulière d'interaction avec le cotexte. [Joi11]

In particular, the principle of implicit definitions, i.e. that the meaning of the logical constants is implicitly defined by (some of) the rules of logic has to be replaced by what we might call a principle of *behavioral definition*, stating that the meaning of the logical constant is (explicitly) defined by clauses describing the behavior of proofs of formulae containing such a constant as its principal operator in fixed contexts.

Since untyped programs live in an independent space of computations, rules, following this behavioral principle, assume a *regulative* role with respect to the "socialization" of programs. Interpreting logical rules as regulative, rather than constitutive, amounts to viewing a "logic" as a set of restrictions imposed on programs to discipline their interaction. For instance, the "logic" of simple types (corresponding, through Curry-Howard, to intuitionistic propositional - and, forgetfully, first-order - logic) is the one in which the interaction of a program with itself is forbidden ("incestuous", one might be tempted to say).

A "logic", in this sense, induces a demarcation between "good" programs, i.e. the typed ones, and "bad" programs, the untyped ones. Indeed, typed programs are exactly those that are the image, under the forgetful translation, of actual derivations in sequent calculus or natural deduction (this is what is indeedasserted by the faithfulness theorem (2.3.1)).

In chapter (5) we provide a stronger result, that we call $\mathbf{\Pi}^1$-completeness, which states that a sequent calculus derivation d of $\vdash A$ can be recovered from a term M which is a realizer of $\forall\overline{\alpha}A^{\mathbb{F}}$, i.e. such that $M \in Red_{\forall\overline{\alpha}A^{\mathbb{F}}}$, where $\forall\overline{\alpha}A^{\mathbb{F}}$ indicates the second order universal closure of $A^{\mathbb{F}}$. Remark that, whereas faithfulness can be easily extended to second order systems, completeness cannot, as a consequence of the incompleteness theorems (see below).

Meaning as use: second interpretation These remarks suggest a second way of interpreting the *meaning as use* paradigm (with respect to the one recalled in subsection (3.1.1)). In proof-theoretical semantics this motto is usually applied to sentences, and thus becomes "the meaning of a sentence is given by its use", where the expression "use" refers to the practice of justifying assertions and drawing consequences from them, by means of arguments and proofs. From the viewpoint of untyped semantics, the motto can be applied (as suggested by Girard in [Gir11]) to proofs rather than to sentences: "the meaning of a proof is given by its use", where the expression "use" refers to the act of "applying" the proof, seen as a program, a method (in the BHK sense), to other proofs-programs, to obtain a new proof-program as a result. In particular, the type of a program is what disciplines the possible (admitted) uses one can make of it. This second interpretation of Wittgenstein's motto can be stated then in the following form: *the meaning of a (typed) program lies in its socializations.*

Remark, furthermore, that the characteristic *polymorphism* of this approach (i.e. the fact that a program may have several types) implies that the untyped program in itself has not enough information as to uniquely determine its use. In particular, polymorphism implies a form of *polysemy*: a program can have several meanings. We recall again Kleene's 1945 remark:

> A realization number by itself of course conveys no information; but given the form of statement of which it is a realization, we shall be able in the light of our definition to read from it the requisite information. [Kle45]

Indeed, internally, such a program could be a λ-term, a numerical code, an inhabitant whatsoever of some *pca*. As Girard puts it,

> The difference between "pure" and typed objects, this is the difference between things as they are and things as they should be. [Gir11]

Finally, the epistemological role played by cut-elimination in the interpretation differs in some important respects between proof-theoretic semantics and untyped semantics: in the first case normalization is a means of reducing arbitrary proofs to proofs having a given (canonical) internal structure; in the second case, the normalization procedure is itself a fundamental component of the behavioral characterization of meanings. In a word, we might say that, whereas in the first case cut-elimination confers meanings to non canonical proofs, in the second case cut-elimination is itself part of the meaning: a logical constant is explained indeed in terms of how proofs in which such a constant occurs behave with respect to cut-elimination; the latter constitutes indeed the general arena where socialization occurs, i.e. where the interdictions which constitute behavioral meanings take place.

Incompleteness A common feature of the untyped approaches is the fact that (intuitionistic) derivability is incomplete with respect to them. In particular, whereas from every derivation in natural deduction or sequent calculus it is possible to extract a realizer (*soundness*), it is not true that from every realizer one can reconstruct a derivation. To recall again Kleene's remark (pag. 74), a realizer may lack relevant information to retrieve the underlying derivation.

A first source of incompleteness, as we have already remarked, is due to the trivial interpretation of atomic propositions and types. In the case of realizability, proof-irrelevance is inherited by all *Harrop formulae*. *Harrop formulae* are defined inductively as follows:

- every atomic true formula is a Harrop formula;
- if A, B are Harrop formulae, then $A \wedge B$, $A \vee B$ are Harrop formulae;
- if A is a formula and B is a Harrop formula, then $A \Rightarrow B$ is a Harrop formulae;
- if A is a Harrop formula, then $\forall x A, \exists x A$ are Harrop formulas.

These are sometimes called *self-realizable* formulae ([Cro04]) since they have trivial realizers. We can look for their trivial realizers by using reducibility: indeed, simple types are the forgetful image of Harrop formulae. We can show then that every simple type contains trivial realizer: a simple type σ can be written as

$$\sigma_1 \to \cdots \to \sigma_k \to \alpha_0 \tag{3.2.7}$$

where the types σ_i are, in turn, of the form $\sigma_{i,1} \to \cdots \to \sigma_{i,k_i} \to \alpha_i$ etc. Now, if M is an arbitrary strong normalizing λ-term, then the term $M' = \lambda x_1 . \dots . \lambda x_k . M$ is a realizer of σ, and by choosing $M = \lambda z.(z)z$, M' will be untypable (since auto-application is a form of socialization which is forbidden by simple types - see above). Then, in particular, M' can be taken as a realizer of A.

In the next chapter we'll show that this phenomenon can be fixed from a second order perspective: in particular, if we replace the type σ with its *universal closure* $\forall\overline{\alpha}\sigma$, then the reducibility candidate technique allows to eliminate trivial realizers. In particular, in chapter (5) we will be able to prove a completeness theorem for $\mathbf{\Pi}^1$-reducibility (5.2.4), which somehow echoes the theorem of Π^1-completeness for second order logic (chapter (2)).

The second source of incompleteness is concerned with formulae and types which are beyond the Π^1 border: typically, a program M which computes a total recursive function f will be a realizer of the Π^0_2 arithmetical formula expressing its totality, and it will be in the reducibility of the type $\mathbf{N} \to \mathbf{N}$. We end this chapter by sketching how the incompleteness of $Red_{\mathbf{N}\to\mathbf{N}}$ can be derived from Gödel's incompleteness theorems:

Theorem 3.2.2 ($\mathbf{\Sigma}^1$-incompleteness)**.** *For every strongly normalizing type system T containing F, there exists a λ-term $M \in Red_{\mathbf{N}\to\mathbf{N}}$ such that $\vdash M : \mathbf{N} \to \mathbf{N}$ is not derivable in T.*

Proof. Since T is strongly normalizing, there exists a total recursive function f which associates with the code of a term typable in T the code of its normal form (and is 0 otherwise). Now, an application of Gödel's second incompleteness theorem (see [GLT89]) shows that for no λ-term M simulating f, $\vdash M : \mathbf{N} \to \mathbf{N}$ can be derived in T. On the other hand, since f is total, once can find a strongly normalizing pure λ-term M_f representing f such, for all strongly normalizing term N of type $\mathbf{N}$, $M_f N$ reduces to a Church integer, i.e to a normal term of type $\mathbf{N}$ (see appendix (C)). Then M is a realizer of $\mathbf{N} \to \mathbf{N}$ which does not have type $\mathbf{N} \to \mathbf{N}$ in T.

□

Chapter 4

Around the second order *Hauptsatz*

The question we address in this chapter is the following: can second order cut-elimination play the same role as first-order cut-elimination within a proof-theoretic theory of validity?

In the first section we first recall Girard's reducibility argument for System F (which implies the second order *Hauptsatz*). The logical complexity of this argument, which is measured by the growing complexity of the set-theoretic comprehension principles exploited, is then contrasted with the existence of an elementary procedure (described in [Sch60, Gir76, KT74]) to replace cuts by means of instances of the comprehension rule. Indeed, this fact seems to indicate that the distinction between cut-free and non cut-free derivation (so as the one between canonical and non canonical derivations), in the second order case, has a purely formal content, as it does not capture structural properties of derivations.

In the second section we discuss the usual objections against impredicative definitions, and show how they are turned, within the proof-theoretic semantic tradition, into objections agains the possibility of a proof-theoretic justification of the rules of second order logic. We argue that the vicious circularity ascribed to the justification of second order logic does not correspond to the circularity at work in the reducibility argument, and we show the substantial harmlessness of these objections from the viewpoint of the untyped interpretation of proofs.

The last section is devoted to characterize the "epistemic" circularity at work in the proof of the second order *Hauptsatz*; in particular, we highlight the potentially catastrophic uses that can be made of this circularity, by reconstructing the faulty normalization argument for Martin-Löf's inconsistent impredicative type theory [ML70b].

4.1 Reducibility and Takeuti's conjecture

4.1.1 Reducibility

The *Hauptsatz* for second order sequent calculus was conjectured in 1953 by Takeuti ([Tak57]). At that time the reducibility technique was not known and the question was attacked, and first resolved, by means of semantical techniques ([Tai68], [Tak67], [Pra68]). The first syntactical proof, in [Gir72], was obtained as a corollary of a strong normalization proof for System F, based on the notion of reducibility candidate we introduced in the preceding section.

A faulty extension Before proceeding to the actual definitions, it is instructive to first consider an intuitive, though wrong, extension of the definition of reducibility given in the preceding chapter. If we follow the pattern of realizability-reducibility clauses, based on elimination rules, it is quite natural to propose an extension to the second order case as follows:

M is a realizer of $\forall\alpha\sigma$ if, for all type τ, M is a realizer of $\sigma[\tau/\alpha]$.

and, in particular, to define the reducibility $Red_{\forall\alpha\sigma}$ as the intersection of all the reducibilities $Red_{\sigma[\tau/\alpha]}$.

Apparently, one can reconstruct a great part of the reducibility argument with the definition above. In particular, a "proof" reducibility enjoys properties $\mathcal{R}\infty - \ni$ as well as a"proof" of strong normalization can be attempted as follows:

False lemma 4.1.1. *Reducibility enjoys properties* **R1** − **3**.

Proof. The argument is by induction over types:

R1) Property **R1** follows immediately from the fact that $Red_{\forall\alpha\sigma} \subseteq Red_\sigma$;

R2) if, for all τ, $M \in Red_{\sigma[\tau/\alpha]}$ and $M \to M'$, then, by induction hypothesis, $M' \in Red_{\sigma[\tau]}$ for all τ, hence $M' \in Red_{\forall\alpha\sigma}$;

R3) let M be simple and, for all M' such that $M \to_1 M'$, $M' \in Red_{\forall\alpha\sigma}$. This implies that, for all type τ, $M' \in Red_{\sigma[\tau]}$ and then, by induction hypothesis, one argues that $M \in Red_{\sigma[\tau]}$ from property **R3** applied to $Red_{\sigma[\tau/\alpha]}$. Finally, one obtains that $M \in Red_{\forall\alpha\sigma}$, since $M \in Red_{\sigma[\tau]}$, for all type τ

□

Strong normalization follows then from the following "lemma":

False lemma 4.1.2. *Let $(x_1 : \sigma_1), \ldots, (x_k : \sigma_k) \vdash M : \sigma$ be derivable in F, with $FV(\sigma) = \{\alpha_1, \ldots, \alpha_n\}$. Then, for all choice of types $\tau_1, \ldots, \tau_n$ and of terms $N_1, \ldots, N_k$, with, for $1 \le i \le k$, $N_i \in Red_{\sigma_i[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n]}$, $M[N_1/x_1, \ldots, N_k/x_k] \in Red_{\sigma[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n]}$.*

Proof. We just consider the case of the rules $(\forall I)$ and $(\forall E)$, since the other rules should be treated similarly to the simply typed case.

$(\forall I)$

$$\frac{\Gamma \vdash M : \sigma \quad \alpha \text{ bindable in } \Gamma}{\Gamma \vdash M : \forall\beta\sigma} \tag{4.1.1}$$

By induction hypothesis, we know that, for all choices of types $\tau_1, \ldots, \tau_n, \tau$ and terms $N_1, \ldots, N_k$, with $N_i \in Red_{\sigma_i[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n,\tau/\beta]}$, one has $M[N_1/x_1, \ldots, N_n/x_k] \in Red_{\sigma[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n,\tau/\beta]}$, but this means exactly that $M[N_1/x_1, \ldots, N_k/x_k] \in Red_{\forall\beta\sigma[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n]}$.

$(\forall E)$

$$\frac{\Gamma \vdash M : \forall\beta\sigma}{\Gamma \vdash M : \sigma[\tau/\beta]} \tag{4.1.2}$$

By induction hypothesis, we know that $M[N_1/x_1, \ldots, N_k/x_k] \in Red_{\sigma[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n,\rho/\beta]}$ for all types $\tau_1, \ldots, \tau_n, \rho$ and terms $N_1, \ldots, N_k$, with $N_i \in Red_{\sigma_i[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n,\rho/\beta]}$, hence in particular $M[N_1/x_1, \ldots, N_k/x_k] \in Red_{\sigma[\tau_1/\alpha_1,\ldots,\tau_n/\alpha_n,\tau/\beta]}$.

□

The problem with the "proofs" above is that, as the reader should have already remarked, they rely on an induction over a non well-founded order: the reducibility of the type $\forall\alpha\sigma$ is defined in terms of the reducibility of *all* types τ (i.e. included $\forall\alpha\sigma$)!

The definition of reducibility in simple type theory is by induction over types. In particular, given a type σ, Red_σ is defined as a function of the Red_τ, for all subtype τ of σ. This is tantamount to saying that the validity for derivations of a formula A in first-order logic is defined in terms of the validity for derivations of the subformulae of A.

At second order the subformula order between formula is lost: since all $A[B/X]$ should be considered morally as "subformulae" of $\forall XA$, it follows that the order is not well-founded. As a consequence, the definition of reducibility $Red_{\forall\alpha\sigma}$ as a function of all the $Red_{\sigma[\tau/\alpha]}$, for every type τ, is by induction on a non-well-founded order over types.

We can detect this circularity in the "proof" of lemma (4.1.1): when proving the property **R3**. For instance, if $\sigma = \alpha$ and $\tau = \forall\alpha\alpha$, one assumes **R3** for τ as induction hypothesis to prove **R3** for $\forall\alpha\alpha$, i.e. τ.

Remark that the proof of lemma (4.1.2), on the contrary, uses an induction over typing derivations, and thus it does not collapse. However, the simply typed part of the proof (which is the same as for theorem (3.2.1)), reposes over the fact that reducibility enjoys properties $\mathbf{R1}-\mathbf{3}$, i.e. over the wrong proof of lemma (4.1.1).

The circularity at work in these wrong arguments is of the same kind as the one at work in Frege's wrong proof in the *Grundgesetze* (see page 7). As we are going to see, the correct way to fix the definition of reducibility involves a subtle trick: in a sense the circularity of second order quantification is not eliminated, though it is reframed in a way that does not make the argument viciously circular.

The theorem It was exactly to cope with the problem just presented that the notion of reducibility candidate was created. Indeed, Girard's solution amounts to replace, in the definition of reducibility for a universal type, the quantification over all types with a quantification over all reducibility candidates. In a word, the dependency of the reducibility over all reducibilities is replaced by a dependency of reducibility over a (large) family of sets.

In order to reframe reducibility in this new setting, we need a notion of reducibility parametrized by a set of reducibility candidates.

Definition 4.1.1 (parametric reducibility). *Let σ be a type and, for each free variable α_i occurring in σ, let $\mathcal{C}_i$ be a reducibility candidate.* Parametric reducibility $Red_\sigma[\dots\mathcal{C}_i/\alpha_i\dots]$ *is a property over λ-terms defined by induction over σ as follows:*

i. if $\sigma = \alpha_i$, then $Red_\sigma[\dots\mathcal{C}_i/\alpha_i\dots](M)$ iff $M \in \mathcal{C}_i$;

ii. if $\sigma = \tau \to \rho$, then $Red_\sigma[\dots\mathcal{C}_i/\alpha_i\dots](M)$ iff for all N such that $Red_\tau[\dots\mathcal{C}_i/\alpha_i\dots](N)$, $Red_\rho[\dots\mathcal{C}_i/\alpha_i\dots](MN)$ holds;

iii. if $\sigma = \forall\alpha\sigma'$, then $Red_\sigma[\dots\mathcal{C}_i/\alpha_i\dots](M)$ holds iff for every reducibility candidate $\mathcal{C}$, $Red_{\sigma'}[\dots\mathcal{C}_i/\alpha_i\dots,\mathcal{C}/\alpha](M)$ holds.

Now, since the parametric reducibility of $\forall\alpha\sigma$ is defined in terms of the parametric reducibility of σ, we can now prove an analog of lemma (4.1.1) by a truly well-founded induction over types:

Lemma 4.1.1. *Parametric reducibility enjoys properties* $\mathbf{R1}-\mathbf{3}$.

Proof. The argument is by induction over types. We limit ourselves to the case of universal types, since the other ones require just a reformulation of lemma (3.2.2).

(R1) let $Red_{\forall\alpha\sigma}[\ldots\mathcal{C}_i/\alpha_i\ldots](M)$ hold, then, $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,\mathcal{SN}/\alpha]$ holds and, by induction hypothesis, $M\in\mathcal{SN}$;

(R2) let $Red_{\forall\alpha\sigma}[\ldots\mathcal{C}_i/\alpha_i\ldots](M)$ hold and $M\to M'$; then, for all candidate $\mathcal{C}$, $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,\mathcal{C}/\alpha](M)$ holds, and by i.h., $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,\mathcal{C}/\alpha](M')$ holds, from which one concludes that $Red_{\forall\alpha\sigma}[\ldots\mathcal{C}_i/\alpha_i\ldots](M')$ holds;

(R3) let M be simple and, for all M' such that $M\to_1 M'$, $Red_{\forall\alpha\sigma}[\ldots\mathcal{C}_i/\alpha_i\ldots](M')$ hold; then, again, for all candidate $\mathcal{C}$, $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,\mathcal{C}/\alpha](M')$ holds for all the M' and, by i.h., $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,\mathcal{C}/\alpha](M)$ holds. One concludes that $Red_{\forall\alpha\sigma}[\ldots\mathcal{C}_i/\alpha_i\ldots](M)$ holds.

□

A consequence of this lemma is that, if σ is a closed type, then its reducibility Red_σ is not parametric and, with the help of an instance of the comprehension axiom, it can be shown that it is a reducibility candidate. In the literature this trick goes under the name of *Girard's trick*: it states the fact that the family of reducibility candidates is closed under intersections indexed by the family itself (for a more detailed mathematical digression on this topic see [Gal90]).

A direct application of Girard's trick gives the following lemma:

Lemma 4.1.2 (substitution lemma)**.** *For all types σ,τ, for all candidates $\mathcal{C}_i$ and for every term M, $Red_{\sigma[\tau/\alpha]}[\ldots\mathcal{C}_i/\alpha_i\ldots](M)$ holds if and only if $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,Red_\tau[\ldots\mathcal{C}_i/\alpha_i\ldots]/\alpha](M)$ holds.*

Proof. The lemma is established by induction over σ. At each stage we use an instance of the comprehension schema to establish that $Red_\tau[\ldots\mathcal{C}_i\ldots]$ is a set.

variable If $\sigma=\alpha_i$, then $Red_{\sigma[\tau/\alpha]}[\ldots\mathcal{C}_i/\alpha_i\ldots](M)$ holds iff $M\in\mathcal{C}_i$ iff $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,Red_\tau[\ldots\mathcal{C}_i/\alpha_i\ldots]/\alpha](M)$ holds; if $\sigma=\alpha$, then $Red_{\sigma[\tau/\alpha]}[\ldots\mathcal{C}_i/\alpha_i\ldots](M)$ holds iff $Red_\tau[\ldots\mathcal{C}_i\ldots](M)$ holds iff $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,Red_\tau[\ldots\mathcal{C}_i/\alpha_i\ldots]/\alpha](M)$ holds.

implication If $\sigma=\sigma_1\to\sigma_2$, then $Red_{\sigma[\tau/\alpha]}[\ldots\mathcal{C}_i/\alpha_i\ldots](M)$ iff for all N such that $Red_{\sigma_1[\tau/\alpha]}[\ldots\mathcal{C}_i/\alpha_i\ldots](N)$ holds, $Red_{\sigma_2[\tau/\alpha]}[\ldots\mathcal{C}_i/\alpha_i\ldots](MN)$ holds. The thesis then immediately follows by induction hypothesis.

universal If $\sigma=\forall\beta\sigma'$, then $Red_{\sigma[\tau/\alpha]}[\ldots\mathcal{C}_i/\alpha_i\ldots](M)$ holds iff for every candidate $\mathcal{D}$, $Red_{\sigma'[\tau/\alpha]}[\ldots\mathcal{C}_i/\alpha_i\ldots,\mathcal{D}/\beta](M)$ holds iff for every candidate $\mathcal{D}$, $Red_{\sigma'}[\ldots\mathcal{C}_i/\alpha_i\ldots,Red_\tau[\ldots\mathcal{C}_i/\alpha_i\ldots,\mathcal{D}/\beta]/\alpha,\mathcal{D}/\beta](M)$ holds (by induction hypothesis), iff $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,Red_\tau[\ldots\mathcal{C}_i/\alpha_i\ldots]/\alpha](M)$ holds.

□

Before proceeding to the theorem, remark that the analogue of lemma (3.2.2) is indeed true by definition: if M is a term and for all candidate $\mathcal{D}$, $Red_\sigma[\ldots\mathcal{C}_i/\alpha_i\ldots,\mathcal{D}/\beta](M)$ holds, then $Red_{\forall\beta\sigma}[\ldots\mathcal{C}_i/\alpha_i\ldots](M)$ holds.

The theorem we prove below is a second order generalization of theorem (3.2.1) of the following form: given a term M of type σ, we show that it is reducible for any choice of reducibility candidates for the free variables of σ and of reducible terms for its free variables.

Theorem 4.1.1. *Let $(x_1:\tau_1),\ldots,(x_k:\tau_k)\vdash M:\sigma$ be derivable in F and let $FV(\sigma)=\{\alpha_1,\ldots,\alpha_n\}$. Then, for every choice of reducibility candidates $\mathcal{C}_1,\ldots,\mathcal{C}_n$ and of terms $N_1,\ldots,N_k$, such that, for $1\le i\le k$, $1\le j\le n$, $Red_{\tau_i}[\ldots\mathcal{C}_j/\alpha_j\ldots](N_i)$ holds, $Red_\sigma[\ldots\mathcal{C}_j/\alpha_j\ldots](M[\ldots N_i/x_i\ldots])$ holds.*

Proof. We just consider the case of the rules ($\forall I$) and ($\forall E$):

($\forall I$)
$$\frac{\Gamma \vdash M : \sigma \quad \alpha \text{ bindable in } \Gamma}{\Gamma \vdash M : \forall\beta\sigma} \tag{4.1.3}$$

By induction hypothesis, we know that, for all reducibility candidate $\mathcal{D}$, $Red_\sigma[...\mathcal{C}_j/\alpha_j \ldots, \mathcal{D}/\beta](M[\ldots N_i/x_i \ldots])$ holds. Hence it follows then that $Red_{\forall\beta\sigma}[...\mathcal{C}_j/\alpha_j \ldots](M[\ldots N_i/x_i \ldots])$ holds.

($\forall E$)
$$\frac{\Gamma \vdash M : \forall\beta\sigma}{\Gamma \vdash M : \sigma[\tau/\beta]} \tag{4.1.4}$$

By induction hypothesis, we know that $Red_{\forall\beta\sigma}[...\mathcal{C}_j \ldots](M[\ldots N_i/x_i \ldots])$ holds, i.e. that, for all reducibility candidate $\mathcal{D}$, $Red_\sigma[...\mathcal{C}_j/\alpha_j \ldots, \mathcal{D}/\beta](M[\ldots N_i/x_i \ldots])$ holds. In particular (comprehension axiom!) this holds for $\mathcal{D} = Red_\tau[\ldots \mathcal{C}_j/\alpha_j \ldots]$, and by the substitution lemma (4.1.2) this implies that $Red_{\sigma[\tau/\beta]}[\ldots \mathcal{C}_j/\alpha_j \ldots](M[\ldots N_i/x_i \ldots])$ holds.

□

As usual, we obtain as an immediate corollary:

Corollary 4.1.1. *If M has type σ in System F, then M is strongly normalizing.*

The difference between the false theorem (4.1.2) and the correct one above is rather subtile. To the consequences of this subtile difference is indeed devoted this entire chapter.

Second order realizability Kleene provided an extension of his interpretation to intuitionistic analysis in [Kle59], by introducing the theory of countable functionals (see also [Kre59]). [TVD88] contains the "standard" extension of realizability to second order arithmetics: the idea is to consider formulae of second order logic parametrized by subsets $S \subseteq \mathbb{N}$ and to add atomic sentences of the form $\underline{n} \in S$, where n is an integer and S is a set. This requires to introduce two new clauses: one for the new formulae of the form $\underline{n} \in S$ and one for universal quantification. We follow the presentation in [VO08] and we let p represent a "pairing function" over codes (i.e. an injective function from pairs of integers to integers):

v. e realizes $\underline{n} \in s$ if $p(e, n) \in S$;

vi. e realizes $\forall X A$ if, for all $S \subset \mathbb{N}$, e realizes $A[S/X]$.

There are some evident analogies between the clauses above for (parametric) realizability and the definition of (parametric) reducibility: by introducing parametrization one is able indeed to adopt a clause with a (non circular) quantification over sets.

More explicitly, [Tai75] contains a realizability interpretation of second order intuitionistic arithmetics built over Girard's reducibility candidates technique. In particular, he defines a forgetful translation from a derivation d of conclusion A to a λ-term d^- (which is actually $\mathbb{F}(d)$) of type $\mathbb{F}(A)$; then, he introduces, for each formula B a constant P_B and defines realizability parametrized to such constants. Clause *vi.* is replaced by a clause of the form

M realizes $\forall\alpha\sigma$ if, for every B and every constant P_B, M realizes $\sigma[P_B/\alpha]$

Finally he shows that the set of realizers so defined are reducibility candidates in the sense of Girard and proceeds in analogy with the proof of theorem (4.1.1).

4.1.2 Takeuti's conjecture: an empty shell?

From the strong normalization theorem for System F one can derive a positive answer to Takeuti's conjecture, i.e. to the *Hauptsatz* for the second order sequent calculus (both intuitionistic and classical, see [Gir72]); similarly, a strong normalization argument for natural deduction formulations of (intuitionistic or classical) second order logic, with the rules below can be deduced

$$\frac{A}{\forall XA}\ (\forall I) \qquad\qquad \frac{\forall XA}{A[P/X]}\ (\forall E) \tag{4.1.5}$$

where, in the rule $(\forall I)$ one requires that X does not occur free in any open assumption. Proof-theoretical definitions of validity, in the style of Prawitz's [Pra71a], will be discussed in the next section.

Before discussing, from an epistemological point of view, the significance of these results, it is convenient to make a couple of remarks on cut-elimination in a second order framework.

First of all, second order rules satisfy Prawitz's inversion principle; in a natural deduction formalism the argument is as follows: a derivation as below

$$\begin{array}{c} \vdots\ d \\ \dfrac{\dfrac{A}{\forall XA}}{A[P/X]} \\ \vdots \end{array} \tag{4.1.6}$$

is reduced into the derivation below

$$\begin{array}{c} \vdots\ d\{P/X\} \\ A[P/X] \\ \vdots \end{array} \tag{4.1.7}$$

Once more, whereas the inversion principle can be established in a local and elementary way, the complete proof of cut-elimination is quite complex and demands for very strong logical principles (the comprehension principles used in lemma (4.1.2)).

In a word, the inversion principle is fundamentally incapable of capturing the logical complexity intrinsic to the normalization argument. In particular, since the derivation d is replaced by the derivation $d\{P/X\}$, in which all occurrences of X are replaced by the predicate P, no concrete inductive measure can be imposed upon derivations in order to turn the *local* normalization (i.e. invertibility) into a *global* normalization argument.

Poor and absorbing formulae Despite the fact that the *Hauptsatz* for second order logic is a logically complex result, for a certain class of second order formulae (called *poor* formulae in [Gir76]), cut-elimination can be proved in an elementary way: as it is remarked in [Sch60], if d is a derivation of a formula $F_0 \Rightarrow A$, where A is arbitrary and F_0 is the formula $\forall x\forall X(X(x) \Rightarrow X(x))$, then d can be transformed into a cut-free proof d' in a primitive recursive way. Formulae like F_0 are called *absorbing* in [Gir76], since in a sense they absorb the cuts. Absorbing formulae are dual to poor formulae. Let d have the following form

$$
\begin{array}{c}
\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Gamma \vdash A, \Delta & \Gamma', A \vdash \Delta'\end{array}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ (cut)\\
\vdots\\
\dfrac{F_0 \vdash A}{\vdash F_0 \Rightarrow A}\ (\Rightarrow R)
\end{array}
\tag{4.1.8}
$$

Then the cut can be "absorbed" by means of a $(\Rightarrow L)$ rule plus a $(\forall L)$ rule and a successive contraction:

$$
\begin{array}{c}
\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Gamma \vdash A, \Delta & \Gamma', A \vdash \Delta'\end{array}}{\Gamma, \Gamma', A \Rightarrow A, \vdash \Delta, \Delta'}\ (\Rightarrow L)}{\Gamma, \Gamma', F_0 \vdash \Delta, \Delta'}\ (\forall L)_{X,x}\\
\vdots\\
\dfrac{\dfrac{F_0, F_0 \vdash A}{F_0 \vdash A}\ (C)}{\vdash F_0 \Rightarrow A}\ (\Rightarrow R)
\end{array}
\tag{4.1.9}
$$

In a word, the comprehension on A "swallows" the cut on A.

It is interesting to reformulate the transformation above in type theory: suppose to have a non normal term $\lambda z.M$ of type $\phi_0 \to \rho$, where ϕ_0 is the type $\forall \alpha(\alpha \to \alpha)$, containing a redex $(\lambda x.P)Q$ and whose typing derivation has the form below:

$$
\begin{array}{c}
\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma', (x : \sigma) \vdash P : \tau\end{array}}{\Gamma' \vdash \lambda x.P : \sigma \to \tau} \quad \begin{array}{c}\vdots\\ \Gamma' \vdash Q : \sigma\end{array}}{\Gamma' \vdash (\lambda x.P)Q : \tau}\\
\vdots\\
\dfrac{\Gamma, (x : \phi_0) \vdash M : \rho}{\Gamma \vdash \lambda z.M : \phi_0 \to \rho}
\end{array}
\tag{4.1.10}
$$

Then we can transform the typing derivation of $\Gamma' \vdash Q : \sigma$ as follows (by using proposition (2.1.1) *i.*):

$$
\dfrac{\dfrac{\Gamma', (z : \phi_0) \vdash z : \phi_0}{\Gamma', (z : \phi_0) \vdash z : \sigma \to \sigma} \quad \begin{array}{c}\vdots\\ \Gamma', (z : \phi_0) \vdash Q : \sigma\end{array}}{\Gamma', (z : \phi_0) \vdash (z)Q : \sigma}
\tag{4.1.11}
$$

Remark that, since $(z)Q$ is simple, it cannot introduce new redexes in P as it is substituted for x. We apply then lemma (2.1.1) and we finally obtain the derivation below:

$$
\begin{array}{c}
\vdots\\
\Gamma', (z : \phi_0) \vdash P[(z)Q/x] : \tau\\
\vdots\\
\dfrac{\Gamma, (z : \phi_0) \vdash M' : \rho}{\Gamma \vdash \lambda z.M' : \phi_0 \to \rho}
\end{array}
\tag{4.1.12}
$$

In definitive, by repeatedly applying this procedure we can recursively transform $\lambda z.M$ into a normal term $\lambda z.M^*$ having the same type $\phi_0 \to \rho$.

[KT74] it is shown that Dedekind's predicate $N(x)$ is absorbing: let d be the following derivation:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\; d_1\\ \Gamma, N(x), C \vdash A\end{array} \quad \begin{array}{c}\vdots\; d_2\\ \Gamma \vdash C\end{array}}{\Gamma, \Gamma', N(x) \vdash A}\,(cut)}{\Gamma, \Gamma' \vdash \forall x(N(x) \Rightarrow A)} \tag{4.1.13}$$

Let then y be a variable that does not occur in C. We transform the cut as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\; d_1\\ \Gamma, N(x), C \vdash A\end{array} \quad \begin{array}{c}\vdots\; d_2\\ \Gamma' \vdash C\end{array}}{\Gamma, \Gamma', N(x), C \Rightarrow C \vdash A}\,(\Rightarrow L) \quad \dfrac{\dfrac{C \vdash C}{\vdash C \Rightarrow C}\,(\Rightarrow L)}{\vdash \forall z(C \Rightarrow C)}\,(\forall R)_z}{\Gamma, \Gamma', N(x), \forall z(C \Rightarrow C) \Rightarrow (C \Rightarrow C) \vdash A}\,(\Rightarrow L)}{\Gamma, \Gamma', N(x), N(x) \vdash A}\,(\forall L)_{\lambda y.C/X}}{\Gamma, \Gamma', N(x) \vdash A}\,(C)}{\Gamma, \Gamma' \vdash \forall x(N(x) \Rightarrow A)} \tag{4.1.14}$$

Now it is Dedekind's predicate which swallows the cut. This fact has a surprising consequence: by recursively applying the transformation above we can obtain an elementary proof of the *Hauptsatz* for Π^0_1 and Π^0_2 arithmetical formulae[1]. The translation of the argument above in type theory shows that, if $\lambda z.M$ is a term of type $\mathbf{N} \to \mathbf{N}$ in system F, then $\lambda z.M$ can be recusively transformed into a normal term $\lambda z.M^*$ having the same type.

Since this elementary argument can clearly be formalized in $\mathbf{HA}^2$, one can prove in $\mathbf{HA}^2$ the *Hauptsatz* for all second order derivations of Π^0_2 formulae.

However, this does not contradict Gödel's second incompleteness theorem nor the fundamental distinction between elementary and logically complex concepts, since the "trivial" cut-elimination that we just presented does not imply consistency: the usual argument to derive consistency from the *Hauptsatz* proceed from the hypothesis of the existence of a derivation d of the falsity to the absurd conclusion that d can be reduced to a cut-free derivation; remark then that, since falsity $\perp$ is not a poor formula, the elementary cut-elimination argument cannot be applied.

Indeed, if there were a derivation d of the absurd, then all that the argument above shows is that d can be transformed into a cut-free derivation d' of $\mathbb{N}(x) \vdash \perp$; if we now cut d' with a derivation of $\mathbb{N}(t)$ for a suitable term t (for instance $t = \underline{0}$), we obtain a new derivation d'' of the absurd, *which might not be cut-free.*

Thus, the method above recursively transforms an arbitrary derivation of a Π^0_2 into a cut-free second order one, in which the cuts are hidden behind occurrences of the $(\forall L)$ rule, i.e. of instances of the comprehension schema. Equivalently, it recursively transforms an arbitrary term of type $\mathbf{N} \to \mathbf{N}$ into a normal one. In particular, since the cut-free derivation obtained violates the subformula (as the cuts are transformed into witnesses for the universal quantifier), it is not possible to apply the usual arguments for deducing semantical properties from cut-elimination (in particular consistency).

[1] [Gir76] provides a systematic investigation of this phenomenon, by giving syntactic and semantic criteria to recognize poor and absorbing formulae. In particular it contains a result named "poverty theorem", which states that if A is a second order formula which is 1-consistent with PA (with induction restricted to Π^0_2 formulae), then all formulae equivalent to A are poor. In particular, for instance, all Gödel's sentences are poor (a result already established in [KT74]).

One could conclude then that the epistemological value of Takeuti's conjecture is, after all, quite limited: on the one hand, the proof of the *Hauptsatz* for second order logic must employ set-theoretical comprehension principles in order to justify the comprehension rules within the system (i.e. must rely on a "pragmatically" or "epistemically" circular argument, see below subsection (4.3.1)); on the other hand, one can directly exploit comprehension rules *within* the system, and eliminate cuts in an elementary, trivial, way!

The meaning of cut-free and canonical derivations in second order logic In the last pages we presented a method which recursively transforms second order derivations into cut-free ones but which does not imply consistency, as the usual *Hauptsatz*. This fact prompts some challenges on the epistemological value of the distinction between cut-free derivations and derivations with cut in second order logic.

In first-order logic cut-free derivations play a significant role in virtue of their structural properties, connected with the subformula property. In second order logic, where the subformula property fails as a consequence of the comprehension rule ($\forall L$), the structural properties of cut-free derivations can hardly be distinguished from those of arbitrary derivations: as the transformation above show, a cut in a derivation can always be replaced by an occurrence of a ($\Rightarrow L$) rule followed by a comprehension rule ($\forall L$), which "swallows" the cut.

Similar remarks can be made for the distinction between canonical and non canonical derivations. As we recalled in the last chapter (subsection (3.1.3)) this distinction is connected with an epistemological distinction between derivations that can be taken as immediately valid, or valid in virtue of their form, and derivations whose validity requires for a reductive argument (a *justification*, in Prawitz's terminology).

Now, though the distinction canonical/non canonical can be formally extended to the second order frame, it seems hard to maintain that the epistemological value of this distinction is preserved in this setting. In particular, the argument of the preceding paragraph shows that Dummett's *fundamental assumption* (subsection (3.1.3)), i.e that every derivation can be reduced in canonical form, can be proved in a purely formal way and does not provide the characterization of a structurally peculiar class of proofs.

4.2 The vicious circle principle

4.2.1 The debate over impredicative definitions

The debate over *non-predicative* ([Rus06b]) or *impredicative* definitions arose in response to the discovery of the paradoxes between the end of the 19th century and the beginning of the 20th century. [Rus06b] is the first reference where the notion is presented and tentatively defined: there Russell calls "non-predicative" the propositional functions which do not define a class and takes the function "x is not a member of x" as an example.

The first argument As confirmed by Poincaré's pitiless remarks in [Poi06], Russell's [Rus06b] shed no light on the source of the paradoxes, and provided no useful demarcation between predicative and impredicative definitions. Poincaré proposed instead an analysis based on what is usually called the "vicious-circle principle" **VCP**. [Poi06] does not contain an explicit definition of the principle, but some examples and some remarks:

> [...] leur définitions sont non prédicatives et présentent cette sorte de cercle vicieux caché que j'ai signalé plus haut: les définitions non prédicatives ne peuvent pas être substituées au terme défini. [Poi06]

Russell's response to Poincaré, in [Rus06a], contained indeed the first explicit formulation of the **VCP**:

> I recognize [...] that the clue to the paradoxes is to be found in the vicious circle suggestion; I recognize further this element of truth in M. Poincaré's objection to totality, that whatever in any way concerns *all* or *any* or *some* of a class must not be itself one of the members of a class. [...]
>
> In M. Peano's language, the principle I want to advocate may be stated: "Whatever involves an apparent variable must not be among the possible values of that variable". [Rus06a]

Remark that, both Poincaré and Russell, expressed the idea of a vicious circle by means of a substitutional criterion. In particular, Russell's formulation of the **VCP** is strictly connected with the substitutional principle **RUS** in [Rus08], giving rise to his formulation of type theory.

In addition to the pragmatical justification of the **VCP**, given by the fact that the principle blocks the construction of the antinomies, an explanation of the principle can be found in [Poi06]: Poincaré's argument is based on a conception of what logic is, and in what logic differs from mathematics. His idea was that a purely logical proof is one which, once the expressions involved in it are replaced by their definitions, can be transformed into a series of tautological propositions. Mathematical proofs, on the contrary, do not reduce to tautologies but to propositions the acknowledgement of whose truth requires the appeal to intuition. For instance, he insists that, if one has proved an equality of the form $X = Y$, then he must be able to reduce the equality into the tautological form $X = X$. The proof itself should provide indeed the substitutions required.

> Mais si l'on remplace successivement les diverses expressions qui y figurent par leur définition at si l'on poursuit cette opération aussi loin qu'on le peut, il ne restera plus á la fin que des identités, de sorte que tout se réduira à une immense tautologie. La Logique reste donc stérile, à moins d'être fécondée par l'intuition. [Poi06]

It is on the basis of this conception of logic that Poincaré argues for the rejection of impredicative definitions. Indeed, he claims that, if an impredicatively defined concept occurs in the proof, the replacement of it with its definition might fail to produce a series of tautologies.

> Dans ces conditions, *la Logistique n'est plus stérile, elle engendre l'antinomie.* [Poi06]

In the next section we try to reconstruct Poincaré's informal argument in the context of a natural deduction frame.

The second argument Among the most well-known defenses of impredicative definitions stands Ramsey's [Ram31]: there he claims that such definitions surely imply some form of circularity, but that this is harmless:

> But, it will be objected, surely in this there is a vicious circle; you cannot include $F(x) = \forall\phi(f(\phi(z), x))$ among the ϕ's, for it presupposes the totality of the ϕ's. This is not, however, really a vicious circle. The proposition $F(a)$ is certainly the logical product of the propositions $f(\phi(z), x)$, but to express it like this is [...] is merely to describe it in a certain way, by reference to a totality of which it may be itself a member, just as we can refer to a man as the tallest in a group, thus identifying him by means of a totality of which he is himself a member without there being any vicious circle. [Ram31]

In a word, Ramsey claims that there is nothing circular in defining an object by reference to a tolatily to which that object belongs, if that totality is already well-defined. The application of this argument, though, presupposes the platonistic thesis that the totality of sets is a well-defined one, independently of the definitions that one can provide of one of its elements.

Carnap's ([Car83]) contains an analysis of Ramsey's argument which is very crude on this point:

> Although this happy result is certainly tempting, I think we should not let ourselves be seduced by it into accepting Ramsey's basic premise; viz., that the totality of properties already exists before their characterization by definition. Such a conception, I believe, is not far removed from a belief in a platonic realm of ideas which exist in themselves. [...]
>
> It seems to me that, by analogy, we should call Ramsey's mathematics "theological mathematics", for when he speaks of the totality of properties he elevates himself above the actually knowable and definable and in certain respects reasons from the standpoint of an infinite mind which is not bound by the wretched necessity of building every structure step by step. [Car83]

Gödel's [G4̈4] contains a very lucid analysis of this contraposition:

> [...] it seems that the vicious circle principle in its first form applies only if the entities involved are constructed by ourselves. In this case there must clearly exist a definition (namely the description of the construction) which does not refer to a totality to which the object defined belongs, because the construction of a thing can certainly not be based on a totality of things to which the thing to be constructed itself belongs. If, however, it is a question of objects that exists independently of our constructions, there is nothing in the least absurd in the existence of totalities containing members which can be described (i.e. uniquely characterized) only by reference to this totality. [...]
>
> So it seems that the vicious circle principle in its first form applies only if one takes the constructivistic (or nominalistic) standpoint towards the objects of logic and mathematics [...] [G4̈4]

In particular, from Carnap's objection and, more clearly, from Gödel's analysis, we can retrieve a second argument, in addition to Poincaré's one, for the **VCP**: if one adopts the "constructivistic" view that definitions create, or constitute the objects defined, rather than simply describing pre-existing objects, then circular definitions should be avoided, since a construction cannot depend on a totality to which the construction itself belongs.

In the next subsection similar arguments for the rejection of second order logic here presented will be described in the context of proof-theoretic semantics; the difference is that, rather than rejecting impredicative definitions, such argument will be addressed to the rejection of implicit definitions by means of rules for impredicative quantification.

4.2.2 Proof-theoretic semantics

We reconstruct Prawitz's definition of validity for second order logic (which follows an adaptation of the reducibility candidate technique). Next we discuss some of the arguments against impredicative quantification that can be found in the literature on proof-theoretic semantics. We argue that some of those arguments seem to presuppose the faulty stipulation of the meaning of a universally quantified formula $\forall XA$ as a function of the meaning of all its substitution instances $A[P/X]$ (similarly to the faulty extension of reducibility presented above).

Second order validity The first works on proof-theoretic semantics were quite neutral on second-order quantification: Prawitz's program of "general proof-theory" was originally conceived to include second order logic. In particular [Pra71a] contains an extension of the definition of proof-theoretic validity to second order logic.

First observe that a naïve extension will run into problems similar to the naïve extension of reducibility discussed above: let us add a condition **V4** as follows:

(V4) *$A = \forall XB$ and d is canonical, i.e. of the form*

$$\dfrac{\begin{array}{c}\vdots\, d'\\ A\end{array}}{\forall XA}\,(\forall I) \tag{4.2.1}$$

and for all formula B, the derivation $d'\{B/X\}$

$$\begin{array}{c}\vdots\, d'\{B/X\}\\ A[B/X]\end{array} \tag{4.2.2}$$

is valid.

The reader will then easily recognize the vicious circularity of this definition: for instance, the validity of a derivation of $\forall XX$ is defined in terms of the validity of derivations of *every* formula!

To overcome this difficulty Prawitz adapts Girard's technique of reducibility candidates to the validity framework: he defines the notion of a *regular set* of derivation in analogy with Girard's reducibility candidates and introduces a new definition of validity for derivations *relative* to an assignment $\mathcal{N}$ of regular sets to the predicate variables occurring in the derivations. The clauses **V1-V3** of the definition of validity in section (3.1.3) are then replaced by clauses **V1′-V3′** when the parametrization occurs, and two further clauses are added: a clause **V0** for atomic formulae and a clause **V4′** for universal quantification:

Definition 4.2.1 (Relative validity for the $\forall \Rightarrow$-fragment of intuitionistic logic)**.** *Let d be a natural deduction derivation of conclusion A. Let $\mathcal{N}$ be an assignment of regular sets to the predicate variables occurring in d. d is valid relative to $\mathcal{N}$ if either:*

(V0) *$A = P(t_1, \dots t_n)$ and $d \in N$, when $\mathcal{N}$ assigns the set N to the predicate $P(x_1, \dots, x_n)$;*

(V1′) *$A = B \Rightarrow C$ and d is canonical, i.e. of the form*

$$\dfrac{\begin{array}{c}[B]\\ \vdots\\ C\end{array}}{B \Rightarrow C}\,(\Rightarrow I) \tag{4.2.3}$$

and for every derivation d' valid relative to $\mathcal{N}$, of conclusion B, the derivation

$$\begin{array}{c}\vdots\, d'\\ B\\ \vdots\\ C\end{array} \tag{4.2.4}$$

is valid relative to $\mathcal{N}$;

(V2′) *d is not canonical and normal;*

(V3′) *d is not canonical and not normal, and for every derivation d' such that d reduces to d' in one step, d' is valid relative to $\mathcal{N}$;*

(V4′) *$A = \forall XB$ and d is canonical, i.e. of the form*

$$\dfrac{\begin{array}{c}\vdots\, d'\\ B\end{array}}{\forall XB}\,(\forall I) \tag{4.2.5}$$

and for every predicate P and every regular set N, the derivation

$$\begin{array}{c} \vdots\, d'\{P/X\} \\ B[P/X] \end{array} \tag{4.2.6}$$

is valid relative to $\mathcal{N}'$, where $\mathcal{N}'$ differs from $\mathcal{N}$ only in that it assigns the set N to the occurrences of the variable X.

Now, by an argument that closely follows the proof of theorem (4.1.1), Prawitz shows that, if d is valid relatively to an assignment $\mathcal{N}$ of regular sets, then d is strongly normalizing. In particular, one has to use a variant of the substitution lemma (4.1.2) to show that, if d is a derivation of a formula of the form $A[B/X]$, then d is valid relative to an assignment $\mathcal{N}$ if and only if it is valid (as a formula of conclusion A) relative to the assignment $\mathcal{N}'$, which differs from $\mathcal{N}$ only in that it assigns to X the *set* of derivation of B which are valid relative to $\mathcal{N}$.

Intuitionistic type theory Martin-Löf's original type theory ([ML70b]), discussed in the next section, was a fully-fledged impredicative theory, admitting a type of all types. After the discovery of Girard's paradox, however (see section (4.3.2), chapter (6) and appendix (B)), his research turned into a predicativist direction, based on a well-founded hierarchy of universes (see [ML75, ML84]).

Martin-Löf's later versions of intuitionistic type theory are based on a distinction between *sets* and *categories*: a set is defined by specifying how its *canonical* elements are formed, and when two *non canonical* elements are equal; a category, instead, is defined by specifying "what an object of the category is and when two such objects are equal" [ML84]. In particular,

> A category need not be a set, since we can grasp what it means to be an object of a given category even without exhaustive rules for forming its objects. For instance, we now grasp what a set is and when two sets are equal, so we have defined the category of sets [...] but it is not a set. [ML84]

A second major difference between the two is that, whereas it is possible, in intuitionistic type theory, to quantify over the elements of a set, it is not possible to quantify over the objects of a category: thus, for instance, it is not possible to quantify over the category of sets, and thus to introduce second order quantification. Martin-Löf claims that it is the ambiguïty about these two notions, when defining types, which leads to the paradoxes of Russell's and his original type theory.

> What about the word *type* in the logical sense given to it by Russell with his ramified (resp. simple) type theory? Is type synonymous with category or with set? In some cases with one, it seems, and in other cases with the other. And it is this confusion of two different concepts which has led to the impredicativity of the simple theory of types. When a type is defined as the range of significance of a propositional function, so that types are what the quantifiers range over, then it seems that a type is the same thing as a set. On the other hand, when one speaks about the simple types of propositions, properties of individuals, relations between individuals etc., it seems as if types and categories are the same. The important difference between the ramified types [...] and the simple types [...] is precisely that the ramified types are (or can be understood as) sets, so that it makes sense to quantify over them, whereas the simple types are mere categories. [ML84]

Martin-Löf's argument seems to presuppose the claim that one cannot provide a non circular definition of what a canonical object of the category of proposition: if such a definition were given, then the category would be a set, and thus one would be entitled to define *new* canonical elements of those set by quantifying over all of them.

Retrieval of the first argument On the same lines of Martin-Löf's rejection are Dummett's views on impredicativity and the **VCP**: in several places (for instance in [Dum91a, Dum06]) he explicitly endorses the **VCP** and in [Dum91b] he rejects the possibility of circular dependencies in the description of the meaning of the logical constant (more below).

Impredicative quantification is a rather controversial theme in the literature on proof-theoretic semantics. Still [Pra71a] contains the remark that Girard's trick is a "wonderful example of impredicativity". Nevertheless, it is possible to find, within the context and the vocabulary of this tradition, arguments against impredicativity which are very similar to the ones described above.

For instance, Sundholm criticizes the meaning explanation of second order quantification, by making reference to Poincaré's argument on the eliminability of the defined notions:

> A meaning-explanation for the second order quantifier begins by stipulating that $(\forall X \in Prop)A$ has to be a proposition under the assumption that A is a propositional function from $Prop$ to $Prop$, that is, that $A \in Prop$, provided $X \in Prop$. One then has to explain, still under the same assumption, which proposition it is:
>
> $$(\forall X \in Prop)A \textit{ is true if and only if } A[P/X] \textit{ is true, for each proposition } P$$
>
> In the special case of $(\forall X \in Prop)X$ one obtains
>
> $$(\forall X \in Prop)X \textit{ is true} =_{def} P \textit{ is true, for each proposition } P$$
>
> but $(\forall X \in Prop)X$ is (meant to be) a proposition, so it has to be considered on the right-hand side. Accordingly (4.2.2) cannot serve as a definition of what it is for $(\forall X \in Prop)X$ to be true; it does not allow for the elimination, effective or not, of
>
> *...is true*
>
> when applied to the alleged proposition $(\forall X \in Prop)X$. [Sun99]

At the same time, behind Dummett's conception of harmony we can recognize a view on logical deduction as essentially self-explanatory (or "sterile", to recall Poincaré's quotation): we recall below an aforementioned quotation:

> The requirement that this criterion for harmony be satisfied conforms to our fundamental conception of what deductive inference accomplishes. An argument or proof convinces us because we construe it as showing that, given that the premisses hold good according to our ordinary criteria, the conclusion must also hold *according to the criteria we already have for its holding.* [Dum91b]

If we draw some consequences from a concept C that we have previously introduced (according to its defining introduction rules) then it must be possible, by harmony, to draw the same consequences from the concepts employed for the introduction of C; in a word, it should be possible to *eliminate* the concept once we replace it by its definition (the introduction rule). Here the similarity with Poincaré's views on the elimination of defined concepts in a purely logical proofs appears compelling.

> The difference between predicative and impredicative second-order quantification is not about a cautious and a bold assumptions about what mathematical entities exists: it is between an axiomatization which is self-explanatory and one that is not. [Dum91a]

Indeed, in the case of an argument involving second order quantification, from the transformation involved in eliminating a cut between an introduction and an elimination rule, there is no warranty that the "concept" introduced (a second order quantification) be eliminated, since it may occur ("circularly") as the witness of the second order elimination rule, as in the example below:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\, d\\ X\end{array}}{\forall XX}\,(\forall I)}{\forall XX}\,(\forall E) \\ \vdots \tag{4.2.7}$$

which reduces to

$$\begin{array}{c}\vdots\, d\{\forall XX/X\}\\ \forall XX\\ \vdots\end{array} \tag{4.2.8}$$

Retrieval of the second argument The second argument against impredicativity, the Ramsey-Gödel one, can be found in several places: in [Dum06] Dummett explicitly endorses their argument

> Quantification over a domain assumes a prior conception of what belongs to that domain: by trying to specify what belongs to the domain by using quantification over that same domain, we assume as already known what we are attempting to specify. [Dum06]

Elsewhere (for instance, in [Dum91a]) Dummett insists that the debate over impredicative quantification reduces in definitive to the debate on whether mathematical entities are *discovered* or *invented*, thus presupposing that, in the second case, the "constructivist" or "nominalist" one, one should be bound to accept the **VCP**, in accordance with Gödel's remarks.

Moreover, as already mentioned, in [Dum91b] he claims that, if we consider the meaning of the logical constant as fixed by self-justifying rules, then an introduction rule which may involve other logical constants of arbitrary complexity (as the second order existential quantifier) cannot be taken as correctly fixing a meaning. Indeed, a speaker could not understand such a meaning by learning to use the introduction rule, since such a use would presuppose the understanding of all meanings (and in particular of that meaning itself). Dummett considers circular dependencies in the meaning as violating the principle of compositionality of meaning, i.e. the principle that the meaning of a complex sentence must be explained in terms of the meanings of the sentences of which it is composes; he is finally led to require that

> Compositionality demands that the relation of dependence imposes upon the sentences of the language a hierarchical structure deviating only slightly from being a partial order. [Dum91b]

It must be observed that this retrieval of the classical arguments against impredicativity in the proof-theoretic domain seems to rely on a dubious assumption: the way in which Dummett describes the assignment of meaning to second order formulae recalls the faulty extension discussed in the previous section. In particular, he seems to assume that the meaning of a universal formula $\forall XA$ must be described as a function of the meanings of all its substitution instances $A[B/X]$, thus violating his compositionality-as-(quasi)-partial order requirement, so as Poincaré's **VCP**. One can argue in a similar way for Sundholm's stipulation of truth for second order formulae.

As it was shown in the preceding section, this is not the correct way to define reducibility and validity for second order formulae; in definitive, it is not the correct way of assigning meaning (proof-theoretically) to the impredicative universal quantifier. By contrast, the definition of validity, parametrized with respect to an assignment of regular sets, does not violate the **VCP**: it is indeed a truly inductive definition, since the validity of derivations of $\forall XA$ (parametrized by $\mathcal{N}$) is defined as a function of the validity of derivations of A (parametrized by apposite extensions of $\mathcal{N}$).

However, the reducibility argument relies on the substitution lemma (4.1.2) which, in turn, presupposes set-existence principles (i.e. comprehension instances) asserting that to reducibilities there actually correspond appropriate sets. In the next section (subsection (4.3.2)) an especially problematic consequence of this fact will be explored.

4.2.3 Untyped semantics

The tradition that we called "untyped semantics" stands quite on the opposite position in the dispute over impredicativity and higher order reasoning: Girard's work on System F was the starting point of a wide literature on impredicative type theories and their interpretations. In particular, finer analyses of the circularity involved in second order quantification have come from the mathematical interpretation of proofs. Just to name a few, the interpretation of impredicativity by means of *direct limits* in denotational semantics (see [GLT89]), or the dinatural interpretation ([GSS92]) that will be discussed in the next chapter.

It can be claimed that the untyped setting is in several senses more "familiar" with a second order frame; first of all, because polymorphism, a fundamental property of untyped programs, happens to be one of the central aspects of second order type theories: a term M of a universal type $\forall\alpha\sigma$ is indeed usually called *polymorphic* since it can be *extracted*, i.e. seen as a term of type $\sigma[\tau/\alpha]$ for all type τ. In particular variables in System F, contrarily to what happens in Russellian type theories, are not *statically* typed: their type can change if an extraction is performed.

For instance, let us consider the coding of pairs in System F: this is obtained by means of terms of the form $\langle M, N\rangle = \lambda z.(z)MN$, where M and N are, respectively, terms of type σ and τ, for certain types σ and τ. $\langle M, N\rangle$ has type $\sigma \wedge \tau =_{def} \forall\alpha((\sigma \to \tau \to \alpha) \to \alpha)$. Now, pairs are fully characterized by the existence of two *projections* P_1, P_2, i.e two terms satisfying the equations below:

$$P_1\langle M, N\rangle =_\beta (P_1)MN =_\beta M \tag{4.2.9}$$

$$P_2\langle M, N\rangle =_\beta (P_2)MN =_\beta N \tag{4.2.10}$$

From these equations it follows then that the subterm $(z)MN$ can be seen *at the same time* as a term of type σ and as a term of type τ. In a sense, the untyped setting is somehow already built-in second order type theory.

A second reason comes from the proof-irrelevance of atomic types: as we have already seen, the fact of interpreting atomic types as *arbitrary* reducibility candidates is a fundamental ingredient in the formalization of second order reducibility. In particular, theorem (4.1.1) states that a (closed) term M of type σ is in the reducibility $Red_\sigma[\dots\mathcal{C}_i/\alpha_i\dots]$ *for every* choice of reducibility candidates $\mathcal{C}_i$ for the free variables of σ. Since this is exactly the same as saying that M is in the reducibility of $\forall\overline{\alpha}\sigma$, i.e. of the universal closure of σ, this means that the proof of reducibility is carried over *as if all type variables were universally quantified*: this way of treating reducibility (and validity) in a uniformly second order way has some important consequences that are explored in the next chapter, and lead to an explanation of impredicative quantification (based on the notion of parametric polymorphism) which is often ignored in the philosophical debate.

The untyped setting allows to reconsider the arguments against impredicativity discussed so far. Let us start from Poincaré's argument on the non eliminability of impredicative definitions. We already observed that Sundholm's version of Poincaré's argument by his definition of truth seems to trace the wrong definition of validity given by the clause **V4**: the whole interest of the reducibility candidates technique is indeed to avoid this vicious circularity by discharging the impredicativity over the set-theoretical frame in which the system is formalized.

As for the argument based on harmony, it can be said that the *Hauptsatz* expresses the fact that a consequence drawn from a previously introduced concept could have been drawn already from *an instance* of the concepts adopted for the introduction; the only problem is that this can in no way be detected *locally*, since from a local viewpoint no reduction (or actual elimination of concepts) is achieved (we already remarked - subsection (4.1.2) - that the complexity of the second order *Hauptsatz* cannot be detected from the inversion principle). This is rather the conclusion of a logically complex *global* argument. In a word, the second order *Hauptsatz* conveys no information *locally*, but provides a *global* information about termination.

A conception of proofs as determined by their global behavior (irrespective of their internal structure) seems then more akin to accepts this lack of local information, with respect to a conception of proofs as determined by their construction (and their local properties like harmony or the inversion principle).

The polymorphism of the untyped setting involves a different approach to the notion of construction: indeed untyped programs are, by definition, effective methods that can be described by means of an inductive definitions. For instance, the definition of λ-terms is given by a predicatively acceptable induction:

- a variable x is a term;
- if M is a term and x is a variable, $\lambda x.M$ is a variable;
- if M, N are terms, then $(M)N$ is a term.

In particular an untyped program is never defined by reference to the totality of untyped programs. At the same time we remarked that a purely impredicative definition of the behavioral norms would lead into a vicious circle: we cannot define a realizer of $\forall\alpha\sigma$ as a realizer of $\sigma[\tau/\alpha]$ for all α. That's exactly the reason for the introduction of reducibility candidates (see above).

The reducibility clause given by quantification over reducibility candidates escapes Poincaré's vicious circle and blocks the Ramsey-Gödel's argument on constructions depending on the totality of constructions; however, circularity is not eliminated from the frame, but just rearranged in a subtle way: an argument which justifies the fact that a program is a realizer of an impredicative type must employ an impredicative comprehension principle. What did we gain by means of this refinement?

4.3 Kaleidoscope effects

By means of Girard's trick the vicious circularity of the (naïve) definition of reducibility is transferred into the circularity of lemma (4.1.2), in which comprehension *in* logic by means of comprehension *outside* logic, i.e. in instances of the comprehension schema of set theory.

> One arrivers at a strange situation where one no longer knows who interprets who: does reducibility interpret term t, or is it that t would eventually be a way to eninciate its own reducibility? "When you gaze long into the abyss, the abyss also gazes into you". [Gir11]

In this section we investigate this form of circularity which, as we remarked in the last section, does not correspond to Poincaré's notion of vicious circularity.

4.3.1 The *Hauptsatz* seen from within

"Pragmatic" and "epistemic" circularities In [Dum91b] Dummett makes a distinction between two different ways in which an argument for the justification of a logical law can be blamed of circularity: on one side he considers

> [...] the ordinary gross circularity that consists of including the conclusion to be reached among the initial premisses of the argument. [Dum91b]

On the other side, he considers arguments that purport

> to arrive at the conclusion that such-and-such a logical law is valid; and the charge is not that this argument must include among its premisses the statement that the logical law is valid, but only that at least one of the inferential steps in the argument must be taken in accordance with that law. We may call this a "pragmatic" circularity. [Dum91b]

A "pragmatically" circular argument is thus one which employs the rule it is up to justify. For instance, an argument for the justification of the rule of modus ponens (i.e. $(\to E)$) will be "pragmatically circular" if it employs somewhere an instance of the rule of modus ponens.

The substitution lemma (4.1.2) contains essentially the proof-theoretic validation of the comprehension rule (i.e. $(\forall E)$) of second order logic: it implies in particular that a term in the reducibility of $\forall\alpha\sigma$ must be in the reducibility $\sigma[\tau/\alpha]$, for every type τ. At the same time there is a passage in the proof which requires an instance of the comprehension schema of set-theory, in order to state that the reducibility of τ, a property, actually defines a set. That is, at least one passage in the argument which justifies comprehension over the type τ requires comprehension over the reducibility of τ (by induction one can verify that the logical complexity of the property of reducibility for τ is major or equal to the logical complexity of the type τ).

> Speaking of circularity, take for instance comprehension: this schema is represented by extraction, but the reducibility of extraction requires comprehension, reoughly the one under study.
>
> [...]
>
> If one carefully looks at the proof of reducibility for system F, one discovers that the reducibility of type A closely imitates the formula A. Which makes that the extraction on B - the only delicate point - is justified by a comprehension on something which is roughly B. [Gir11]

Coherently with his views on the **VCP** and on the meaning of universal quantification, Dummett claims that the justification of second order quantification is a viciously circular one (for instance in [Dum91a]). Nevertheless, as we discussed in the preceding section, the circularity involved in Girard's trick rather appears as a "pragmatic" one.

What is then the epistemological status of this apparently weaker notion of circularity? Here's Dummett's view:

> [...] if the justification is addressed to someone who genuinely doubts whether the law is valid, and is intended to persuade him that it os, it will fail of its purpose, since he will not accept the argument. If, on the other hand, it is intended to satisfy the philosopher's perplexity about our entitlement to reason in accordance with such a law, it may well do so. [Dum91b]

In other words, "pragmatic" circularity is enough to make the reducibility argument powerless in a debate over the legitimacy of second order quantification, but it is enough to reassure the adept of the second order church of the goodness of his faith.

There exists a vast literature in epistemology over a similar notion of "epistemic circularity": in [Als86] Alston defines an argument for the reliability of a source of belief as "epistemically circular" if the argument relies on premisses that are themselves based on the source. For instance, Alston claims that arguments about the reliability of perception are usually epistemically circular, since they are based on track-records of the form

S has the perceptual belief that p and p is true

and the acknowledgement of their truth presupposes the reliability of perception. Also in this case, the argument is not a viciously circular one since that perception is reliable is not a premiss of the argument. Alston's (rather controversial) diagnosis is that epistemically circular arguments are no harm, unless their purported conclusions *actually are true*:

> Epistemic circularity does not in and of itself disqualify the argument. But even granting this point, the argument will not do its job unless *we are* justified in accepting its premises. [Als86]

Alston's reliabilism is the starting point of a long debate that is not in the scope of this short discussion. At the same time we can retain the notion of "epistemic circularity" to indicate those arguments whose validity presupposes the truth of the conclusion they purport.

Internal approximations of the *Hauptsatz* Getting back to logic, a very interesting case of epistemic circularity arises from the remark that, for any type σ, the reducibility Red_σ can be expressed by a predicate in the language of second order arithmetics. In particular, this implies that, for any term M having type σ the entire argument for the reducibility of M can be proved in second order arithmetics.

This can be seen from the definition of reducibility (or, similarly, from the definition of validity relative to an assignment $\mathcal{N}$): let M be a term having type σ in F; then the parametric reducibility predicates $Red_\sigma[\mathcal{C}_1, \dots, \mathcal{C}_n]$ for the types σ occurring in the typing of M can all be expressed by predicates $\overline{Red_\sigma}[X_1, \dots, X_n](n)$ (where n codes a λ-term) in the language of $\mathbf{HA}^2$, by means of the following clauses:

$$\overline{Red_{\alpha_i}}[X_1, \dots, X_n](n) := X_i(n) \tag{4.3.1}$$

$$\overline{Red_{\sigma\to\tau}}[X_1, \dots, X_n](n) := \forall m\big(\overline{Red_\sigma}[X_1, \dots, X_n](m) \Rightarrow \overline{Red_\tau}[X_1, \dots, X_n](@(n,m))\big) \tag{4.3.2}$$

$$\overline{Red_{\forall\alpha\sigma}}[X_1, \dots, X_n](n) := \forall Z\big(\mathbf{CR}[Z] \Rightarrow \overline{Red_\sigma}[X_1, \dots, X_n, Z](n)\big) \tag{4.3.3}$$

where $@(n,m)$ is the code of the λ-term obtained by applying the λ-term coded by n to the one coded by m and $\mathbf{CR}[Z]$ is the arithmetical first-order predicate (with parameter Z) corresponding to the property "Z is a reducibility candidate". Remark how the logical complexity of the predicate $\overline{Red_\sigma}$ grows along with the logical complexity of the type σ.

Then, starting from the proof of lemma (4.1.1) one can construct, for all closed type σ, a derivation of $\mathbf{CR}[\overline{Red_\sigma}(n)]$. Hence, one can reconstruct in $\mathbf{HA}^2$ the proof of the substitution lemma (4.1.2); remark that the comprehension axioms used in that proofs are here replaced by the comprehension rule of $\mathbf{HA}^2$: if $\overline{Red_{\forall\alpha\sigma}}[X_1, \dots, X_n](n)$ holds, then, since $\mathbf{CR}[\overline{Red_\tau}(x)]$ holds, it follows that $\overline{Red_\sigma}[X_1, \dots, X_n, \overline{Red_\tau}](n)$ holds, and finally (by induction) that $\overline{Red_{\sigma[\tau/\alpha]}}[X_1, \dots, X_n](n)$ holds.

Hence, the reducibility argument showing that $M \in Red_\sigma$ can be formalized in second order arithmetics (by means of some appropriate coding, see for instance [Gir72]).

More generally, if one takes a subsystem of F' of F generated by finitely many extractions, one can formalize in $\mathbf{HA}^2$ the whole reducibility argument for F' (this is shown in [Gir72]). There is a similarity here with the question of the derivability of *reflection principles* in arithmetics (see [KL68], [Gir72]): one can show that, for each subsystem $\mathcal{T}$ of second order arithmetics with a finite number of comprehension axioms, and for each formula A of second order arithmetics, the reflexion principle $Thm_{\mathcal{T}}(\ulcorner A \urcorner) \Rightarrow A$ is derivable in second order arithmetics.

A wonderful application of this idea is at work in the first part of the proof of theorem (2.3.2): one has to recover, from a term M of type $\mathbf{N} \to \mathbf{N}$ in System F computing a recursive function f, a derivation in $\mathbf{HA}^2$ of the totality of the function f. Then one codes directly in $\mathbf{HA}^2$ the reducibility argument for M, by relying on the fact that the latter can use just a finite number of instances of the comprehension schema. Hence one proves in $\mathbf{HA}^2$ that, for any (Church) integer $\mathbf{n}$, there exists a (Church) integer $\mathbf{m}$ which corresponds to the normal form of the term $M\mathbf{n}$. By some coding one recovers then a derivation of the totality of f.

However, as a consequence of Gödel's second incompleteness theorem, it is not possible (if we admit that $\mathbf{HA}^2$ is consistent) to formalize the whole reducibility argument for System F within second order arithmetics. In other words, there exists no predicate $R(n)$, depending on a variable Z, in the language of second order arithmetics such that, for all type σ, there exists a second order formula B_σ such that $R[B_\sigma/Z]$ is equivalent to Red_σ.

Remark that the definition of reducibility is given by means of an iterated inductive definition over the types. In particular, since the induction is non-monotone, as shown by the implicative clause

$$Red_{\sigma\to\tau}(M) \textit{ if and only if } \forall N(Red_\sigma(N) \Rightarrow Red_\tau(MN))$$

it can be shown that, whereas for every type σ, the property Red_σ can be expressed in second order arithmetics, there is no formula of second order arithmetics that can express reducibility of all types in a uniform way (such a problem in the formalization of reducibility is of the same nature as the one in the formalization of the notion of truth, since the latter is defined by a non-monotone induction over formulae).

In definitive, the *Hauptsatz* for second order logic can be approximated *within* second order logic but cannot be globally formalized in it. These has at least two consequences: first, it reveals that the reducibility argument for a derivations involving a certain set of rules can be simulated, or coded, *within* the same logical system by using the same rules. In a sense, these results can be seen as a concrete application of Dummett's "pragmatic" circularity. Second, since the theory in which the global argument is formalized must be able to code the rules of second order logic (in order to formalize the global notion of reducibility), the validity, or reducibility, of the global argument, by presupposing the validity of the stronger theory, already presupposes the validity, or reducibility, of second order logic. In a word, it will be an epistemically circular one in Alston's sense.

On the class Π^0_2 We show how epistemic circularity is at work in the explanation of the proofs for Π^0_2 formulae. Remark that these are the formulae which allow to express the *Hauptsatz*: reducibility arguments essentially prove that, for all term M of a certain recursively encodable system, the reduction sequences starting from M are all finite. This can be formalized as a Π^0_2 arithmetical formula, i.e. as a Π^2 logical formula.

Moreover, in chapter (2) we recalled that proofs of Π^0_2 formulae correspond, under the forgetful translation, to programs which compute a certain recursive function.

We are now able to collect a series of properties of this class of formulae, and of their proofs, which can be useful to frame and to sum up the epistemological issues concerned with proof-theoretic arguments for validity. Indeed, we first observed that, from the viewpoint of the BHK interpretation of proofs, Π^0_n formulae have a delicate epistemological content: technically, a proof of $\forall nA$ is taken to be a method μ which assigns, with each integer k, a proof $\mu(k)$ of $A[\underline{k}/n]$.

In the Π^0_2 case this means that μ assigns, with each integer k, a proof of $\exists m A[\underline{k}/n]$, with A quantifier free: for each k, μ picks up an integer h and a proof of $A[\underline{k}/n, \underline{h}/m]$ (here we use Σ^0_1-completeness).

Constructively this clause appears quite problematic, since it reduces a problem apparently involving infinite verifications into another one, which still requires infinite verifications: how do we verify that μ actually produces, for every integer k, an integer h? Kreisel's solution ([Kre65, Tro69]), as we saw, was to add a second term to the proof, i.e. a "verification" that μ actually does the job. By the way, such a verification would still be an argument saying that, *for every* n, *there exists* an m such that μ applied to n produces an m such that... In a word, *the verification would be a second proof of a* Π^0_2 *formula*. This appears as a real blindspot of the theory of constructions.

We can appreciate the circularity involved if we look at this phenomenon from the viewpoint of theorem (2.3.2): a proof of the totality of a certain recursive function f is a proof of a Π^0_2 statement. At the same time, by theorem (2.3.2), such a proof corresponds to a program M_f which computes the function f. Now, from the viewpoint of the realizability/reducibility interpretation, our proof will be valid exactly when, for all integer k, the program M_f applied to the Church numeral $\mathbf{k}$ produces as output a Church numeral $\mathbf{h}$. In other words, in order to acknowledge the validity of the proof, one has to show that, *for every* integer k, there exists an integer h such that $M_f\mathbf{k}$ reduces to $\mathbf{h}$. That is, the argument for the validity of the proof which shows the totality of f is in the end another argument for the totality of f!

Apparently, then, nothing seems to be gained from the proof-theoretic interpretation of proofs of Π^0_2 formulae: their explanation reproduces in the end exactly the same structure to be explained (i.e. that of the proof of totality of a recursive function). In the end, we are not that far from the explanatory circularity that was reproached to the Tarskian explanation of validity.

Furthermore, as seen in section (4.1), it turns out that cut-elimination is of no help here: Π^0_2 formulae are indeed poor, and enjoy a trivial *Hauptsatz*. This means that from a cut-free proof of a Π^0_2 formula we cannot extract more information than from an arbitrary one.

In the end, two different challenges can be posed with respect to these proofs (which extend more generally to second order logic and its reducibility arguments): firstly, a technical question: *what kind of proof theory can be developed for epistemically circular proofs?* We'll try to develop two possible and complementary answers in the next chapters. Secondly, a philosophical question, which will be left open: *what is the content of an epistemically circular proof?*

4.3.2 A paradox of reducibility

We end this chapter by recalling an extension of the reducibility technique for a strongly impredicative type theory due to Martin-Löf, which was shown to be inconsistent in 1971 ([Gir72]). Girard's paradox is discussed in appendix (B), here we limit ourselves to provide a simplified sketch of the reducibility argument for Martin-Löf's theory. This rather elegant extension of Girard's technique can be seen, on the one hand, as an interesting exercise in the practice of "pragmatic" or "epistemic" circularity; on the other hand, as a proof of the limited epistemological value of these results: the validity of these arguments depends on the reliability of the (set-theoretical) frame in which this is formalized and the latter must be conceived as to "reflect" the properties of the type system. This is why an inconsistent type theory could be proved reducible, in an extremely clever way, within a likewise inconsistent set theory.

> Always the propensity at making circles, illustrated by the faulty normalisation proof given by Martin-Löf for its first system: the extraction on a rather dubious type was justified by a comprehension on more or less the same thing...but the system was nevertheless contradictory. [Gir11]

Dependent types and Martin-Löf's impredicative type theory The most basic distinction in type theory is the one between two categories: the category of *terms* (let us call it λ) and the category of *types* (let us call it ν). When we work in simple type theory we build elements of the two categories inductively. In particular, the rules for the formation of types can be written as typing rules which construct an element of the category ν given one or more elements of the category ν. In the case of implication, we can write:

$$\frac{\Gamma \vdash \sigma : \nu \quad \Gamma \vdash \tau : \nu}{\Gamma \vdash \sigma \to \tau : \nu} \tag{4.3.4}$$

where Γ contains declarations of the form $(\alpha_i : \nu)$ for the free type variables occurring in σ and τ. The implication $\to$ can then be seen as a *constant* of the category $\nu \to \nu \to \nu$. In a word, we can consider types, in addition to terms, as constructions themselves, and provide apposite typing rules for them.

This frame suggests a very natural extension: one can consider types $\tau(x)$ depending on a variable x which is declared of another type σ:

$$\Gamma, (x : \sigma) \vdash \tau(x) : \nu \tag{4.3.5}$$

these *dependent types* were firstly discovered by [DB70] and constitute one of the main feature of Martin-Löf's intuitionistic type theory. Given a type σ and a type $\tau(x)$ depending on a variable x of type σ, one can construct a *dependent product* $(\Pi x : \sigma)\tau$, which is the dependent version of an implication:

$$\frac{\Gamma \vdash \sigma : \nu \quad \Gamma, (x : \sigma) \vdash \tau : \nu}{\Gamma \vdash (\Pi x : \sigma)\tau : \nu} \ (\Pi I) \tag{4.3.6}$$

The introduction rule associated to the dependent product is a dependent extension of the λ-introduction rule of simple type theory:

$$\frac{\Gamma, (x : \sigma) \vdash M : \tau(x)}{\Gamma \vdash \lambda x.M : (\Pi x : \sigma)\tau(x)} \ (\lambda I) \tag{4.3.7}$$

The elimination rule for Π is the dependent extension of the application rule of simple type theory:

$$\frac{\Gamma \vdash M : (\Pi x : \sigma)\tau(x) \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau[N/x]} \ (\lambda E) \tag{4.3.8}$$

Dependent products can be used to translate predicate calculus in type theory in a more direct (and less forgetful) way than by $\mathbb{F}$: the idea is first to translate individuals t into terms t^d of an apposite type ι (in the case of the language of arithmetics, one chooses for the type ι the type $\mathbf{N}$ and translates terms in an obvious way); then, predicates $P(x_1, \dots, x_n)$ are translated into dependent types $P^*(x_1, \dots, x_n)$, where the variables $x_1, \dots, x_n$ are declared of the type of the individuals. The translation of first-order formulae is then immediate:

$$P(t_1, \dots, t_n)^d := P^d(t_1^d, \dots, t_n^d) \qquad (A \Rightarrow B)^d := (\Pi x : A^d)B^d \qquad (\forall x A(x))^d := (\Pi x : \iota)A(x)^d \tag{4.3.9}$$

where, in $(\Pi x : A^d)B^d$, x is a fresh variable, so that B^d does not actually depend on x.

Remark that, if τ does not depend on x, the typing rule for $(\Pi x : \sigma)\tau$ (λI) reduces to the typing rule of the implication type $\sigma \to \tau$:

$$\frac{\Gamma, (x : \sigma) \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \ (\lambda I) \tag{4.3.10}$$

In the following, when τ does not depend on x, we will note $(\Pi x : \sigma)\tau(x)$ simply as $\sigma \to \tau$.

The brilliant idea at the basis of Martin-Löf's original type theory [ML70b] (that here we abbreviate as $ML70$) was to simulate impredicative quantification (i.e. system F) by means of dependent products: a type like, for instance, $\alpha \to \alpha$, depends indeed on the variable α (declared of category ν). The impredicative quantification $\forall\alpha(\alpha \to \alpha)$ of system F corresponds then to a product over *all* types, i.e. all objects of the category ν. Observe that this quantification over the objects of a category is forbidden in the successive (and more well-known) versions of Martin-Löf's type theory (see section (4.2.2)).

Indeed, in $ML70$, the category ν is a type, the type of all types. This allows to write the second order type above as the product $(\Pi\alpha : \nu)(\alpha \to \alpha)$. Now, in order to formally construct this impredicative type, in accordance with the rule (ΠI), he adds a very simple axiom, which states that ν is indeed a type:

$$\frac{}{\vdash \nu : \nu}\ (\nu I) \tag{4.3.11}$$

Now we can construct our impredicative type as follows:

$$\frac{\dfrac{}{\vdash \nu : \nu}\ (\nu I) \qquad \dfrac{(\alpha : \nu) \vdash \alpha : \nu \quad (\alpha : \nu) \vdash \alpha : \nu}{(\alpha : \nu) \vdash (\alpha \to \alpha) : \nu}\ (\Pi I)}{\vdash (\Pi\alpha : \nu)(\alpha \to \alpha) : \nu}\ (\Pi I) \tag{4.3.12}$$

The rules (λI) and (λE) simulate then the rules for universal quantification of system F, in its original version "à la Curry" (see subsection (2.1.3)):

$$\frac{\Gamma, (\alpha : \nu) \vdash M : \sigma}{\Gamma \vdash \lambda\alpha.M : (\Pi\alpha : \nu)\sigma}\ (\lambda I) \qquad \frac{\Gamma \vdash M : (\Pi\alpha : \nu)\sigma \quad \Gamma \vdash \tau : \nu}{\Gamma \vdash M\tau : \sigma[\tau/\alpha]}\ (\lambda E) \tag{4.3.13}$$

To give an example of how this theory works, we show how to build an inductive proof of $(\Pi x : \mathbf{N}_d)\sigma$, where $\mathbf{N}_d$ is the variant of the type $\mathbf{N}$ for Church integers (i.e. the dependent translation of Dedekind's predicate) in $ML70$:

$$\mathbf{N}_d := (\Pi\alpha : \nu)\big((\alpha \to \alpha) \to (\alpha \to \alpha)\big) \tag{4.3.14}$$

Suppose now to have a term (or a construction, in Martin-Löf's terminology) M_0 of type σ, for a certain $\sigma : \nu$, and a construction M_s of type $\sigma \to \sigma$; then we can build a construction of type $(\Pi x : \mathbf{N}_d)\sigma$ as follows:

$$\frac{\dfrac{\dfrac{\dfrac{(x : \mathbf{N}_d) \vdash x : \mathbf{N}_d \quad \vdash \sigma : \nu}{(x : \mathbf{N}_d) \vdash x\sigma : (\sigma \to \sigma) \to (\sigma \to \sigma)}\ (\lambda E) \qquad \begin{matrix}\vdots \\ \vdash M_s : \sigma \to \sigma\end{matrix}}{(x : \mathbf{N}_d) \vdash (x\sigma)M_s : \sigma \to \sigma}\ (\lambda E) \qquad \begin{matrix}\vdots \\ \vdash M_0 : \sigma\end{matrix}}{(x : \mathbf{N}_d) \vdash (x\sigma)M_sM_0 : \sigma}\ (\lambda E)}{\vdash \lambda x.(x\sigma)M_sM_0 : (\Pi x : \mathbf{N}_d)\sigma}\ (\lambda I) \tag{4.3.15}$$

The reader will remark that the term obtained is the same that he would have obtained in system F (à la Curry).

Interestingly, $ML70$ contains a power-set operator $\mathbf{P}$ given by

$$\mathbf{P} := (\Pi\alpha \in \nu)(\alpha \to \nu) \tag{4.3.16}$$

of type $\nu \to \nu$. With the aid of $\mathbf{P}$ we can construct, by recursion, a term $\mathbf{Z} := \lambda x.(x\nu)\mathbf{P}\mathbf{N}_d$, of type $\mathbf{N}_d \to \nu$, which, when applied to a Church numeral $\mathbf{k}$, produces the k-th power of the type $\mathbf{N}$, i.e. the type

$$(\dots((\mathbf{N} \to \underbrace{\nu) \to \nu) \cdots \to \nu) \to \nu}_{k \text{ times}} \tag{4.3.17}$$

remark that the existence of this function in set-theory requires the replacement axiom. This gives a first approximation to the huge expressivity of $ML70$, which is far more expressive than system F.

The reducibility proof We provide a simplified sketch of the reducibility proof for $ML70$ that was given in [ML70b]. The argument we present does not correspond directly to the one in [ML70b], but is reconstructed in analogy with the definition of reducibility candidate given in the last chapter. This proof is of great interest for two reasons: first, it constitutes a very elegant extension of the reducibility technique, and a wonderful exploitation of the "pragmatic" circularity of impredicative systems. Second, and most interestingly, the result of this proof is false: as it is well-known, [Gir72] contains the proof that a not normalizing term can be typed in $ML70$, obtained by reconstructing in the system a version of Burali-Forti's paradox (more on this in chapter (6)).

Indeed, as a result of the discovery of Girard's paradox, Martin-Löf's original system was abandoned and his research on intuitionistic type theory was directed towards a purely predicative development of dependent types (see [ML75, ML84]). Still, it seems very interesting to discuss the details of his reducibility argument since, as we'll see, its fault does not lie in the argument itself (it is a very ingenious generalization of Girard's trick), but rather in the assumptions to be made with respect to the theory in which the argument is formalized: as we saw, by Girard's trick, the impredicativity of a type system is reflected into the impredicativity of the theory in which the reducibility argument is formalized. Now, if the impredicativity of the type system is problematic (or paradoxical, as in this case), such a fault will be transmitted to the theory in which the argument is formalized: as a consequence of the fact that the argument is false, it cannot be formalized in ZF or in other (thought to be) consistent set-theories.

In order to cope with dependent types, we have to enlarge the notion of reducibility candidate: in the case of system F it was enough to associate, with each type σ, a set Red_σ of λ-terms satisfying properties $\mathbf{R1-3}$; now, in order to interpret a type $\tau(x)$ depending on a variable x of type σ, we must take a function associating, with every element in the interpretation of σ, a certain set of λ-terms satisfying $\mathbf{R1-3}$ (this idea is used for instance in the reducibility argument for system F^ω, see section (2.4)).

Let Δ denote the set of "objects", i.e. of all terms and type symbols of $ML70$ (in chapter (6) we'll see that we can take for Δ the set Λ of λ-terms); we denote elements of Δ indistinguishably by small letters $a, b, c, \dots$. Let us call an *extended reducibility candidate* (simply *e.r.c.*) a pair $\mathcal{E} = (s, R)$ made of a set s and a relation $R(a, \xi)$ between objects and elements of the set s which satisfies the following properties:

(ER1) *$R(a, \xi)$ implies that a is strongly normalizing;*

(ER2) *if $R(a, \xi)$ and $a \to a'$, then $R(a', \xi)$;*

(ER3) *if, for all a' such that $a \to_1 a'$, $R(a', \xi)$, then $R(a, \xi)$.*

The idea of the interpretation is the following: whenever we have a typing statement of the form $a : b$, we interpret b by means of an *e.r.c.* $\mathcal{E}_a = (s_b, R_b)$ and a by means of a term $\alpha_a(\xi) \in s_b$ (parametrized by an object $\xi \in s_b$) such that $R_b(a, \alpha_a(\xi))$ holds. In a sense, the set s indicates the "type" of the candidate, i.e. if it is a set of terms, a function from terms to terms, or a function from functions ... whereas the property R corresponds intuitively to a reducibility predicate: it states the reducibility of a, as a result of the construction ξ (that can be thought of as a realizer).

The interpretation of the objects in Δ is the following:

1. *if a is a variable, then $\mathcal{E}_a$ is an arbitrary e.r.c.;*

2. *if a is a variable, then $\alpha_a(\xi) = \xi$;*

3. *if $a = (\Pi x : a_1)a_2(x)$ then $\mathcal{E}_a = (s_a, R_a)$, where s_a is the set of all functions η which, to an object $\xi \in s_{a_1}$, associate an object $\eta(\xi) \in s_{a_2(x)}(\xi)$, and $R_a(b, \eta)$ is defined by the following clause:*

$$R_a(b, \eta) \text{ if, and only if } \forall c \forall \xi (R_{a_1}(c, \xi) \Rightarrow R_{a_2(x)}(\xi)(bc, \eta(\xi))) \tag{4.3.18}$$

4. *if $a = \lambda x.a'$, with x of type b_1 and a' of type b_2, then $\alpha_a(\xi)$ is the function which, to an object $\xi \in s_{b_1}$, associates $\alpha_{a'(x)}(\xi)$.*

5. *if $a = bc$, then $\alpha_a = \alpha_b(\alpha_c)$;*

6. *if a is ν, then $\mathcal{C}_\nu = \alpha_\nu = (s_\nu, R_\nu)$, where s_ν is a* class *containing all pairs (s, R), where s is a set and R a relation over Δ and s, and $R_\nu(a, (s, R))$ holds when a is strongly normalizable and (s, R) satisfies* **ER1 − 3** *(i.e. it is an e.r.c.).*

The reducibility argument proceeds in this way: given a derivation of a typing judgement of the form

$$(x_1 : a_1), \ldots, (x_n : a_n(x_1, \ldots, x_{n-1})) \vdash c(x_1, \ldots, x_n) : a(x_1, \ldots, x_n) \tag{4.3.19}$$

one has to show, by induction on the derivation, two things:

- first, that for all choices of elements $\xi_1 \in s_{a_1}, \ldots, \xi_n \in s_{a_n(x_1,\ldots,x_{n-1})(\xi_1,\ldots,\xi_{n-1})}$, the pair

$$\Big(s_{a(x_1,\ldots,x_n)}(\xi_1, \ldots, \xi_n), R_{a(x_1,\ldots,x_n)}(\xi_1, \ldots, \xi_n)\Big) \tag{4.3.20}$$

  is an element of s_ν satisfying **R1 − 3**; remark that this implies in particular showing that the elements of the pair above are sets.

- second, that for all objects $c_1, \ldots, c_n(x_1, \ldots, x_{n-1})$ such that $R_{a_1}(c_1, \alpha_{c_1}), \ldots,$ $R_{a_n(c_1,\ldots,c_{n-1})}(\alpha_{c_1}, \ldots, \alpha_{c_{n-1}})(c_n(c_1, \ldots, c_{n-1}), \alpha_{c_n(c_1,\ldots,c_{n-1})})$,

$$R_{a(c_1,\ldots,c_n)}(\alpha_{c_1}, \ldots, \alpha_{c_n})(c(c_1, \ldots, c_n), \alpha_{c(c_1,\ldots,c_n)}(\alpha_{c_1}, \ldots, \alpha_{c_n})) \tag{4.3.21}$$

  holds.

For the rules $(\Pi I), (\lambda I), (\lambda E)$, the proof is very painful to write, but essentially follows the pattern of the reducibility arguments already presented (it is indeed more or less clear that the clause (4.3.18) is a generalization to dependent types of the usual clause for the reducibility of implication). Remark in particular that the only set-theoretical constructions needed to state that the interpretations of types are *e.r.c.* are essentially two: the function-space construction, i.e. the construction that, given two sets S, T, produces the set T^S of all functions from S to T, and the cartesian product construction, i.e. the construction that, given two sets S, T, produces the set $S \times T$ of all ordered pairs of elements of S and T. This means that this part of the argument can be formalized in a very weak set-theoretical universe, like $V_{\omega+\omega}$[2].

The really problematic part of the argument concerns the strongly impredicative axiom $\vdash \nu : \nu$: it must be shown indeed that the *class* s_ν of all pairs (s, R) made of a *set* s and of a relation $R \subseteq \Delta \times s$ is a *set* and, moreover, that $R_\nu(\nu, \alpha_\nu)$ holds, i.e. that (s_ν, R_ν) is an *e.r.c.*.

Let us start from the latter: let us suppose that s_ν is a set; we have to show that $R_\nu(a, (s, R))$ satisfies **ER1 − 3**. **R1** is immediate from the definition of R_ν; as for **ER2**, if $R_\nu(a, (s, R))$

[2]The *Van Neumann* universes V_α are defined, for α an ordinal number, by transfinite induction as follows: $V_0 = \emptyset$, $V_{\alpha+1} = \wp(V_\alpha)$ and for λ limit, $V_\lambda = \bigcup_{\beta<\lambda} V_\beta$.

holds, then a is strongly normalizable and R satisfies **ER1 − 3**; now, if $a \to a'$, a' is strongly normalizable too and thus $R_\nu(a', (s, R))$ holds. For **ER3**, let us suppose that, for all a' such that $a \to_1 a'$, $R_\nu(a', (s, R))$ holds; then a is strongly normalizable and (s, R) satisfies **ER1 − 3**, hence $R_\nu(a, (s, R))$ holds.

It finally remains to show that s_ν is a set. We proceed as follows: we fix a set-theoretical universe V closed with respect to the basic operations of cartesian product and powerset operation (it suffices to take $V = V_\alpha$, with $\alpha \geq \omega$), and we try to individuate the properties needed for V to contain s_ν as a subset. Since s_ν contains all the pairs (s, R) where s is a set and $R \subseteq \Delta \times s$, it follows that s_ν is contained in the class

$$V \times \wp(\Delta \times V) \tag{4.3.22}$$

where we use the fact that V is transitive[3]. Thus, in order to assert that s_ν is a set, we must require that $V \times \wp(\Delta \times V) \in V$ (again, by transitivity). Now, since V is closed with respect to the power-set and the cartesian product, this reduces to require

$$V \in V \tag{4.3.23}$$

which is false for all universes V_α, for α an ordinal number. In other words, the strongly impredicative axiom $\vdash \nu : \nu$ "circularly" looks for a strongly impredicative universe such that $V \in V$.

[3]I.e. the property that $S \in T \in V$ implies $S \in V$.

Part III

Explaining how

Chapter 5

Impredicativity and parametric polymorphism

The intuitive explanation of a proof of a universally quantified type $\forall\alpha\sigma$ as a function from types to terms, so as the set-theoretic interpretation of quantification as an intersection over *all* sets, run into some difficulties and paradoxes due to impredicativity. Rather, a proof of $\forall\alpha\sigma$ should be thought as a proof of the (simple) type σ in which the type variable α stands for an "arbitrary" or "generic" type.

This intuition (at the basis of Carnap's defense of impredicativity in [Car83]) finds a robust mathematical grounding in the theories of *parametric polymorphism* ([Rey83]) and in the *dinatural interpretation of polymorphism* ([BFSS90, GSS92]). These approaches provide a powerful explanation of impredicative quantification which, though having been widely known in the computer science community since the eighties, has been substantially ignored in the philosophical debate (with the sole exception of [LF97]). The fact that a second order proof cannot actually discriminate between different types imposes indeed very strong constraints on the form that such a proofs can have. These constraints allow then to reconstruct the internal structure of proofs from the study of their semantics (this "magical" aspect of parametricity was indeed resumed by Wadler's slogan "Theorems for free!" [Wad89]).

In this chapter, we first recall the parametric and dinatural interpretation of polymorphism, so as the paradoxes which arise from the violation of such "generic" quantification. Then we present some technical results which highlight the finitary character of this explanation of impredicative quantification: first, a purely combinatorial description of the constraints imposed by the parametricity and dinaturality conditions is provided, by coding both conditions through the application to reducible terms of certain simply typed λ-terms H_σ, K_σ. This syntactical criterion allows then to derive the main result of this chapter: the $\mathbf{\Pi}^1$-completeness theorem (5.2.4) (conjectured in [Gir11]), which states that a closed normal term in the reducibility of a closed type of the form $\forall\overline{\alpha}(\sigma \to \tau)$ can be given the type $\sigma \to \tau$ in simple type theory.

The proof of this theorem exploits the "magic" of parametric polymorphism, yielding a characterization of the internal structure of reducible λ-terms. As a consequence, a bridge between the interactionist and the inferentialist interpretation of proofs presented in chapter (3) is obtained: a corollary of the theorem (5.2.4) is that one can recover a "last rule condition" for reducible closed normal λ-terms.

5.1 Set-theoretic *vs* "generic" quantification

We recall Reynolds's result that there exists no set-theoretical model of System F in which the implication type is interpreted by the function space. This result provides a (quite neglected in the philosophical literature) proof-theoretic argument against the identification of second order quantification and set-theoretic intersection and contradicts Shapiro's thesis (see subsection (1.2.1)) of a substantial homogeneity between the proof-theory of second order logic and set theory.

The set-theoretic intuition of second order quantification as a quantification over all sets must be then replaced by a different one. We recall Carnap's intuition that a proof of a second order statement $\forall X A$ is not built by "running all possible cases" but by producing an argument for A in which the variable X stands for an "arbitrary property" ([Car83]). We recall then two results which constitute (following [LF97]) a mathematical vindication of Carnap's remark: first, Girard's remark in [Gir72] that, by adding to System F a non "generic" term J_σ, one obtains a counterexample to reducibility; second, the *genericity theorem* (5.1.1), which asserts that two terms which are equal on one type, must be equal on every type.

5.1.1 Reynolds' paradox: why second order logic is not set-theory

We first provide an informal description of how impredicative quantification can be used to produce non set-theoretic functions. Next we present in some more detail the idea of Reynolds' proof.

Impredicativity produces "too many" functions In the previous chapter we recalled the main objections against the use of impredicatively defined notions. For instance, if we define a set N as *the smallest set* containing 0 and closed with respect to the successor function, and then we wish to show that this set N is itself closed under the successor function, we run into a form of circularity: on the one hand N is defined by reference to a totality of sets to which it belongs (the totality of sets containing 0 and closed under the successor function); on the other hand, we must use the fact that N belongs to that totality to show that N is closed under the successor function.

Indeed, if we wished to verify the closure of N under the successor operation *set by set*, we would stumble on a gross circularity: we should verify, for each set S, *included* N, that it is closed under the successor function.

We can describe this kind of argument in a set-theoretic frame: let, for any set s, $J(s)$ be the set containing 0 and, for any $x \in s$, the set $x \cup \{x\}$ (i.e. the set-theoretic successor of x). The application $J(s)$ defines indeed a monotone operator from the category of sets to itself, i.e. a map such that, for any two sets s, t such that $s \subseteq t$, $J(s) \subseteq J(t)$.

The set N of natural numbers can then be defined as "the smallest set" N such that $J(N) \subseteq N$. Indeed this definition imitates Dedekind's definition since it states that N is the intersection of all the sets containing 0 and closed under the successor operations, i.e. the intersection of all the sets closed under the operator J. At the same time this definition immediately implies that N is closed under the operator J.

Remark that definitions of sets by means of expressions like "the smallest set such that" should be regarded with suspect from the viewpoint of set-theory: they presuppose indeed a quantification over *all sets* which is not allowed in standard axiomatic set-theories.

More generally, let $\phi[\alpha]$ be a type in which the variable alpha occurs positively and let Φ be the type $\forall\alpha\big((\phi[\alpha] \to \alpha) \to \alpha\big)$. Intuitively, the type Φ can be though as "the smallest set" w

such that $F(w) \subseteq w$, i.e. closed with respect to the operator F over sets which is expressed by the type $\phi[\alpha]$.

It is possible to build terms inhabiting the following two types (see [Coq86] for a proof of this fact):

$$Func(\phi) := \forall\alpha\forall\beta\big((\alpha \to \beta) \to (\phi[\alpha] \to \phi[\beta])\big) \tag{5.1.1}$$

$$Ind(\phi) := \forall\alpha\big((\phi[\alpha] \to \alpha) \to (\Phi \to \alpha)\big) \tag{5.1.2}$$

Set-theoretically, $\phi[\alpha]$ corresponds to a *monotone operator* $F : Set \to Set$ from the category of sets to itself. This means that, for all sets s, t, if $s \subseteq t$, then $F(s) \subseteq F(t)$. Now the type (5.1.1) expresses the *functoriality* of the operator F: it states that, for all sets s, t and for all function $f : s \to t$, there exists a function $F(f) : F(s) \to F(t)$.

The type (5.1.2) corresponds to a *generalized induction principle for* F: it expresses the fact that, if s is a set which is closed with respect to F, then w must be contained in s (since w is contained in any set closed with respect to F). For instance, if s is a set containing 0 and closed under the successor operation, then $N \subseteq s$.

Reynolds' ingenious idea was to exploit this elegant theory in order to construct types which, when interpreted set-theoretically, would contain "too many" functions. Consider the type $\omega[\alpha] := (\alpha \to \mathbf{Bool}) \to \mathbf{Bool}$, where **Bool** is any type with two elements. Set-theoretically, $\omega[\alpha]$ corresponds to the monotone operator $O(s) = \wp(\wp(s))$; now, the type $\Omega = \forall\alpha((\omega[\alpha] \to \alpha) \to \alpha)$ should correspond to "the smallest set" closed under to operator $O(s)$, i.e. to "the smallest set" o such that $O(o) = \wp(\wp(o)) \subseteq o$. Hence, the existence of such a set would imply the existence of an (injective) function from the double power of a set to the set itself, contradicting Cantor's theorem.

"Polymorphism is not set-theoretic" This is indeed the title of a famous paper [Rey84] by Reynolds, where he exploits the idea sketched above to prove that there exists no set-theoretic model of System F. By a set-theoretic model he essentially meant an interpretation which assigns sets to the types of system F and elements of those sets to typed terms in such a way that the type $\sigma \to \tau$ is interpreted as the set of functions from the interpretation of σ to the interpretation of τ.

More formally, the idea is to consider an interpretation $[\![\_]\!]$ parametrized by a map η, which assigns sets to the type variables and, to any variable x declared of type σ, an element $\eta(x) \in [\![\sigma]\!]\eta$. The interpretation of types must respect the clauses:

$$[\![\alpha]\!]\eta = \eta(\alpha) \tag{5.1.3}$$

$$[\![\sigma \to \tau]\!]\eta = [\![\tau]\!]\eta^{([\![\sigma]\!]\eta)} \tag{5.1.4}$$

The interpretation of terms must respect the clauses below (we use superscripts to note the types):

$$[\![x^\sigma]\!]\eta = \eta(x) \tag{5.1.5}$$

$$[\![M^{\sigma\to\tau}N^\sigma]\!]\eta = [\![M^{\sigma\to\tau}]\!]\eta([\![N^\sigma]\!]\eta) \tag{5.1.6}$$

$$[\![\lambda x^\sigma.M^\tau]\!]\eta = \{(u,v) | u \in [\![\sigma]\!]\eta \text{ and } v = [\![M^\tau]\!](\eta \cup \{x \mapsto u\})\} \tag{5.1.7}$$

Remark that the definition of the sets $[\![\sigma]\!]\eta$ and $[\![M]\!]\eta$ in Reynold's set-theoretic interpretation closely resembles, respectively, the definition of the R_a and of the α_a in the reducibility interpretation of Martin-Löf's paradoxical type theory (see (4.3.2)). In particular, λ-abstracted terms are interpreted as certain functions in the appropriate function space, and term application is interpreted as function application. Several analogies can be indeed found between

Girard's paradox (concerning Martin-Löf's type theory) and Reynold's argument (see [Coq86] for a discussion).

We are now able to give a sketch of Reynold's proof: let us first suppose that a set-theoretic interpretation $\llbracket \_ \rrbracket\eta$ in the sense above exists. The argument is developed in three steps:

1) There exists a set b with at least two elements. This is shown by taking as b the interpretation of the Boolean type $\mathbf{Bool} =_{def} \forall\alpha(\alpha \to \alpha \to \alpha)$. One indeed easily shows that to the two distinct λ-terms $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$ there must correspond two distinct elements of b.

2) One considers then the positive operator $\omega[\alpha] = (\alpha \to \mathbf{Bool}) \to \mathbf{Bool}$. One can verify that the λ-term $func$ below

$$func := \lambda f.\lambda z.\lambda u.z(\lambda x.u(fx)) \tag{5.1.8}$$

has type $Func(\omega) \equiv \forall\alpha\forall\beta((\alpha \to \beta) \to (\omega[\alpha] \to \omega[\beta]))$ and that the λ-term ind below

$$ind := \lambda f.\lambda u.uf \tag{5.1.9}$$

has type $Ind(\omega) \equiv \forall\alpha((\omega[\alpha] \to \alpha) \to (\Omega \to \alpha))$, where Ω is the type $\forall\alpha((\omega[\alpha] \to \alpha) \to \alpha)$.

By relying on the considerations above, the interpretation $W(s)$ of $\omega[\alpha]$ is shown to be a functor from the category of sets to itself which satisfies a generalized induction principle: this means that, for all sets s, t and function $f : s \to t$, one can define a function $\llbracket func \rrbracket\eta(f) : W(s) \to W(t)$, and that for all set s and function $g : W(s) \to s$, there exists a function $\llbracket ind \rrbracket\eta(g) : w \to s$, where w is the interpretation of Ω.

3) Finally, the terms $func$ and ind above are used to construct the λ-term inj below

$$inj := \lambda z.\lambda f.f(func((ind)f)z) \tag{5.1.10}$$

which has type $\omega[\Omega] \to \Omega$.

Now, by summing up all the results, he can state the following: there exists a function $\llbracket inj \rrbracket\eta : W(w) \to w$ such that, for any set s and for any function $g : W(s) \to s$, there exists a function $\llbracket ind \rrbracket\eta(g) : w \to s$ which makes the following diagram commute:

$$\begin{tikzcd}
W(w) \arrow[r, "\llbracket func \rrbracket\eta(\llbracket ind \rrbracket\eta(g))"] \arrow[d, "\llbracket inj \rrbracket\eta"'] & W(s) \arrow[d, "g"] \\
w \arrow[r, "\llbracket ind \rrbracket\eta(g)"] & s
\end{tikzcd} \tag{5.1.11}$$

Now, by means of general results on *initial algebras* (see [LS86]) it can be shown that the function $\llbracket in \rrbracket\eta(g)$ is injective. Thus, there exists an injective function from $b^{(b^w)}$ to b, contradicting Cantor's theorem.

Reynolds' paradox establishes the following fact: if we wish to interpret proofs of an implication $A \Rightarrow B$ as functions from the interpretation of A to the interpretation of B, then the naïve interpretation of universal quantification as a quantification (or an intersection) over *all* sets must be abandoned: the interpretation of a universally quantified formula is definitely too big to be itself a set.

Remark the difference with the case of the reducibility interpretation (section (3.2.2)): Girard's trick states that the interpretation of a universally quantified formula (as an intersection over all reducibility candidates) is actually a set. However the reducibility of an implication type $\sigma \to \tau$ is not the function space $Red_\tau^{Red_\sigma}$ but the (much smaller) set $Red_\sigma \to Red_\tau$ (see section (3.2.2)).

5.1.2 Carnap's defense of impredicativity

Reynolds' result faces us with a compelling question: once we discard the set-theoretic intuition of universal quantification as set-theoretical intersection, how can we make sense of a proof involving an impredicatively defined concept? In particular, once we consider the second order Dedekind's predicate $N(x)$, how can we justify the validity of the "circular" argument for the fact that $N(x)$ holds of 0 and is closed under the successor function, if we cannot rely on the intuition that, in order to prove that $N(t)$ holds, one has to prove that t belongs to *any* set containing $\underline{0}$ and closed under the successor function?

In his defense of impredicative definitions in [Car83], Carnap discusses this form of circular arguments:

> For example, to ascertain whether the number three is inductive, we must, according to the definition, investigate whether every property which is hereditary and belongs to zero also belongs to three. But if we must do this for every property, we must also do it for the property "inductive" which is also a property of numbers. Therefore, in order to determine whether the number three is inductive, we must determine among other things whether the property "inductive" is hereditary, whether it belongs to zero and, finally - this is the crucial point - whether it belongs to three. But this means that it would be impossible to determine whether three is an inductive number. [Car83]

Unsatisfied by the predicativist answer that circularly defined concepts should be simply abandoned, as this would imply the impossibility to prove as simple a fact as the one that three is a natural number, Carnap tries to develop a different answer:

> If we had to examine every single property, an unbreakable circle would indeed result, for then we would run headlong against the property "inductive". Establishing whether something had it would then be impossible in principle, and the concept would be meaningless. But the verification of a universal logical or mathematical sentence does not consist in running through a series of individual cases [...] The belief that we must run through all individual cases rests on a confusion of "numerical generality" [...] We do not establish specific generalities by running through individual cases but by logically deriving certain properties from certain others. [Car83]

Carnap's argument is that the intuition that the verification of a universally quantified formula consists in the verification of all its instances does not reflect the way in which proofs are actually constructed:

> [...] that the number two is inductive means that the property "belonging to two" follows logically from the proeprty "being hereditary and belonging to zero". In symbols, $f(2)$ can be derived for an arbitrary f from $Her(f) \wedge f(0)$ by logical operations. [...] First, the derivation of $f(0)$ from $Her(f) \wedge f(0)$ is trivial [...] The remaining steps are based on the definition of the concept "hereditary"
>
> $$Her(f) =_{def} \forall n(f(n) \Rightarrow f(n+1))$$
>
> Using this definition, we can easily show that $f(0+1)$ and hence $f(1)$ are derivable from $Her(f) \wedge f(0)$ and thereby [...] we can derive $f(1+1)$ and hence $f(2)$ fro $Her(f) \wedge f(0)$, thereby showing that the number two is inductive. [Car83].

The derivation that Carnap is describing corresponds to the usual proof of $\mathbb{N}(\underline{2})$, the one which translates forgetfully into the Church numeral $\mathbf{2} = \lambda f.\lambda x.f(fx)$ (remark indeed that he uses one time the hypothesis $f(0)$ - i.e. the variable x - and two times the hypothesis $Her(f)$ - i.e. the variable f).

The proof above represents thus a finite argument schema that can be reproduced for an "arbitrary f"; it is indeed not necessary, for Carnap, to run into all the possible instances of the schema to recognize that it will work for them.

> If we reject the belief that it is necessary to run through individual cases and rather make it clear to ourselves that the complete verification of a statement means nothing more than its logical validity for an arbitrary property, we will come to the conclusion that impredicative definitions are logically admissible. [Car83]

The intuition behind these remarks is that the argument schema uses the property f as a *parameter*: for each choice of f, the argument can be reproduced uniformly with respect to f. In [LF97] Carnap's intuitions are compared with a remark by Herbrand on "prototype proofs" of a universally quantified statement:

> [...] when we say that a theorem is true for all x, we mean that for each x individually it is possible to iterate its proof, which may just be considered a *prototype* of each individual proof. [Her71]

Interestingly, in [LF97] it is also claimed that Carnap's intuition

> [...] seems very close to the "realizability interpretation" in Intuitionistic Logic [...] Carnap seems to claim that the possibility of an analysis of provability justifies "logical admissibility". [LF97]

Carnap is indeed advocating the fact that the argument provides, for each property f, an actual proof that $f(2)$ holds, under the assumptions that $Her(f)$ and $f(0)$ hold. In a sense, we might say that his argument (or, better, the term **2**) is a realizer of the second order formula $\forall f(Her(f) \Rightarrow f(0) \Rightarrow f(2))$.

In the next section we'll try to give substance to Carnap's intuition (following [LF97]) by means of the notion of *parametric polymorphism*; in particular, it will be shown that, by imposing a parametricity constraint over reducible λ-terms (i.e. by imposing that their dependence over type variable be "prototypical"), we will show that we can obtain information over the form of such terms, yielding a powerful proof-theoretical analysis of second order quantification.

5.1.3 The operator J and the genericity theorem

In [Gir72] the following argument is presented[1]: let us add to System F a constant 0 of type $\forall\alpha\alpha$ and let us suppose that there exists a term J of type $\forall\alpha(\sigma \to \alpha)$, where σ is $\forall\beta\beta \to \forall\beta\beta$ satisfying the following reduction rules:

$$J_\sigma M \to M \tag{5.1.12}$$

$$J_\rho M \to 0 \text{ if } \rho \neq \sigma \tag{5.1.13}$$

Remark that the reduction behavior of J depends on the type on which it is extracted.

Now, from the hypothesis of the existence of J, we can construct a counterexample to the reducibility of System F: the reader will find below a typing derivation of the term $O := (J_\sigma \lambda x.(x\{\sigma\})x)\Lambda\beta.(J_\beta)\lambda x.(x\{\sigma\})x$.

[1] We discuss the argument in the original version *à la Church* of System F, see subsection (2.1.3).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(x:\forall\beta\beta)\vdash x:\forall\beta\beta}{(x:\forall\beta\beta)\vdash x:\sigma} \quad (x:\forall\beta\beta)\vdash x:\forall\beta\beta}{(x:\forall\beta\beta)\vdash (x\{\sigma\})x:\forall\beta\beta}\,(@)}{\vdash \lambda x.(x\{\sigma\})x:\sigma}\,(\lambda) \quad \begin{array}{c}\vdots\\ \vdash J_\sigma:\sigma\to\sigma\end{array}}{\vdash (J_\beta)\lambda x.(x\{\sigma\})x:\beta}\,(@)}{\vdash (J_\beta)\lambda x.(x\{\sigma\})x:\forall\beta\beta}\,(\forall I)}{\vdash J_\sigma\lambda x.(x\{\sigma\})x:\sigma}\,(\forall E) \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(x:\forall\beta\beta)\vdash x:\forall\beta\beta}{(x:\forall\beta\beta)\vdash x:\sigma} \quad (x:\forall\beta\beta)\vdash x:\forall\beta\beta}{(x:\forall\beta\beta)\vdash (x\{\sigma\})x:\forall\beta\beta}\,(@)}{\vdash \lambda x.(x\{\sigma\})x:\sigma}\,(\lambda) \quad \begin{array}{c}\vdots\\ \vdash J_\beta:\sigma\to\beta\end{array}}{\vdash (J_\beta)\lambda x.(x\{\sigma\})x:\beta}\,(@)}{\vdash \Lambda\beta.(J_\beta)\lambda x.(x\{\sigma\})x:\forall\beta\beta}\,(\forall I)}{\vdash O:\forall\beta\beta}\,(@) \tag{5.1.14}$$

The reduction behavior of O is the following:

$$O \to_1 (\lambda x.(x\{\sigma\})x)J_\beta\lambda x.(x\{\sigma\})x \to_1 ((\Lambda\beta.(J_\beta)\lambda x.(x\{\sigma\})x)\{\sigma\})(J_\beta\lambda x.(x\{\sigma\})x \to_1 O \tag{5.1.15}$$

It follows then that System F cannot contain terms discriminating between types. We can also verify that J is not in $Red_{\forall\alpha(\sigma\to\alpha)}$: indeed, if it were, then, for all candidate of reducibility $\mathcal{C}$, J would be in the set $Red_\sigma \to \mathcal{C}$; in particular, for all reducibility candidate $\mathcal{C}$ and for all $M \in Red_\sigma$, $JM \in \mathcal{C}$. Now since, $J_\sigma M = M$, it follows that $Red_\sigma \subseteq \mathcal{C}$, for all $\mathcal{C}$, which is false: for instance, the term $\lambda x.(x)x$ is in Red_σ but not in $Red_\sigma \to Red_\sigma$ (as this would imply that $(\lambda x.(x)x)\lambda x.(x)x \in Red_\sigma$, which contradicts **R1**).

A thorough analysis of the conditions underlying Girard's example is contained in [LMS93]. There an extension Fc of System F is proposed which contains the following axiom:

$$\textbf{Axiom C:}\ \textit{if } M \textit{ has type } \forall\alpha\sigma \textit{ and } \alpha \notin FV(\sigma) \textit{ then, for all } \tau,\tau',\ M\{\tau\} = M\{\tau'\} \tag{5.1.16}$$

Axiom C states that the term M cannot depend on the type to be substituted for α if α is not free in σ. Axiom C is compatible with System F, in the sense that there are models of System F which satisfy Axiom C (furthermore, the authors of [LMS93] state that they know of no non-trivial model of System F in which Axiom C is false).

Then, with respect to the system Fc the following theorem is proved:

Theorem 5.1.1 (Genericity theorem, [LMS93])**.** *Let M and N have type $\forall\alpha\sigma$. Then, if there exists a type τ such that $M\{\tau\} = N\{\tau\}$, then $M = N$.*

The genericity theorem states that polymorphic terms which are equal *on one* input type must be equal *on all* input types. This result reveals a syntactical property of polymorphic terms which, as it is advocated in [LF97], seems to vindicate Carnap's intuition: if two distinct proofs of a universally quantified formula $\forall XA$ can be made equal to another proof of the same formula when the two are applied to a certain predicate P, then the two proofs must be able to discriminate between predicates. Hence the variable X cannot stand in those proofs for an "arbitrary property", since the "argument schema" of the two proofs changes according to the predicate substituted for X.

Theorem (5.1.1) is a very strong theorem which tells that, in a sense, there are "not so many" polymorphic terms. In the next section we'll discuss some semantic and syntactic properties which characterize this aspect of second order quantification and we'll show their extreme power: since a derivation of a universally quantified formula corresponds to a function over types which is, in a sense, the same over all types, it follows that there are very few degrees of freedom for defining such a function.

5.2 Parametricity and the completeness of simple type theory

We present the two main formalizations of the intuitive idea of parametric quantification: Reynold's *parametricity* ([Rey83]) and the dinatural interpretation ([BFSS90, GSS92]). In particular we show the very strong constraints that these conditions force on the "degrees of freedom" of terms.

As Reynolds' parametricity is reformulated in the setting of reducibility candidates, it is proved (theorem (5.2.2)) that a closed normal term in the reducibility of a universally closed simple type must by parametric (which is obviously false for non closed simple types); moreover, it is proved that a parametric reducible term must satisfy the dinaturality criterion (theorem (5.2.3)). Finally, by relying on a syntactical formulation of the latter a $\mathbf{\Pi}^1$-completeness theorem (5.2.4), which is the main result of this chapter, is proved: if M is a closed normal term in $Red_{\forall\overline{\alpha}(\sigma\to\tau)}$, with σ, τ simple types, then $\vdash M : \sigma \to \tau$ is derivable in simple type theory.

5.2.1 The mathematics of parametricity

Parametric families Let us consider the class of *families* δ_s, with s running over *all* sets, such that, for all s, δ_s is a function from s to the cartesian product $s \times s$. Since the families δ_s are indexed over the class of all sets, the class just described appears to be too big to be contained in usual set-theories.

However, suppose we want our families δ_s to depend "generically" with respect to their index-set. The intuition is the following: δ_s should send an element x of an *arbitrary* set s into an element of its cartesian product, in a way which does not depend on the information that $x \in s$. Let now s, t be two arbitrary sets and f a function from s to t; if x is an arbitrary element of s, then $f(x)$ is an element of t whose only property "visible to δ_s" is its "relatedness" to x by means of the function f. Now, the idea is that the action of δ_s should be so "generic" with respect to s not to break the "relatedness" of x and $f(x)$; in symbols, we should have

$$f \times f(\delta_s(x)) = \delta_t(f(x)) \tag{5.2.1}$$

which corresponds to require that the following diagram should commute for all sets s, t and $f : s \to t$:

$$\begin{tikzcd} s \arrow[r, "\delta_s"] \arrow[d, "f"] & s \times s \arrow[d, "f\times f"] \\ t \arrow[r, "\delta_t"'] & t \times t \end{tikzcd} \tag{5.2.2}$$

One can easily be convinced then that the requirement above is a very strong one: it collapses a huge class of families into a set with just one element, the diagonal family δ_s given by

$$\delta_s(x) = (x, x) \tag{5.2.3}$$

To be convinced of that, it suffices to take $s = t = 2$, the set with two elements $0, 1$: let us suppose that, for a certain $x \in 2$, $\delta_2(x) = (y_1, y_2)$ with $y_1 \neq x$ or $y_2 \neq x$; for instance, let us suppose $\delta_2(0) = (1, 0)$ and let us choose as $f : 2 \to 2$ the function with always returns 0. Now the diagram above implies that δ_2 should commute with f, whereas one has

$$(0, 0) = f \times f(\delta_2(0)) \neq \delta_2(f(0)) = (1, 0) \tag{5.2.4}$$

In definitive, the only thing a function "generically" sending sets into their cartesian product can do is to take its argument as a "black box" and to duplicate it.

The remarks above illustrate the ideas at the heart of Reynolds' notion of *parametricity*: intuitively, if the action of δ_s does not really depend on s, then, if we take two sets s, t and an arbitrary binary relation $r \subseteq s \times t$, then the action of δ_s should not be able to "break" their "relatedness". More formally, let us first extend r to the cartesian products $s \times s$ and $t \times t$: we define a relation $r \times r \subseteq (s \times s) \times (t \times t)$ by

$$(x, y) r \times r (x', y') \text{ if and only if } x r x' \text{ and } y r y' \tag{5.2.5}$$

Reynolds' parametricity corresponds then to ask that, for all $x \in s, y \in t$, if xry, then $\delta_s(x) r \times r \delta_t(y)$.

We can easily verify that parametricity implies the diagrammatic version of "genericity" sketched above: let s, t be sets and f be a function $f : s \to t$; let then $r^f \subseteq s \times t$ be the relation defined by

$$x r^f y \text{ if and only if } f(x) = y \tag{5.2.6}$$

which codes the "relatedness" of the example above. Parametricity implies that, for all $x \in s, y \in t$, if $f(x) = y$, then

$$f\big(\delta_s(x)\big)_1 = \big(\delta_s(y)\big)_1 \text{ and } f\big(\delta_s(x)\big)_2 = \big(\delta_s(y)\big)_2 \tag{5.2.7}$$

and thus

$$f\big(\delta_s(x)\big)_1 = \big(\delta_s(f(x))\big)_1 \text{ and } f\big(\delta_s(x)\big)_2 = \big(\delta_s(f(x))\big)_2 \tag{5.2.8}$$

which is exactly what is expressed by diagram (5.2.2) and, in particular, implies $\delta_s(x) = (x, x)$. This simple example illustrates the idea in the proof of theorem (5.2.3) which establishes that the parametricity criterion implies the dinaturality criterion (see below), which is a generalization of the diagrammatic criterion above.

Parametric polymorphism There exists a quite vast literature on semantical characterization of the genericity of second order type quantification. The first informal remarks can be found in [Str67], where a distinction is made between *ad hoc* polymorphism and *parametric* polymorphism, from a computer science perspective:

> In ad hoc polymorphism there is no single systematic way of determining the type of the result from the type of the arguments. [...] All the ordinary arithmetic operators and functions come into this category. [...]
>
> Parametric polymorphism is more regular and may be illustrated by an example. Suppose f is a function whose argument is of type α and whose results is of β (so that the type of `f` might be written $\alpha \to \beta$), and that `L` is a list whose elements are all of type α (so that the type of `L` is α`list`). We can imagine a function, say `Map`, which applies `f` in turn to each member of `L` and makes a list of the results. Thus `Map`[`f`, `L`] will produce a β`list`. We would like `Map` to work on all types of list provided `f` was a suitable function, so that `Map` would have to be polymorphic. However its polymorphism is of a particularly simple parametric type which could be written
>
> $$(\alpha \to \beta, \alpha\mathtt{list}) \to \beta\mathtt{list}$$
>
> where α and β stand for any types. [Str67]

It must be observed that Strachey's remarks anticipate Girard's System F by five years and Reynold's work on parametric polymorphism by more than ten years.

An example of ad hoc polymorphism would be a program for addition which takes as input a type for numbers: depending on whether numbers are of type integers, rationals or reals, it would indeed perform a different algorithm for addition. On the other hand, parametric polymorphism is the one we find in second order type theory.

The first formalization of parametric polymorphism is due to Reynolds' [Rey83], in connection with his set-theoretic semantics: his idea was to show that, given two different though "related" set assignments η, ζ, the two interpretations of a term M will still be "related". In the following lines we reformulate Reynolds' parametricity in the setting of reducibility candidates: the idea will be to consider arbitrary binary relations over reducibility candidates, and to show that terms $M \in Red_{\forall\alpha\sigma}$ preserve these relations.

In the following, when speaking of λ-terms, we will consider equivalence classes of β-equivalent λ-terms; similarly, when speaking of relations, we will speak of relation over these equivalence classes.

Let us consider arbitrary assignments $\mathcal{N}$ of reducibility candidates to type variables. Parametric reducibility can be redefined in terms of such assignments (as in the case of Prawitz's definition of validity relative to an assignment of regular sets, see section (4.2.2)): for instance, if $FV(\sigma) = \{\alpha_1, \dots, \alpha_n\}$ and, for $1 \leq i \leq n$, $\mathcal{N}(\alpha_i) = \mathcal{C}_i$, then $Red_\sigma[\mathcal{N}]$ is just $Red_\sigma[\dots \mathcal{C}_i/\alpha_i \dots]$.

Let, for each pair of candidates $\mathcal{C}_1, \mathcal{C}_2$, the set $Rel(\mathcal{C}_1, \mathcal{C}_2) := \wp(\mathcal{C}_1 \times \mathcal{C}_2)$ be the set of all binary relations between elements of $\mathcal{C}_1$ and $\mathcal{C}_2$. A *relation assignment* $\mathcal{R}$ will be an assignment, for each pair of reducibility candidates $\mathcal{C}_1, \mathcal{C}_2$, of a relation $R_{\mathcal{C}_1,\mathcal{C}_2} \in Rel(\mathcal{C}_1, \mathcal{C}_2)$. Given two assignments $\mathcal{N}_1, \mathcal{N}_2$, a relation assignment $\mathcal{R}$ and a simple type σ, we can then define the set $\mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma]$, which will be a binary relation over the candidates $Red_\sigma[\mathcal{N}_1]$ and $Red_\sigma[\mathcal{N}_2]$:

- if $\sigma \equiv \alpha_i$, and $\mathcal{N}_p(\alpha_i) = \mathcal{C}_p$ for $1 \leq p \leq 2$, then, for all $M \in \mathcal{C}_1, N \in \mathcal{C}_2$, $M\mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma]N$ holds if and only if $MR_{\mathcal{C}_1,\mathcal{C}_2}N$ holds;
- if $\sigma \equiv \tau \to \rho$, then, for all $M \in Red_\sigma[\mathcal{N}_1]$ and $N \in Red_\sigma[\mathcal{N}_2]$, $M\mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma]N$ holds if and only if, for all $P \in Red_\tau[\mathcal{N}_1]$ and $Q \in Red_\tau[\mathcal{N}_2]$, if $P\mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\tau]Q$ holds, then $(MP)\mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\rho](NQ)$ holds.

We can thus finally define parametricity for closed λ-terms in the reducibility of the *universal closure* of a simple type as follows:

Definition 5.2.1 (parametricity)**.** *Let M be a closed λ-term in $Red_{\forall\overline{\alpha}\sigma}$, where σ is a simple type. Then M is* parametric *if, for all assignments $\mathcal{N}_1, \mathcal{N}_2$ and for all relation assignment $\mathcal{R}$, $M\mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma]M$ holds.*

Let us try to see how parametricity works in a simple case: let M be in $Red_{\forall\alpha(\alpha\to\alpha)}$. This means that, for all reducibility candidate $\mathcal{C}$ and all $N \in \mathcal{C}$, MN is still in $\mathcal{C}$. Now parametricity says that, for all pairs of candidates $\mathcal{C}_1, \mathcal{C}_2$, however we pick up a binary relation r on $\mathcal{C}_1$ and $\mathcal{C}_2$ and two terms P, Q, respectively in $\mathcal{C}_1$ and $\mathcal{C}_2$, such that PrQ holds, then $(MP)r(MQ)$ still holds.

Given two arbitrary candidates $\mathcal{C}_1, \mathcal{C}_2$ and two terms P, Q, respectively in $\mathcal{C}_1$ and $\mathcal{C}_2$, let us take as relation $r = \{(P, Q)\}$; from the fact that $M \in Red_{\forall\alpha(\alpha\to\alpha)}$ it follows that $MP \in \mathcal{C}_1$ and $MQ \in \mathcal{C}_2$; from parametricity it follows then that $(MP)r(MQ)$, i.e. that either $MP = P$ and $MQ = Q$, either $MP = Q$ and $MQ = P$. Hence, in particular, with $\mathcal{C}_1 = \mathcal{C}_2$ and $P = Q$, we have that $MP = P$. Since this holds for all candidates $\mathcal{C}_1, \mathcal{C}_2$ and for all terms $P \in \mathcal{C}_1, Q \in \mathcal{C}_2$, this means that M must send every λ-term into itself: it must thus behave like the identity term $id = \lambda x.x$[2].

Reynolds' *abstraction theorem* (see [Rey83]) states that all simply typed closed λ-terms are parametric. We reprove his result in our formulation based on reducibility candidates:

Theorem 5.2.1 (abstraction theorem)**.** *If M is a closed normal λ-term such that $\vdash M : \sigma$ is derivable in simple type theory, then M is parametric.*

[2]In particular, by applying Böhm's theorem, it follows that $M =_\beta \lambda x.x$.

Proof. We show a more general result: if M is a normal term, with $FV(M) = \{x_1, \dots, x_n\}$ and $(x_1 : \tau_1), \dots, (x_n : \tau_n) \vdash M : \sigma_1 \to \dots \to \sigma_k \to \alpha$ is derivable in simple type theory, then, for any assignments $\mathcal{N}_1, \mathcal{N}_2$ and relation assignment $\mathcal{R}$ and for terms $F_1, G_1, \dots, F_{n+k}, G_{n+k}$ such that $F_i \in Red_{\tau_i}[\mathcal{N}_1]$, $G_i \in Red_{\tau_i}[\mathcal{N}_2]$, for $1 \leq i \leq n$ and $F_i \in Red_{\sigma_i}[\mathcal{N}_1]$ and $G_i \in Red_{\sigma_i}[\mathcal{N}_2]$ for $n+1 \leq i \leq n+k$, one has

$$F_i \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\tau_i] G_i \ (1 \leq i \leq n) \Rightarrow M[F_1/x_1, \dots, F_n/x_n] \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma_1 \to \dots \to \sigma_k] M[G_1/x_1, \dots, G_n/x_n] \tag{5.2.9}$$

We prove this by induction on the typing derivation d of M:

(ax) If d is the derivation

$$\frac{}{(x_1 : \tau_1), \dots, (x_n : \tau_n) \vdash x_i : \tau_i} \tag{5.2.10}$$

then the thesis immediately follows from the assumption $F_i \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\tau_i] G_i$.

(λ) If d is the derivation

$$\frac{\begin{array}{c}\vdots\\ (x_1 : \tau_1), \dots, (x_n : \tau_n), (z : \sigma_1) \vdash P : \sigma_2 \to \dots \to \sigma_k \to \alpha\end{array}}{(x_1 : \tau_1), \dots, (x_n : \tau_n) \vdash \lambda z.P : \sigma_1 \to \dots \to \sigma_k \to \alpha} \tag{5.2.11}$$

then, by induction hypothesis, if $F_i \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\tau_i] G_i$, for $1 \leq i \leq n$ and $F_{n+1} \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma_1] G_{n+1}$, then

$$P[F_1/x_1, \dots, F_n/x_n, F_{n+1}/x_{n+1}] \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma_2 \to \dots \to \sigma_k \to \alpha] P[G_1/x_1, \dots, G_n/x_n, G_{n+1}/x_{n+1}] \tag{5.2.12}$$

The thesis results from the fact that

$$(\lambda z.P[F_1/x_1, \dots, F_n/x_n]) F_{n+1} =_\beta P[F_1/x_1, \dots, F_n/x_n, F_{n+1}/x_{n+1}] \tag{5.2.13}$$

and similarly for G_{n+1}.

$(@)$ If d is the derivation

$$\frac{\begin{array}{c}\vdots\\ (x_1 : \tau_1), \dots, (x_n : \tau_n) \vdash P : \sigma_0 \to \sigma_1 \to \dots \to \sigma_k \to \alpha\end{array} \quad \begin{array}{c}\vdots\\ (x_1 : \tau_1), \dots, (x_n : \tau_n) \vdash Q : \sigma_0\end{array}}{(x_1 : \tau_1), \dots, (x_n : \tau_n) \vdash PQ : \sigma_1 \to \dots \to \sigma_k \to \alpha} \tag{5.2.14}$$

then, by induction hypothesis, if $F_i \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\tau_i] G_i$, for $1 \leq i \leq n$ and $F_0 \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma_0] G_0$, then $\big(P[F_1/x_1, \dots, F_n/x_n]\big) F_0 \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma_1 \to \dots \to \sigma_k \to \alpha] \big(P[G_1/x_1, \dots, G_n/x_n]\big) G_0$ and moreover $Q[F_1/x_1, \dots, F_n/x_n] \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma_0] Q[G_1/x_1, \dots, G_n/x_n]$. The result follows then from the identities

$$\big(P[F_1/x_1, \dots, F_n/x_n]\big) Q[F_1/x_1, \dots, F_n/x_n] = PQ[F_1/x_1, \dots, F_n/x_n] \tag{5.2.15}$$

$$\big(P[G_1/x_1, \dots, G_n/x_n]\big) Q[G_1/x_1, \dots, G_n/x_n] = PQ[G_1/x_1, \dots, G_n/x_n] \tag{5.2.16}$$

□

Reducibility and parametricity In the previous section we showed that Girard's non generic operator J is not reducible. In this section we investigate the relationship between the reducibility of the universal closure of a simple type and the parametricity of its elements.

Remark first that, if we do not take into consideration the second order closure of a simple type, then a reducible term need not be parametric: for instance, the term $U := \lambda x.\delta$, where $\delta = \lambda z.(z)z$, is in $Red_{\alpha\to\alpha}$ (here we consider the non parametric definition of reducibility in section (3.2.2)) but in no way it can be considered parametric. Let P, Q be two closed normal terms and let r be a relation over $\mathcal{SN}$ such that PrQ holds and for no other strongly normalizing term N, NrN holds. Then parametricity would require that, since PrQ, also $(UP)r(UQ)$ holds, but $UP = \delta = UQ$, so this is impossible.

The results proved below show then that parametricity is obtained as soon as one consider reducibility with respect to the universal closure of simple types. This result is a first step towards the $\mathbf{\Pi}^1$-completeness theorem of the following subsection: in order to show that a reducible term is typable, we will need to pass from parametricity to the dinatural interpretation of simple type theory (see below), which embodies the diagrammatic idea of genericity.

Since the idea of our proof is to show that we can, in a sense, code parametricity in reducibility candidates, we start with a lemma which allows the definition of *ad hoc* candidates:

Lemma 5.2.1. *Let s be a set of closed normal λ-terms and let $\mathcal{C}_s$ be the smallest reducibility candidate containing s. Then, if $M \in \mathcal{C}_s$ is closed and normal, $M \in s$.*

Proof. Let us suppose that $M \in \mathcal{C}_s - s$ is closed and normal. Let m be the set containing all the λ-terms which are β-equivalent to M. Remark that all the terms in M are closed. We will show that the set $m' := \mathcal{C}_s - m$ is a reducibility candidate containing s, contradicting the hypothesis that $\mathcal{C}_s$ is the smallest such candidate. We show then that m' satisfies the properties $\mathbf{R1} - \mathbf{3}$.

m' satisfies **R1** since it is contained in $\mathcal{C}_s$. Moreover, if $N \in m'$ and $N \to N'$, since N is not β-equivalent to M, N' is neither; moreover, by **R2** applied to $\mathcal{C}_s$ it follows that $N' \in \mathcal{C}_s - m$, i.e. $N' \in m'$. Finally, let N be a simple term such that, for all N' such that $N \to_1 N'$, $N' \in m'$; if N is normal, then it must be an open term which belongs to $\mathcal{C}_s$ by **R3**; since N is open, $N \notin m$ and thus $N \in m'$. If N is not normal, then all its immediate reducts are in $\mathcal{C}_s$, hence $N \in \mathcal{C}_s$; moreover, since the N' are in m', they are not β-equivalent to M and neither N is, so that $N \in m'$.

□

We can now prove the parametricity theorem: in the proof we will use the abstraction theorem (5.2.1) and lemma (5.2.1) to define *ad hoc* candidates.

Theorem 5.2.2 (parametricity)**.** *Let M be a closed λ-term in $Red_{\forall\overline{\alpha}\sigma}$, where σ is a simple type. Then M is parametric.*

Proof. Let $\sigma \equiv \sigma_1 \to \cdots \to \sigma_k \to \alpha$ and let us suppose that M is not parametric: this means that there exist two assignments $\mathcal{N}_1, \mathcal{N}_2$, a relation assignment $\mathcal{R}$ and terms $F_1, G_1, \dots, F_k, G_k$ such that $F_i \in Red_{\sigma_i}[\mathcal{N}_1], G_i \in Red_{\sigma_i}[\mathcal{N}_2]$ and $F_i \mathcal{R}_{\mathcal{C}_1,\mathcal{C}_2}[\sigma_i] G_i$, for $1 \leq i \leq k$ and $(M)F_1 \dots F_k \mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\alpha](M)G_1 \dots G_k$ does not hold.

By Reynold's abstraction theorem, it follows that, for all terms P such that

$$(x_1 : \sigma_1), \dots, (x_k : \sigma_k) \vdash P : \alpha \tag{5.2.17}$$

is derivable in simple type theory, either $(M)F_1 \dots F_k \neq P[F_1/x_1, \dots, F_k/x_k]$, either $(M)G_1 \dots G_k \neq P[G_1/x_1, \dots, G_k/x_k]$.

Let s_1, s_2 be the following sets:

$$s_1 := \{P[F_1/x_1, \ldots, F_k/x_k] | (x_1 : \sigma_1), \ldots, (x_k : \sigma_k) \vdash P : \alpha\} \tag{5.2.18}$$
$$s_2 := \{P[G_1/x_1, \ldots, G_k/x_k] | (x_1 : \sigma_1), \ldots, (x_k : \sigma_k) \vdash P : \alpha\} \tag{5.2.19}$$

and let $\mathcal{M}_1, \mathcal{M}_2$ be assignments such that $\mathcal{M}_1(\alpha) = \mathcal{C}_{s_1}$ and $\mathcal{M}_2(\alpha) = \mathcal{C}_{s_2}$. Remark that we have to exploit the reducibility theorem for simple type theory in order to know that the terms $P[F_1/x_1, \ldots, F_k/x_k]$ and $P[G_1/x_1, \ldots, G_k/x_k]$ have a normal form.

Let us show, by induction on σ_i, that $F_i \in Red_{\sigma_i}[\mathcal{M}_1]$ and $G_i \in Red_{\sigma_i}[\mathcal{M}_2]$. If $\sigma_i \equiv \alpha$, then $x_i[F_1/x_1, \ldots, F_k/x_k] = F_i \in \mathcal{M}_1(\alpha)$ and $x_i[G_1/x_1, \ldots, G_k/x_k] = G_i \in \mathcal{M}_2(\alpha)$. If $\sigma_i = \rho_1 \to \cdots \to \rho_r \to \alpha$, then let $N_j \in Red_{\rho j}[\mathcal{M}_1]$, for $1 \leq j \leq r$; then $(F_i)N_1 \ldots N_r = (x_i)P_1 \ldots P_r[F_1/x_1, \ldots, F_k/x_k] \in Red_\alpha[\mathcal{M}_1]$ by induction hypothesis applied to the terms N_j. One argues similarly to show $G_i \in Red_{\sigma_i}[\mathcal{M}_2]$.

Now, from the definition of reducibility, it follows that $M \in Red_\sigma[\mathcal{M}_1]$ and $M \in Red_\sigma[\mathcal{M}_2]$, and this implies both $(M)F_1 \ldots F_k \in \mathcal{C}_{s_1}$ and $(M)G_1 \ldots G_k \in \mathcal{C}_{s_2}$, contradicting the hypothesis. □

5.2.2 The dinatural interpretation: new equations for polymorphic terms

We introduce the dinatural interpretation of simple type theory ([BFSS90, GSS92]), which generalizes diagrams like (5.2.2). The interest of this interpretation is that, from the genericity expressed by such diagrams we will be able to recover syntactic properties of typed λ-terms. In the next section, from a syntactic formulation of the dinaturality criterion, we will derive an equational characterization of reducible terms.

Let us restart from our example above, from the viewpoint of type theory: we consider a term M of type $\forall\alpha(\alpha \to \alpha \times \alpha)$. The polymorphic nature of M means that, if we interpret types as object of a *ccc* category $\mathcal{C}$ and terms as morphisms between those objects, M corresponds to a family μ_A of morphisms $A \to A \times A$, for any object A.

The genericity condition is given then by the following diagram, for every objects A, B of $\mathcal{C}$ and morphism $f : A \to B$:

$$\begin{tikzcd} A \arrow[r, "\mu_A"] \arrow[d, "f"] & A \times A \arrow[d, "f \times f"] \\ B \arrow[r, "\mu_B"'] & B \times B \end{tikzcd} \tag{5.2.20}$$

which corresponds to the equation

$$\mu_A \circ (f \times f) = f \circ \mu_B \tag{5.2.21}$$

One of the main results of the interpretation we are going to sketch is that the equation (5.2.21) implies a syntactic equation concerning M: the functional equality expressed by (5.2.21) implies indeed the β-equivalence of the following terms (under the typing assumptions $(f : \sigma \to \tau), (x : \sigma)$, for arbitrary types σ, τ):

$$\lambda u.(u)f(MxP_1)f(MxP_2) =_\beta M(fx) \tag{5.2.22}$$

where $\lambda u.(u)N_1N_2$ is the usual construct for pairs, $P_1 = \lambda x.\lambda y.x$, $P_2 = \lambda x.\lambda y.y$ are the usual projections. It can be easily verified that the (only) choice for M is the term $\lambda x.\lambda u.(u)xx$ (which corresponds to the diagonal family $\delta_s(x) = (x, x)$.

This result is quite interesting: from a semantical property expressing the genericity of an arbitrary polymorphic term of type $\alpha \to \alpha \times \alpha$ we recovered an equation which characterizes such a (unique) term. As Wadler comments in a famous paper entitled “Theorems for free!”:

> From the type of a polymorphic function we can derive a theorem that it satisfies. Every function of the same type satisfies the same theorem. This provides a free source of useful theorems, courtesy of Reynolds’ abstraction theorem for the polymorphic lambda calculus. [Wad89]

Let us fix a *ccc* category $\mathcal{C}$. The usual (let us call it “first grade”) category-theoretic interpretation of simple type theory in *ccc* categories associates types σ with objects A_σ of the category and closed terms M of type $\sigma \to \tau$ with morphisms $f_M : A_\sigma \to A_\tau$.

The idea of the so-called *dinatural interpretation* (or “second grade” interpretation) is to interpret types as certain *functors* over the category $\mathcal{C}$ and terms as *dinatural transformations* between these functors. Remark that the “first grade” interpretation is still an ingredient of the dinatural one: closed terms of simple type theory correspond indeed to morphisms in $\mathcal{C}$.

Let us first recall that a *natural transformation* $\eta_A : \mathbf{F} \to \mathbf{G}$ between two (covariant) functors $\mathbf{F}, \mathbf{G} : \mathcal{C} \to \mathcal{C}$ is a family of maps such that, for each objects A, B of $\mathcal{C}$ and each morphism $f : A \to B$, the following diagram commutes:

$$\begin{tikzcd}
\mathbf{F}A \arrow[r, "\eta_A"] \arrow[d, "\mathbf{F}f"] & \mathbf{G}A \arrow[d, "\mathbf{G}f"] \\
\mathbf{F}B \arrow[r, "\eta_B"'] & \mathbf{G}B
\end{tikzcd} \tag{5.2.23}$$

For instance, we can interpret the types α and $\alpha \times \alpha$ as certain (covariant) functors $\mathbf{F}A$, $\mathbf{G}A = \mathbf{F}A \times \mathbf{F}A$: the intuition is that, if N is a term of type $\sigma \to \tau$, then any term of type σ or $\sigma \times \sigma$ can be transformed, using N, into a term, respectively, of type τ or $\tau \times \tau$. Then, our term M of type $\alpha \to \alpha \times \alpha$ can be interpreted as a natural transformation μ_A from $\mathbf{F}A$ to $\mathbf{G}A$: the naturality condition corresponds indeed to diagram (5.2.20).

The problem with natural transformations arises with the interpretation of implication types: seen as a functor, an implication $\alpha \to \beta$ is covariant on the variable β and controvariant on the variable α (such functors are usually called *multivariant*). The notion of *dinatural transformation* is then the extension of the idea of naturality to the case of multivariant functors: a dinatural transformation $\eta_A : \mathbf{F} \to \mathbf{G}$ between two multivariant functors $\mathbf{F}AB, \mathbf{G}AB$ (where the variable A stands for the controvariant part of the functor, and the variable B for its covariant part), is a family of maps such that, for each object A, B and each morphism $f : A \to B$, the following diagram commutes:

$$\begin{tikzcd}
 & \mathbf{F}AA \arrow[r, "\eta_A"] & \mathbf{G}AA \arrow[dr, "\mathbf{G}Af"] & \\
\mathbf{F}BA \arrow[ur, "\mathbf{F}fA"] \arrow[dr, "\mathbf{F}Bf"'] & & & \mathbf{G}AB \\
 & \mathbf{F}BB \arrow[r, "\eta_B"'] & \mathbf{G}BB \arrow[ur, "\mathbf{G}fB"'] &
\end{tikzcd} \tag{5.2.24}$$

In the case of a functor with no contravariant variable, the definition above reduces immediately to the one of natural transformations.

A surprising fact about dinatural transformations is that, unlike natural transformations, they cannot be composed: in the diagram below, which virtually interprets a cut between two proofs, represented by the dinatural transformations η_A, ζ_A, whereas the two inner hexagones commute, the outer hexagon need not commute:

$$\begin{tikzcd}
& \mathbf{F}AA \arrow[r, "\eta_A"] & \mathbf{G}AA \arrow[r, "\zeta_A"] \arrow[dr, "\mathbf{G}Af"] & \mathbf{H}AA \arrow[dr, "\mathbf{H}Af"] & \\
\mathbf{F}BA \arrow[ur, "\mathbf{F}fA"] \arrow[dr, "\mathbf{F}Bf"'] & \mathbf{G}BA \arrow[ur, "\mathbf{G}fA"] \arrow[dr, "\mathbf{G}Bf"'] & & \mathbf{G}AB & \mathbf{H}AB \\
& \mathbf{F}BB \arrow[r, "\eta_B"'] & \mathbf{G}BB \arrow[r, "\zeta_B"'] \arrow[ur, "\mathbf{G}fB"] & \mathbf{H}BB \arrow[ur, "\mathbf{H}fB"'] &
\end{tikzcd} \tag{5.2.25}$$

As a result, the interpretation only works for normal terms, since the cut-rule cannot be interpreted. This implies that the result that we will get from the Π^1-completeness theorem (5.2.4), in particular the structural conditions on the form of reducible terms (corollary (5.2.1)), will be limited to normal terms.

We won't state in detail the interpretation of closed normal λ-terms in the dinatural calculus. The reader will find a detailed description in [GSS92], where the interpretation of natural deduction derivations is considered too. In this paragraph we will limit ourselves to present some interesting examples, in which from the dinaturality hypothesis we will derive equations characterizing polymorphic typed λ-terms. In the next section we will provide a syntactic description of the equations implied by dinaturality and we will prove the validity of such equations over λ-terms as a consequence of the parametricity theorem (5.2.2).

The first example we consider concerns the type $\mathbf{N}$: let M be a normal term of type $\mathbf{N}$. The interpretation of M will be then a dinatural transformation ν_A from the functor $\mathbf{F}BA = A^B$ to itself. The dinaturality of ν_A corresponds then, for all object A, B and morphisms $f : A \to B, g : B \to A$, to the commutation of the diagram below:

$$\begin{tikzcd}
& A^A \arrow[r, "\nu_A"] & A^A \arrow[dr, "f^B"] & \\
A^B \arrow[ur, "A^f"] \arrow[dr, "f^B"'] & & & B^A \\
& B^B \arrow[r, "\nu_B"'] & B^B \arrow[ur, "B^f"'] &
\end{tikzcd} \tag{5.2.26}$$

from which we can derive the equation below:

$$\nu_A(f \circ g) \circ f = f \circ \nu_B(g \circ f) \tag{5.2.27}$$

The only solutions for ν_A are the *iterators*, i.e. $\nu_A(f) = \underbrace{f \circ f \circ \cdots \circ f}_{k \text{ times}}$, for a certain $k \in \mathbb{N}$. Indeed, in this case equation (5.2.27) reduces to the valid equation below:

$$(f \circ g) \circ (f \circ g) \circ \cdots \circ f = f \circ (g \circ f) \circ \cdots \circ (g \circ f) \tag{5.2.28}$$

Syntactically, equation (5.2.27) corresponds to the following equation for a closed normal term M of type $\mathbf{N}$ (under the assumptions $f : \sigma \to \tau$, $g : \tau \to \sigma$, for arbitrary types σ, τ):

$$f((M)\lambda u.g(fu)x) =_\beta ((M)\lambda u.f(gu))(fx) \tag{5.2.29}$$

The (unique) solutions to equation (5.2.29) are precisely the Church numerals $\lambda f.\lambda x.f^n x$ (which correspond to the iterators) and the identity id (see corollary (5.2.1)).

A second example concerns fixed points: suppose M is a closed normal term of type $\forall\alpha((\alpha \to \alpha) \to \alpha)$. The interpretation of M is then a dinatural transformation ϕ_A from the functor $\mathbf{F}BA = A^B$ to the identity functor $\mathbf{I}BA = A$, i.e., for any object A, B and morphism $f : A \to B$, the following diagram commutes:

$$
\begin{tikzcd}
 & A^A \arrow[r, "\phi_A"] & A \arrow[dr, "f"] & \\
A^B \arrow[ur, "A^f"] \arrow[dr, "f^B"'] & & & B \\
 & B^B \arrow[r, "\phi_B"'] & B \arrow[ur, "B"'] &
\end{tikzcd}
\tag{5.2.30}
$$

Then, for all $g : B \to A$, one has the equation below:

$$\phi_A(f \circ g) \circ f = \phi_B(g \circ f) \tag{5.2.31}$$

Now, if we choose $B = A$ and $g = id_A$, equation (5.2.31) reduces to

$$\phi_A(f) \circ f = \phi_A(f) \tag{5.2.32}$$

i.e., in set-theoretic notation, $f(\phi_A(f)) = \phi_A(f)$, which means that ϕ_A is a uniform fixed point. Syntactically, this means that, for any type σ, if f is declared of type $\sigma \to \sigma$ then the following equation holds:

$$f(Mf) =_\beta Mf \tag{5.2.33}$$

and thus M must be a fixed point combinator at each type. In particular, since System F is strongly normalizable, we can thus deduce that there is no closed normal term of type $\forall\alpha((\alpha \to \alpha) \to \alpha)$.

5.2.3 A completeness theorem

In this subsection we will use the dinatural interpretation to derive the $\mathbf{\Pi}^1$-completeness theorem: this theorem says that, if M is a closed normal λ-term in the reducibility $Red_{\forall\overline{\alpha}\sigma}$ of a universally closed simple type, then $\vdash M : \sigma$ is derivable in simple type theory.

The theorem is obviously false if we do not take into account the second order universal closure of the simple type σ: in chapter (3) (section (3.2.3)) one can find examples of reducible though not typable terms. Thus the passage through impredicative quantification turns out to be fundamental in order to have a *finite enough* description of λ-terms in terms of their behavior. This rather counter-intuitive aspect will be briefly discussed in the next section.

The $\mathbf{\Pi}^1$-completeness theorem can be seen as a counterpart to the Π^1-completeness theorem of chapter (2). In particular, in virtue of the $\mathbf{\Sigma}^1$-incompleteness theorem (3.2.2) it provides an upper bound for the completeness (or faithfulness) theorem (2.3.1) of the behavioral interpretation of typing. Moreover, the theorem has several corollaries which allow to retrieve, for simple type theories, equivalent of the *canonicity condition* which hold for valid derivations in natural deduction in the sense of Prawitz's validity (section (4.2.2)). Again, the interest of the result lies in the fact that, in order to retrieve the canonicity conditions, one has to pass through an impredicative interpretation of proofs.

Analogous completeness results can be found in the literature: for instance, in [Hin83], it is proved that simple type theory is complete with respect to several variants of denotational semantics. In [Coq05] the reader will find a brief historical reconstruction of the subject, along with a reformulation of Hindley's result under the form: if a closed normal λ-term M is in the (impredicative) intersection of all the interpretations of the simple type σ, then $\vdash M : \sigma$ is derivable in simple type theory. As the author remarks,

> This shows that the a priori impredicative intersection $\bigcap_{X:\Lambda\to\mathcal{H}} T(\alpha = X)(t)$ has a predicative description. [Coq05]

In particular, Coquand exploits this result to derive some results (in the line of corollary (5.2.1)) about terms in the $\mathbf{\Pi}^1$-fragment of System F.

The argument that follows, however, applies directly to the reducibility interpretation of System F and has the following structure: we first develop a syntactical formulation of the dinatural interpretation; in particular we describe the equations forced by the dinaturality condition in full generality by means of certain typed λ-terms H_σ, K_σ. Next we use these operators to show that Reynolds' parametricity implies the (syntactical consequences of the) dinaturality condition. In particular, it will be shown that the terms H_σ, K_σ in a sense characterize the "degrees of freedom" left by the parametricity condition. As a consequence of the parametricity theorem (5.2.2), this result implies that a closed normal λ-term in the reducibility $Red_{\forall\overline{\alpha}(\sigma\to\tau)}$ must satisfy the syntactic equations expressing dinaturality. Finally, we will prove our main theorem (5.2.4) by defining an inductive argument which explicitly retrieves the typing of M from the fact that M satisfies such equations.

Coding dinaturality in λ-calculus We start by a lemma which illustrates the general idea of the $\mathbf{\Pi}^1$-completeness theorem: the equations for a closed normal λ-term M which come from the dinaturality condition are of the form

$$f((M)P_1 \dots P_k) = (M)P_1' \dots P_k' \tag{5.2.34}$$

where f is a variable declared of type $\alpha \to \beta$. Then, the validity of (5.2.34) forces very restrictive conditions of the form of M:

Lemma 5.2.2. *Let, for all $1 \leq i \leq k$, P_i, P_i' be λ-terms of the same arity k_i. Then, if M is a closed normal λ-term which satisfies an equation of the form*

$$f((M)P_1 \dots P_k) = (M)P_1' \dots P_k' \tag{5.2.35}$$

then M is of the form

$$M = \lambda x_1. \dots .\lambda x_h.(x_i)Q_1 \dots Q_p \tag{5.2.36}$$

for certain terms $Q_1, \dots, Q_p$, where $h \leq k$, $1 \leq i \leq h$, and $p \geq k_i - (k-h)$.

Proof. Let us first show that $h \leq k$: suppose indeed $M = \lambda x_1. \dots .\lambda x_k.\lambda y.M'$, then the equation is not satisfied, since one obtains

$$f((M)P_1 \dots P_k) \to f(\lambda y.M'[P_1/x_1, \dots, P_k/x_k]) \neq \lambda y.M'[P_1'/x_1, \dots, P_k'/x_k] \leftarrow (M)P_1' \dots P_k' \tag{5.2.37}$$

Let us show now that $p \geq k_i - (k-h)$: if $p < k_i - (k-h)$, then the term $(M)P_1 \dots P_k$ reduces to

$$(P_i)Q_1' \dots Q_p' P_{h+1} \dots P_k \to_1 \lambda z_{(k_i-(k-h))-p}. \dots .\lambda z_{k_i}.W \tag{5.2.38}$$

for a certain term W; then equation (5.2.35) is not satisfied:

$$f((M)P_1 \dots P_k) \to f(\lambda z_{(k_i-(k-h))-p}.\dots.\lambda z_{k_i}.W) \neq \lambda z_{(k_i-(k-h))-p}.\dots.\lambda z_{k_i}.W' \leftarrow (M)P'_1 \dots P'_k \tag{5.2.39}$$

□

We pass now to the definition of typed terms H_σ, K_σ by which we will be able to code dinaturality in λ-calculus in full generality.

For any simple type σ, whose free variables are $\gamma_1, \dots, \gamma_l$, let σ^α_β (resp. σ^β_α) denote the result of replacing, in σ, all positive occurrences of γ_u, for $1 \leq u \leq l$, by the variable α_u (resp. β_u), and all negative occurrences of γ_u by the variable β_u (resp. α_u). Let then $\sigma_\alpha, \sigma_\beta$ denote, respectively, $\sigma[\alpha_1/\gamma_1, \dots, \alpha_l/\gamma_l]$ and $\sigma[\beta_1/\gamma_1, \dots, \beta_l/\gamma_l]$.

The idea of the definition below is the following: let us assume that M is in the reducibility of $\forall\overline{\alpha}(\sigma \to \tau)$. From the dinaturality of the associated family μ_A we wish to derive an equation for M. In order to build such an equation we need to define four typed terms, constructed with the help of l distinct variables f_u declared of type $\alpha_u \to \beta_u$, for $1 \leq u \leq l$:

$$\begin{array}{ll} H_\sigma : \sigma_\alpha \to \sigma^\beta_\alpha & K_\sigma : \sigma^\alpha_\beta \to \sigma_\alpha \\ J_\sigma : \sigma^\alpha_\beta \to \sigma_\beta & I_\sigma : \sigma_\beta \to \sigma^\beta_\alpha \end{array} \tag{5.2.40}$$

We define simultaneously the operators $H^\alpha_\beta, K^\beta_\alpha$ by induction over σ:

- if $\sigma \equiv \alpha_u$, then $H_\sigma := \lambda x.f_u x : \alpha_u \to \beta_u$ and $K_\sigma := \lambda x.x : \alpha_u \to \alpha_u$;
- if $\sigma = \rho_1 \to \dots \to \rho_k \to \alpha_u$, then $H_\sigma : \sigma_\alpha \to \sigma^\beta_\alpha$ is

$$H_\sigma := \lambda g.\lambda h_1.\dots.\lambda h_k.f_u\big(g(K_{\rho_1}h_1)(K_{\rho_2}h_2)\dots(K_{\rho_k}h_k)\big) \tag{5.2.41}$$

  and $K_\sigma : \sigma^\alpha_\beta \to \sigma_\alpha$ is

$$K_\sigma := \lambda g.\lambda h_1.\dots.\lambda h_k.g(H_{\rho_1}h_1)(H_{\rho_2}h_2)\dots(H_{\rho_k}h_k) \tag{5.2.42}$$

A crucial property of the terms H_σ, K_σ is the following:

Lemma 5.2.3. *For every simple type σ, τ the equation below*

$$H_\sigma(K_\tau g) = K_\sigma(H_\tau g) \tag{5.2.43}$$

holds if and only if $\sigma \equiv \tau$.

Proof. We argue by induction on σ: if $\sigma \equiv \tau \equiv \alpha_u$, then $H_{\alpha_u}(K_{\alpha_u}g) = f_u g = K_{\alpha_u}(H_{\alpha_u}g)$; conversely, if $\sigma \not\equiv \tau \equiv \alpha_u$, then

$$\begin{array}{r} H_\sigma(K_\alpha g) = H_\sigma g = \lambda h_1.\dots.\lambda h_k.f_v\big(g(K_{\rho 1}h_k)\dots(K_{\rho k}h_k)\big) \neq \\ \lambda h_1.\dots.\lambda h_k.(f_u g)(H_{\sigma_1}h_1)\dots(H_{\rho k}h_k) = K_\rho(f_u g) \end{array} \tag{5.2.44}$$

which is false for all $1 \leq u, v \leq l$.

Similarly, one has

$$\begin{array}{r} H_{\alpha_u}(K_\sigma g) = f_u(K_\sigma g) = f_u\big(\lambda h_1.\dots.\lambda h_k.g(K_{\rho_1}h_1)\dots(K_{\rho_k}h_k)\big) \neq \\ \lambda h_1.\dots.\lambda h_k.f_v\big(g(H_{\rho_1}k_1)\dots(H_{\rho_k}h_k)\big) = H_\sigma g = K_\alpha(H_\sigma g) \end{array} \tag{5.2.45}$$

for all $1 \leq u, v \leq l$, since $k = 0$ only if σ is a variable.

For the induction step, remark that, if $\sigma \equiv \rho_1 \to \cdots \to \rho_k \to \alpha_u$, then

$$H_\sigma(K_\sigma g) \to_1 H_\sigma\big(\lambda h_1.\dots.\lambda h_k.g(H_{\rho_1}h_1)\dots(H_{\rho_k}h_k)\big) \to_1 f_u\big(g(H_{\rho_1}(K_{\rho_1}h_1))\dots(H_{\rho_k}(K_{\rho_k}h_k))\big) \tag{5.2.46}$$

and

$$K_\sigma(H_\sigma g) \to_1 K_\sigma\big(\lambda h_1.\dots.\lambda h_k.f_u\big(g(K_{\rho_1}h_1)\dots(K_{\rho_k}h_k)\big)\big) \to_1 f_u\big(g(K_{\rho_1}(H_{\rho_1}h_1))\dots(K_{\rho_k}(H_{\rho_k}h_k)\big) \tag{5.2.47}$$

which implies that equation (5.2.43) holds if and only if the equations below all hold

$$H_{\rho_i}(K_{\rho_i}h_i) = K_{\rho_i}(H_{\rho_i}h_i) \tag{5.2.48}$$

for $1 \leq i \leq k$, so one can apply the induction hypothesis.

For the converse direction, let $\sigma \equiv \sigma_1 \to \cdots \to \sigma_k \to \alpha_u$ and $\tau = \tau_1 \to \cdots \to \tau_{k'} \to \alpha_v$. Suppose $k' = k + d$, $H_\sigma(K_\tau g)$ reduces to

$$\lambda h_1.\dots.\lambda h_k.f_u\big(\lambda h_{k+1}.\dots.\lambda h_{k'}.g(K_{\sigma_1}(H_{\tau 1}h_1))\dots(K_{\sigma_k}(H_{\tau_k}h_k))(H_{\tau_{k+1}}h_{k+1})\dots(H_{\tau_{k'}}h_{k'})\big) \tag{5.2.49}$$

and $K_\sigma(H_\tau g)$ reduces to

$$\lambda h_1.\dots.\lambda h_k.\lambda h_{k+1}.\dots.\lambda h_{k'}.f_v\big(g(H_{\sigma_1}(K_{\tau_1}h_1))\dots(H_{\sigma_k}(K_{\tau_k}h_k))(K_{\tau_{k+1}}h_{k+1})\dots(K_{\tau_{k'}}h_{k'})\big) \tag{5.2.50}$$

Now the two terms are equal only if $d = 0$, $u = v$ and, for all $1 \leq i \leq k$, $\sigma_i \equiv \tau_i$ (by induction hypothesis), i.e only if $\sigma \equiv \tau$. A similar argument can be made for the hypothesis $k = k' + d$.

□

By a similar construction, we can define simultaneously the terms J_σ, I_σ; in particular, it turns out that, as pure λ-terms, $J_\sigma = H_\sigma$ and $I_\sigma = K_\sigma$.

The diagram below will help the reader think of a term $M \in Red_{\forall\overline{\alpha}(\sigma\to\tau)}$ in terms of dinatural transformations:

$$\begin{tikzcd}
& \sigma_\alpha \arrow[r, "M_\alpha"] & \tau_\alpha \arrow[dr, "H_\tau"] & \\
\sigma^\alpha_\beta \arrow[ur, "K_\sigma"] \arrow[dr, "J_\sigma"'] & & & \tau^\beta_\alpha \\
& \sigma_\beta \arrow[r, "M_\beta"'] & \tau_\beta \arrow[ur, "I_\tau"'] &
\end{tikzcd} \tag{5.2.51}$$

The equation associated to the diagram above, given $g : \sigma^\alpha_\beta$ is then

$$H_\tau(M(K_\sigma g)) = I_\tau(M(J_\sigma g)) \tag{5.2.52}$$

which, from an untyped perspective, is just the equation

$$H_\tau(M(K_\sigma g)) = K_\tau(M(H_\sigma g)) \tag{5.2.53}$$

We can now refine lemma (5.2.2) as follows:

Lemma 5.2.4. *Let M be a closed normal λ-term satisfying:*

$$f_u\big((M)(K_{\sigma_1}L_1)\dots(K_{\sigma_n}L_n)\big) = (M)(H_{\sigma_1}L_1)\dots(H_{\sigma_n}L_n) \tag{5.2.54}$$

then M has the form

$$M = \lambda x_1.\dots.\lambda x_h.(x_i)Q_1\dots Q_p \tag{5.2.55}$$

for certain terms $Q_1, \dots, Q_p$, where $h \leq n$, $1 \leq i \leq h$, and $p = k_i - (n - h)$.

Proof. Let d be $k_i - (n-h)$. We just have to prove that $p = d$. Let us suppose $p > d$: the term $f_u((M)(K_{\sigma_1} L_1)\dots(K_{\sigma_n} L_2))$ reduces then in turn to:

$$f_u\big((K_{\sigma_i} L_i) W_1 \dots W_{p+(n-h)}\big) \tag{5.2.56}$$

for certain terms $W_1, \dots, W_{p+(n-h)}$, and to

$$f_u\big(L_i(H_{\rho_1} W_1)(H_{\rho_2} W_2)\dots(H_{\rho_{k_1}} W_{k_i}) W_{k_i+1} W_d\big) \tag{5.2.57}$$

where $\sigma_i = \rho_1 \to \dots \to \rho_{k_i} \to \alpha$.

The term $(M)(H_{\sigma_1} L_1)\dots(H_{\sigma_n} L_n)$ reduces in turn to

$$f_u\big((H_{\sigma_i} L_i) W'_1 \dots W'_{p+(n-h)}\big) \tag{5.2.58}$$

for certain terms $W'_1, \dots, W'_{p+(n-h)}$, and to

$$f_u\big(L_i(K_{\rho_1} W'_1)(K_{\rho_2} W_2)\dots(K_{\rho_{k_1}} W_{k_i})\big) W_{k_i+1} W_d \tag{5.2.59}$$

violating equation (5.2.35).

$\square$

Parametricity implies dinaturality We will show that a parametric λ-term in $Red_{\forall\overline{\alpha}(\sigma\to\tau)}$ must satisfy equation (5.2.53). We first prove the following useful lemma, which states that the terms H_σ, K_σ code the "degrees of freedom" left by parametricity: in particular, it says that Reynolds' criterion is blind to the transformations operated by these terms.

Lemma 5.2.5. *Let σ be a simple type with $FV(\sigma) = \{\gamma_1, \dots, \gamma_l\}$ and $f_1, \dots, f_u$ distinct variables. Let $\mathcal{N}_1, \mathcal{N}_2$ be two assignments and $\mathcal{R}^f$ a relation assignment such that, for all $1 \le u \le l$, if $\mathcal{N}_1(\gamma_u) = \mathcal{C}_u$ and $\mathcal{N}_2(\gamma_u) = \mathcal{D}_u$, then $\mathcal{R}^f_{\mathcal{C}_u,\mathcal{D}_u}$ is the relation r^f_u on $\mathcal{C}_u$ and $\mathcal{D}_u$ given by*

$$M r^f_u N \text{ if and only if } f_u(M) = N \tag{5.2.60}$$

Then, if $MR^f_{\mathcal{N}_1,\mathcal{N}_2}[\sigma]N$ holds, $H_\sigma M = K_\sigma N$ holds and moreover

$$(K_\sigma g) R^f_{\mathcal{N}_1,\mathcal{N}_2}[\sigma](H_\sigma g) \tag{5.2.61}$$

Proof. We show the two theses simultaneously by induction over σ.

- $\sigma \equiv \alpha$: if $MR^f_{\mathcal{N}_1,\mathcal{N}_2}[\alpha_u]N$, $H_{\alpha_u} M = (\lambda x.f_u x)M = f_u M = N = (\lambda x.x)N = K_{\alpha_u} N$. Moreover $f_u(K_\alpha g) = f_u g = H_\alpha g$.
- $\sigma \equiv \sigma_1 \to \dots \to \sigma_k \to \alpha_u$: let us first suppose that $MR^f_{\mathcal{N}_1,\mathcal{N}_2}[\sigma]N$; this means that, if $F_i R^f_{\mathcal{N}_1,\mathcal{N}_2}[\sigma_i] G_i$, then $f_u((M)F_1 \dots F_k) = (N) G_1 \dots G_k$. By induction hypothesis, we know that $(K_{\sigma_i} h_i) R^f_{\mathcal{N}_1,\mathcal{N}_2}[\sigma_i](H_{\sigma_i} h_i)$, hence

  $$f_u\big((M)(K_{\sigma_1} h_1)\dots(K_{\sigma_k} h_k)\big) = (N)(H_{\sigma_1} h_1)\dots(H_{\sigma_k} h_k) \tag{5.2.62}$$

  and thus

  $$\lambda h_1.\dots.\lambda h_k.f_u\big(M(K_{\sigma_1} h_1)\dots(K_{\sigma_k} h_k)\big) = \lambda h_1.\dots.\lambda h_k.(N)(H_{\sigma_1} h_1)\dots(H_{\sigma_k} h_k) \tag{5.2.63}$$

  that is $H_\sigma M = K_\sigma N$.

It remains to show that $(K_\sigma g)R^f_{\mathcal{N}_1,\mathcal{N}_2}[\sigma](H_\sigma g)$, that is, that $f_u\big((K_\sigma g)F_1 \dots F_k\big) = (H_\sigma g)G_1 \dots G_k$, under the assumption that $F_i R^f_{\mathcal{N}_1,\mathcal{N}_2}[\sigma_i]G_i$; by induction hypothesis the assumption implies that $H_{\sigma_i}F_i = K_{\sigma_i}G_i$, and then

$$f_u\big((K_\sigma g)F_1 \dots F_k\big) = f_u\big(g(H_{\sigma_1}F_1)\dots(H_{\sigma_k}F_k)\big) \tag{5.2.64}$$

is exactly

$$(H_\sigma g)G_1 \dots G_k = f_u\big(g(K_{\sigma_1}G_1)\dots(K_{\sigma_k}G_k)\big) \tag{5.2.65}$$

□

Now that we know that the terms H_σ, K_σ are well correlated with Reynolds' parametricity, we can state the main result of this paragraph:

Theorem 5.2.3 (parametricity implies dinaturality)**.** *Let M be in $Red_{\forall\overline{\alpha}(\sigma\to\tau)}$ where σ is a simple type. If for all assignment $\mathcal{N}_1, \mathcal{N}_2$ and relation assignment $\mathcal{R}$, $M\mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma \to \tau]M$, then $H_\tau(M(K_\sigma g)) = K_\tau(M(H_\sigma g))$.*

Proof. Let us write $\sigma \to \tau$ as $\tau_0 \to \tau_1 \to \dots \tau_k \to \alpha_u$; $M\mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\sigma \to \tau]M$ means that, for all $F_0, G_0, \dots, F_k, G_k$ such that $F_i\mathcal{R}_{\mathcal{N}_1,\mathcal{N}_2}[\tau_i]G_i$, for $0 \le i \le k$, $f_u((M)F_0 \dots F_k) = (M)G_0 \dots G_k$.

In particular, by choosing the assignments $\mathcal{N}_1$ and $\mathcal{N}_2$ such that $\mathcal{N}_1(\alpha) = \mathcal{C}_1$ and $\mathcal{N}_2(\alpha) = \mathcal{C}_2$, and the relation assignment $R^f_{\mathcal{N}_1,\mathcal{N}_2}$, by lemma (5.2.5) we have that $(K_{\tau_i}h_i)\mathcal{R}^f_{\mathcal{N}_1,\mathcal{N}_2}[\tau_i](H_{\tau_i}h_i)$ and then

$$f_u\big((M)(K_{\tau_0}h_0)(K_{\tau_1}h_1)\dots(K_{\tau_k}h_k)\big) = (M)(H_{\tau_0}h_0)(H_{\tau_1}h_1)\dots(H_{\tau_k}h_k) \tag{5.2.66}$$

which is exactly the desired equation.

□

$\mathbf{\Pi}^1$-completeness We now have all the ingredients necessary to prove the $\mathbf{\Pi}^1$-completeness theorem: the idea of the proof is to use lemma (5.2.4) to recursively reconstruct the typing of M.

Theorem 5.2.4 ($\mathbf{\Pi}^1$-completeness)**.** *Let $\sigma \to \tau$ be a simple type. If M is a closed normal term such that $M \in Red_{\forall\overline{\alpha}(\sigma\to\tau)}$, then $\vdash M : \sigma \to \tau$ is derivable in simple type theory.*

Proof. From the fact that M satisfies equation (5.2.53) we will derive a typing of M.

Let $\sigma = \mu_1 \to \dots \to \mu_{q_1} \to \alpha_u$ and $\tau = \rho_1 \to \dots \to \rho_{q_2} \to \alpha_v$. We show, by induction over the number of applications in M, that if M satisfies an equation of the form (5.2.53), then it is a typed term of the form

$$\lambda x_0.\lambda x_1.\dots.\lambda x_h.(x_i)M_1 \dots M_p \tag{5.2.67}$$

where the variable x_0 has type $\rho_0 := \sigma$, the variables $x_j, 1 \le j \le h$ have type ρ_j, $\rho_i = \lambda_1 \to \dots \to \lambda_{k_i} \to \alpha_w$, for $0 \le i \le q_2 + 1$ and the M_j, for $0 \le j \le k_i$ are typed terms of type λ_j.

Equation (5.2.53) reduces to the following

$$f_v\big((M(K_\sigma g))(K_{\rho_1}h_1)\dots(K_{\rho_{q_2}}h_{q_2})\big) = M(H_\sigma g)(H_{\rho_1}h_1)\dots(H_{\rho_{q_2}}h_{q_2}) \tag{5.2.68}$$

By lemma (5.2.4) this implies that $M = \lambda x_0.\lambda x_1.\dots.\lambda x_h.(x_i)M_1 \dots M_p$, where $h \le q_2 + 1$, $0 \le i \le h$ and $p = k_i - (q_2 - h) - 1$.

If $p = 0$, then we are done; otherwise, equation (5.2.68) reduces to (where ρ_0 is σ)

$$f_v\big((K_{\rho_i}h_i)M_1^\dagger \dots M_p^\dagger(K_{\rho_{p+1}}h_{p+1})\dots(K_{\rho_{q_2}}h_{q_2})\big) = (H_{\rho_i}h_i)M_1^\ddagger \dots M_p^\ddagger(H_{\rho_{p+1}}h_{p+1})\dots(H_{\rho_{q_2}}h_{q_2}) \tag{5.2.69}$$

where $M_i^\dagger = M_i[(K_{\rho_j}h_j)/x_j]$ and $M_i^\ddagger = M_i[(H_{\rho_j}h_j)/x_j]$, for $1 \leq i \leq p$; this in turn reduces to

$$
\begin{aligned}
&f_v\big(h_i(H_{\lambda_1}M_1^\dagger)(H_{\lambda_2}M_2^\dagger)\dots(H_{\lambda_p}M_p^\dagger)(H_{\lambda_{p+1}}(K_{\rho_{h+1}}h_{h+1}))\dots(H_{\lambda_{k_i}}(K_{\rho_{q_2}}h_{q_2}))\big) = \\
&\quad f_v\big(h_i(K_{\lambda_1}M_1^\ddagger)(K_{\lambda_2}M_2^\ddagger)\dots(K_{\lambda_p}M_p^\ddagger)(K_{\lambda_{p+1}}(H_{\rho_{h+1}}h_{h+1}))\dots(K_{\lambda_{k_i}}(H_{\rho_{q_2}}h_{q_2}))\big)
\end{aligned}
\tag{5.2.70}
$$

Remark that, by lemma (5.2.3), the equation above implies that, for all $1 \leq j \leq q_2 + 1 - h$, one has $\rho_{h+j} \equiv \lambda_{p+j}$.

As a consequence of the remark above, equation (5.2.70) holds if and only if, for $1 \leq j \leq k_i$, the equations

$$H_{\lambda_j}M_j^\dagger = K_{\lambda_j}M_j^\ddagger \tag{5.2.71}$$

hold. Since $FV(M_j) = \{x_1, \dots, x_h\}$, we can restate the equations above as

$$H_{\lambda_j}\big(M_j^*(K_{\rho_1}h_1)\dots(K_{\rho_p}h_p)\big) = K_{\lambda_j}\big(M_j^*(H_{\rho_1}h_1)\dots(H_{\rho_p}h_p)\big) \tag{5.2.72}$$

where $M_j^* = \lambda x_1.\dots.\lambda x_h.M_j$, which reduce to

$$f_{w'}\big(M_j^*(K_{\rho_1}h_1)\dots(K_{\rho_p}h_p)(K_{\nu_1}h_1')\dots(K_{\nu_r}h_r')\big) = M_j^*(H_{\rho_1}h_1)\dots(H_{\rho_p}h_p)(H_{\nu_1}h_1')\dots(H_{\nu_r}h_r') \tag{5.2.73}$$

for a certain integer w', where $\lambda_i = \nu_1 \to \dots \to \nu_r$.

We can now apply the induction hypothesis, since the number of applications in M_j^* is strictly smaller than the number of applications in M. It follows that M_j^* has type $\rho_1 \to \dots \to \rho_p \to \nu_1 \to \dots \to \nu_r \to \alpha$ and then M_j has type $\nu_1 \to \dots \to \nu_r \to \alpha$.

$\square$

The following corollary lists some easy consequences of theorem (5.2.4) which determine the inner structure of closed normal reducible λ-terms and can be seen as introducing *last rule conditions* for reducibility (see next section):

Corollary 5.2.1 (Last rule conditions)**.** *The following hold:*

i. there are no closed normal terms in $Red_{\forall\alpha\alpha}$;

ii. the only closed normal term in $Red_{\forall\alpha(\alpha\to\alpha)}$ is $id := \lambda x.x$;

iii. the only closed normal terms in $Red_{\mathbf{N}}$ are id and the Chuch numerals;

iv. for all simple types σ, τ, the closed normal terms in $Red_{\forall\overline{\alpha}(\sigma\to\tau)}$ are of the form $\lambda z.M$, where M is a normal term such that, for all closed $N \in Red_{\forall\overline{\alpha}\sigma}$, $M[N/z] \in Red_{\forall\overline{\alpha}\tau}$;

v. for all simple types σ, τ, the closed normal terms in $Red_{\forall\alpha((\sigma\to\tau\to\alpha)\to\alpha)}$ (with $\alpha \notin FV(\sigma) \cup FV(\tau)$) are of the form $\lambda x.(x)MN$, where M, N are, respectively, closed normal terms in Red_σ and Red_τ;

vi. for all simple types σ, τ, the closed normal terms in $Red_{\forall\alpha((\sigma\to\alpha)\to(\tau\to\alpha)\to\alpha)}$ (with $\alpha \notin FV(\sigma) \cup FV(\tau)$) are of the form $\lambda x.\lambda y.(x)M$ or $\lambda x.\lambda y.(y)N$, where M, N are, respectively, closed normal terms in Red_σ and Red_τ.

Proof. i. If $M \in Red_{\forall\alpha\alpha}$, then $\lambda x.M \in Red_{\forall\alpha(\alpha\to\alpha)}$, where x is a fresh variable. Then, by lemma (5.2.2), M must be either of the form $\lambda x.(x)M_1 \dots M_p$, either of the form $(y)M_1 \dots M_p$; in both cases it follows that M is not closed.

ii. By lemma (5.2.4), $M = \lambda x.(x)M_1 \dots M_p$ and $p = 0$.

iii. The only closed normal terms of type $(\alpha \to \alpha) \to (\alpha \to \alpha)$ in simple type theory are id and the Church numerals.

iv. If M has type $\sigma \to \tau$ and is closed, then it has form $\lambda z.M'$ and its typing derivation has form

$$\frac{\begin{array}{c}\vdots\\ (z:\sigma) \vdash M' : \tau\end{array}}{\vdash M : \sigma \to \tau}\ (\to I) \tag{5.2.74}$$

Let N be a closed term in $Red_{\forall\overline{\alpha}\sigma}$; by theorem (5.2.4), $\vdash N : \sigma$ is derivable and, by the substitution lemma (2.1.1), $\vdash M[N/z]\tau$ is derivable too; moreover, since $M[N/z]$ is closed too, $\vdash M[N/z] : \forall\overline{\alpha}\tau$ is derivable in System F and, by the realizability theorem (3.2.1) for System F, it follows that $M[N/z] \in Red_{\forall\overline{\alpha}\tau}$.

v. If M has type $(\sigma \to \tau \to \alpha) \to \alpha$, then (lemma (5.2.4)) it has the form $\lambda x.(x)PQ$, where P is a normal term of type σ and Q is a normal term of type τ.Moreover, since $\alpha \notin FV(\sigma) \cup FV(\tau)$ and P, Q are normal terms, it follows that both terms are closed (i.e. $x \notin FV(P) \cup FV(Q)$): indeed, if it were not the case, say if $x \in FV(P)$, then there would exist a maximal subterm P' of P of a type ρ such that $\alpha \in FV(\rho)$; maximality means here that P occurs in a subterm of the form $(z)P_1 \dots P_l P P_{l+2} \dots P_n$, where $P = \lambda y_1. \dots .\lambda y_k.P''$ and $z = y_u$ for a certain $1 \leq u \leq k$.

Indeed, if P' is not maximal, then it occurs in a subterm of the form $(z)P_1 \dots P_l P' P_{l+2} \dots P_n$ (otherwise α would appear in σ) where $z \neq y_u$, for $1 \leq u \leq k$. This implies the existence of a subterm P''' of P, of which $(z)P_1 \dots P_l P' P_{l+2} \dots P_n$ is a subterm and whose type contains α. By induction one finds then a maximal subterm P' with the desired property.

Now, if such a maximal subterm P' exists, it follows that α occurs in σ, since $\sigma \equiv \sigma_1 \to \dots \to \sigma_{k'} \to \alpha$ and z has type σ_u.

vi. If M has type $(\sigma \to \alpha) \to (\tau \to \alpha) \to \alpha$ in simple type theory, then it has the from $\lambda x_1.\lambda x_2.(x_p)P$, where $1 \leq p \leq 2$ and P is a normal term of type σ (if $p = 1$) or type τ (if $p = 2$). Moreover, by the same argument as above, P must be closed.

□

5.3 An impredicative bridge

Theorem (5.2.4) and its corollaries (5.2.1) allow to retrieve structural conditions on the form of λ-terms from the reducibility semantics, thus providing a bridge between the untyped interpretation and proof-theoretic semantics. However, such a reconstruction heavily depends upon the interpretation of second order universal quantification, i.e. on the acceptance of an impredicative, non hierarchical, explanation of proofs.

In chapter (3) we discussed the main differences between the proof-theoretic semantics perspective and the untyped one arising from realizability and Tait-Girard reducibility. In particular, we insisted on the fact that the former focuses on the thesis that introduction rules are self-justifying and provides a definition of validity based on a *last rule condition*, namely the fact that a valid derivation should reduce to a derivation ending with an introduction rule for the principal connective of its conclusion. In the untyped perspective, on the contrary, derivations are interpreted by means of untyped programs and the focus is rather on the behavior of these programs under the cut-elimination (or normalization) procedure, thus ignoring the internal structure of derivations.

The results of the previous section allow then to define a bridge between the two perspectives, in particular a way to retrieve structural conditions on the form of proofs (as the last rule condition) starting from a behavioral description of untyped λ-terms. Such a bridge can be illustrated by the schema below:

$$\boxed{\text{Behavioral norms} \quad + \quad \text{Parametricity} \quad \Rightarrow \quad \text{Last rule condition}}$$

The chain of arguments can be described as follows: let A be a propositional formula, σ be $A^{\mathbb{F}}$ and M be a closed normal λ-term in $Red_{\forall\overline{\alpha}\sigma}$. As a consequence of theorem (5.2.2) and (5.2.3) M is parametric and satisfies the dinatural equations (5.2.53). Hence, from the $\mathbf{\Pi}^1$-completeness theorem (5.2.4) it follows that $\vdash M : \sigma$ is derivable in simple type theory; finally, from the faithfulness theorem (2.3.1) it follows that there exists a normal derivation d_M of conclusion A such that $\mathbb{F}(d_M) = M$.

As a consequence of corollary (5.2.1), we can thus state the following characterization (that we translate from sequent calculus to a natural deduction setting, more familiar to the proof-theoretic semantics tradition):

- there exists no closed derivation of P, for any atomic formula P;
- the only closed canonical derivation of $P \to P$, for any atomic formula P, is the (valid) derivation $\dfrac{[P]}{P \to P}\,(\to I)$;
- if $M \in Red_{\forall\overline{\alpha}(A^{\mathbb{F}} \to B^{\mathbb{F}})}$ is closed and normal, then d_M is a (valid) canonical derivation of $A \Rightarrow B$;
- if $M \in Red_{\forall\overline{\alpha}((A^{\mathbb{F}} \to B^{\mathbb{F}} \to \alpha) \to \alpha)}$ is closed and normal, d_M is of the form

$$\dfrac{\dfrac{\dfrac{[A \Rightarrow B \Rightarrow P]^x \qquad \begin{array}{c}\vdots\, d_1\\ A\end{array}}{B \Rightarrow P}\,(\Rightarrow E) \qquad \begin{array}{c}\vdots\, d_2\\ B\end{array}}{P}\,(\Rightarrow E)}{(A \Rightarrow B \Rightarrow P) \Rightarrow P}\,(\Rightarrow I)_x \tag{5.3.1}$$

  where d_1 is a closed canonical derivation d_1 of A and d_2 a closed canonical derivation of B;
- if $M \in Red_{\forall\overline{\alpha}((A^{\mathbb{F}} \to \alpha) \to (B^{\mathbb{F}} \to \alpha) \to \alpha)}$ is closed and normal, then d_M is either of the form

$$\dfrac{\dfrac{\dfrac{[A \Rightarrow P]^x \qquad \begin{array}{c}\vdots\, d_1\\ A\end{array}}{P}\,(\Rightarrow E) \qquad [B \Rightarrow P]^y}{(B \Rightarrow P) \Rightarrow P}\,(\Rightarrow I)_y}{(A \Rightarrow P) \Rightarrow (B \Rightarrow P) \Rightarrow P}\,(\Rightarrow I)_x \tag{5.3.2}$$

  where d_1 is a closed canonical derivation d_1 of A, either of the form

$$\dfrac{\dfrac{\dfrac{[B \Rightarrow P]^y \qquad \begin{array}{c}\vdots\, d_2\\ B\end{array}}{P}\,(\Rightarrow E)}{(B \Rightarrow P) \Rightarrow P}\,(\Rightarrow I)_y \qquad [A \Rightarrow P]^x}{(A \Rightarrow P) \Rightarrow (B \Rightarrow P) \Rightarrow P}\,(\Rightarrow I)_x \tag{5.3.3}$$

  where d_2 is a closed canonical derivation of B;

The truly remarkable fact about this bridge is that, in order to recover structural conditions over the form of derivations, one has to pass through impredicative quantification. This might appear quite counterintuitive at first, since, in order to decrease the "degrees of freedom" in the construction of a normal λ-term up to a finite number, one has to impose a *prima facie* infinitary condition like impredicative universal quantification.

More precisely, if we just consider simple types, then, as a consequence of the remarks in section (3.2.3), it turns out that the closed normal λ-terms in Red_σ are infinitely many and cannot be characterized structurally: the hierarchical, "predicative", definition of Red_σ cannot capture the internal structure of its terms. On the contrary, as soon as one introduces the universal closure $\forall\overline{\alpha}\sigma$, things change radically: as a consequence of lemma (5.2.4), one can define a sort of proof-search algorithm (used in the proof of theorem (5.2.4)) for enumerating all closed normal reducible terms, *following the subformula principle*: indeed, if $M \in Red_{\sigma_1\to\dots\to\sigma_k\to\alpha}$, then lemma (5.2.4) provides a *finite* list of possible configurations for M depending on other terms in the reducibility *of the subtypes of the* σ_i. In a word, *an impredicative notion (reducibility for universal types) is needed in order to recover the hierarchical, "predicative", inner structure of normal derivations.*

The interest of this bridge is that it allows to combine the two interpretations of proofs into a uniform frame (obviously, at the price of accepting the "epistemic circularity" - section (4.3.1) - of second order quantification): we can at the same time think of proofs as programs and as rule-based constructions. This issues the question of the compatibility of the two paradigms, the one based on the analysis of the behavior of derivations, the other based on the distinction between canonical and non canonical derivations.

Chapter 6

Vicious circles and typability

System F introduces a notion of typing which is not in accordance with Russell's rejection of vicious circles: polymorphic types allow the typing of terms containing variables applied to itself, as the term $\delta = \lambda x.(x)x$. Hence the type discipline does not forbid, but rather controls the several auto-applications that might occur in a term. Indeed, whereas the term δ can be typed in System F, the term $(\delta)\delta$, which is not normalizing, cannot be typed: its two autoapplications are in a sense *incompatible* and thus rejected by the type discipline.

This chapter is devoted to an analysis of polymorphic typability, i.e. of the polymorphic type disciplines of System F and its extensions U^- and N. The adoption of unification techniques to investigate the typability problem is quite standard in the literature (see [GRDR91]) and provides a geometrical interpretation of the vicious circles arising from the typability problem for a λ-term containing auto-applications (see [LC89] and [Mal90]). Here this interpretation is recalled and generalized, in order to investigate the possibility of an entirely combinatorial (i.e. not relying on reducibility theorems) characterization of the typability problem.

The main results of this chapter are the following: a generalization of the notion of "compatibility" (introduced in [Mal90] to investigate typability in System F) between constraints forced by recursive type equations is given, and its is proved (theorem (6.2.1)) that all strongly normalizable λ-terms are compatible. Then the typability problem for System U^- is considered and it is conjectured that compatibility (a decidable property) fully characterizes typability in that system. A generalized notion of constraint is introduced and some ideas towards the proof of the conjecture are sketched (in particular concerning the necessity of an impredicative universe like $\mathcal{U} = \forall \mathcal{X}\mathcal{X}$ in order to solve recursive equations in a general way).

Finally, some interesting consequences are drawn from the conjecture: first, such a characterization of typability for System U^- would imply that the terms typable in U^- are exactly the same which are typable in System N by means of types in normal form. Second, every strongly normalizable λ-term would be typable in System U^-. Third, a sketch is provided of the fact that, if the conjecture were true, then, for every total unary recursive function there would exist a λ-term computing the function and having type $\mathbf{N} \to \mathbf{N}$ in U^-.

6.1 Typing and unification

We recall in this section the equational characterization of the typing conditions for a pure λ-term in simple type theory and in System F. This means that a typing of a term exists in one of the two systems if and only if a solution can be found, among the types of the systems, to a finite set of equations between *schemes*, a syntactical counterpart of types (see [GRDR91, GRDR88]).

This characterization reduces the typability problem to a problem of first-order (resp. second order) unification, which is decidable in the first order case and undecidable in the second order case.

The interest of this characterization lies in the fact that it allows to investigate a problem of *derivability* within a formal system (as the systems of type inference are sequent calculus systems which essentially come from logic, as shown in chapter (2)) as a problem of *solvability* of a system of equations between functional terms. In particular, this provides a geometrical analysis of the "vicious circles" of the polymorphic type discipline, that is, the possibility to give a type to terms containing applications of a variable to itself, which is developed in the next section.

In the final subsection a new, equivalent, formulation of the equational characterization of polymorphic typing is defined, based on the definition of a tree which captures the dependency relations between type variables. This formulation will be used in the next section to investigate the typability problem for System U.

6.1.1 Equations in the simple type discipline

Let $\sigma = \tau$, for two types σ, τ, mean that σ and τ are the same type up to α-equivalence (i.e. up to permutation of bound variables), and $\sigma \equiv \tau$ mean that σ and τ are syntactically equal expressions. In the case of simple types, since there are no bound type variables, the two relations are equivalent.

Schemes and principal derivations The equational approach to typing in simple type theory arises as a consequence of the following two properties, which are easily established by induction over typing derivations:

(T1) $\Gamma \vdash \lambda x.M : \sigma$ *is derivable iff there exist types* σ_1, σ_2 *such that* $\sigma = \sigma_1 \to \sigma_2$ *and* $\Gamma, (x : \sigma_1) \vdash M : \sigma_2$ *is derivable;*

(T2) $\Gamma \vdash MN : \sigma$ *is derivable iff there exist types* σ_1, σ_2 *such that* $\Gamma \vdash M : \sigma_1 \to \sigma$ *and* $\Gamma \vdash N : \sigma_2$ *are both derivable and moreover* $\sigma_1 = \sigma_2$.

On the basis of properties $\mathbf{T1-2}$, one can devise, for each λ-term M, a system of equational specifications which characterizes all possible typings of M: this is usually done in two steps (see [GRDR91]). First, one defines *schemes*: a scheme can be thought as the set of all possible types that can be obtained from it by substitution. For instance, a scheme $\phi \to \psi$ indicates the set of all types of the form $\sigma \to \tau$, for arbitrary σ, τ.

Next one defines, for each λ-term M, a *principal typing derivation* d_M which characterizes all possible typings of M, in the sense that any typing of M can be obtained from d_M by substituting types for the schemes occurring in it. From the derivation d_M one finally extracts a finite set of equations over schemes which characterize the derivation.

The syntax of (simple) schemes is the same as the one of simple types: one has a set of *scheme variables* $\phi, \psi, \ldots$ and defines schemes Φ, Ψ by the following grammar:

$$\Phi, \Psi := \phi | \Phi \to \Psi \tag{6.1.1}$$

Two schemes Φ, Ψ are *disjoint*, when the scheme variables occurring in them are pairwise distinct. A scheme is *linear* if all the scheme variables occurring in it are pairwise distinct. Moreover, let e be a set of equations between schemes of the form $\phi = \Phi$; then e is said a *linear system* if, for all equation $\phi = \Phi \in e$, Φ is linear and ϕ and Φ are disjoint.

A *ground substitution* S is any map (that we will note Φ^S) from schemes to simple types preserving implication, i.e. such that

$$(\Phi \to \Psi)^S = \Phi^S \to \Psi^S \tag{6.1.2}$$

A scheme Φ can be though then as the set of all the simple types that can be obtained from it by means of ground substitutions, i.e. as the set $[\Phi]$ of all types of the form Φ^S, where S is a ground substitution. For instance, $[\phi]$ is the set of all types, whereas $[\phi \to \psi]$ is the set of all types of the form $\sigma \to \tau$, ecc.

The principal typing derivation d_M and the set $eq(M)$ of equational specifications, for a λ-term M with free variables $x_1, \ldots, x_n$, are defined by induction on the construction of M as follows:

- if $M = x_i$, for a certain $1 \le i \le n$, then d_M is the derivation $(x_1 : \phi_1), \ldots, (x_n.\phi_n) \vdash x_i : \psi$, where $\phi_1, \ldots, \phi_n, \psi$ are distinct scheme variables, and $eq(M) = \{\phi_i = \psi\}$;
- if $M = \lambda z.M'$, then d_M is the following derivation:

$$\frac{\begin{array}{c}\vdots\, d_{M'} \\ \Gamma, (x : \phi_1) \vdash M' : \phi_2\end{array}}{\Gamma \vdash \lambda z.M' : \phi} \tag{6.1.3}$$

  where ϕ is a fresh scheme variable, and $eq(M) = eq(M') \cup \{\phi = \phi_1 \to \phi_2\}$;

- if $M = M_1 M_2$, then d_M is the following derivation:

$$\frac{\begin{array}{cc}\vdots\, d_{M_1} & \vdots\, d_{M_2} \\ \Gamma \vdash M_1 : \phi & \Gamma \vdash M_2 : \psi\end{array}}{\Gamma \vdash M_1 M_2 : \chi} \tag{6.1.4}$$

  where χ is a fresh scheme variable, and $eq(M) = eq(M_1) \cup eq(M_2) \cup \{\phi = \psi \to \chi\}$.

One can easily verify by induction that $eq(M)$ is a linear system.

Let us say that a ground substitution S *satisfies* a set e of equations over schemes if, for all equation $\Phi = \Psi \in e$, $\Phi^S = \Psi^S$. If a ground substitution S satisfies a system $eq(M)$, then one can define a typing derivation d_M^S in simple type theory, which is obtained by replacing, in d_M, all schemes Φ by types Φ^S.

For each λ-term M, let S be a ground substitution which satisfies $eq(M)$; one can verify by induction that d_M^S is a correct typing derivation in simple type theory. Indeed one can show the following two properties of principal typing derivations (the proof can be found in [GRDR88]):

Proposition 6.1.1 (principal typing derivations, [GRDR88])**.** *Let M be a λ-term, then the following two hold:*

i. if a ground substitution S satisfies $eq(M)$, then d_M^S is a typing derivation of M in S;

ii. if d is a typing derivation of M in S, then there exists a ground substitution S satisfying $eq(M)$ and such that $d = d_M^S$.

Proof. Both parts are proved by a straightforward induction on the construction of d_M^S. □

First order unification and the principal typing The interest of reducing the problem of the simple typing of terms to the solution of a system of equational specifications for schemes is that we can apply first-order unification to solve the problem in a decidable way. Indeed, simple schemes can be represented by means of a first-order functional language, i.e. a language for defining terms by means of variables $x, y, \ldots$ and a symbol f for a binary function (which corresponds to implication). As a consequence, the systems $eq(M)$ can be seen as systems of equations of the form $t = u$ between first-order terms.

First-order unification, a technique whose origins can be traced back to Herbrand's thesis [Her67] and was firstly formalized in [Rob65], is a standard approach to the solution of systems of equations between first-order terms. Given a system E of first order equations $t_i = u_i$, for $1 \leq i \leq n$, a *unifier* for E is a substitution θ (i.e. a map from first order variables to first order terms), such that, for all $1 \leq i \leq n$, $t_i\theta = u_i\theta$ is a syntactic identity.

A possible way to implement the idea of unification is by means of a set of transformations which simplify a system of first-order equations into one whose solution is trivial, similarly to what happens when one applies Gaussian elimination to solve a system of linear equations (this approach is developed for instance in [GS89]).

A system E is in *solved form* if all its equations are of the form $x_i = t_i$, where x_i does not occur in t_i nor in the rest of E. A system in solved form admits the trivial solution defined by $\theta(x_i) = t_i$, for all $1 \leq i \leq n$.

The simplification transformations are of three types:

identity Equations of the form $x_i = x_i$ can be simply eliminated, since they carry no information;

decomposition If E contains an equation of the form $f(t_1, \dots, t_n) = f(u_1, \dots, u_n)$, then we can simplify the system by replacing the equation by the n simpler equations $t_1 = u_1, \dots, t_n = u_n$. If E contains an equation of the form $f(t_1, \dots, t_n) = g(u_1, \dots, u_m)$, then the algorithm must output "$FAILURE$", since no substitution can make the two terms equal;

variable elimination If E contains an equation of the form $x_i = t_i$, then two possible cases occur: if x_i occurs in t_i, then the algorithm must output "$FAILURE$" since no substitution can make x_i and t_i equal (this is the so-called *occur-check*); if x_i does not occur in t_i, then the equation can be eliminated and the remaining system $E' = E - \{x_i = t_i\}$ transformed into $E'[t_i/x_i]$.

The transformations above, seen as inference rules, define a *non deterministic* algorithm which either outputs "FAILURE" in all of its branches, either outputs a system in solved form in one of them; since all branches can be shown to be finite, the unification algorithm is decidable. Remark that variable elimination is responsible for the fact that different systems in solved form may appear in different branches: this transformation corresponds intuitively, in the case of linear systems, to the usual procedure of *solving the system for a variable* x_i.

The two main features of first order unification are the fact that the algorithm is decidable and the fact that, if at least one of the branches does not output "FAILURE", then it produces a *most general unifier* (*m.g.u.*): this means that all other unifiers θ' for S can be factorized as $\theta' \circ \theta''$ for a certain substitution θ''.

The systems which come from simple type theory correspond to a simplified case of unification, since the language contains only one symbol for (binary) function. This implies that the $FAILURE$ case in the decomposition transformation never occurs, and thus the only case of failure is provided by an occur-check. For instance, given the λ-term $M = \lambda x.(x)x$, if ξ is the type variable assigned to the variable x, then an equation of the form $\xi = \xi \to \xi'$, which contains an occur-check, is produced during the execution of the unification algorithm, which must output $FAILURE$.

More generally, all systems from which a *recursive* equation (i.e. an equation of the form $\xi = \Phi$, where ξ occurs in the scheme Φ) is derivable, cannot be solved by first-order unification; hence all terms inducing such systems cannot be typed in simple type theory. An example of these terms are all terms containing an *auto-application* of the form

$$\lambda x_1.\dots.\lambda x_n.(x_i)M_1 \dots M_p x_i M_{p+1} \dots M_k \tag{6.1.5}$$

for $1 \leq i \leq n$. This is indeed tantamount to saying that the discipline of simple types forbids any form of auto-application, i.e. of application of a variable to itself (this was indeed the reason why Russell introduced this discipline).

From the properties of first-order unification one can derive two fundamental properties of type inference in simple type discipline: first, typability and type-checking are both decidable; second, if a term M is typable, then it has a *principal* typing (up to permutation of type variables), namely the one which comes from a most general unifier of $eq(M)$ (see [Hin69, Mil78]).

6.1.2 Equations in the polymorphic type discipline

One of the main features of the simple type discipline, as we have observed, is the "Russellian" property that auto-applications cannot be typed or, equivalently, that recursive equations cannot be unified. This restriction is overcome in the polymorphic type discipline: for instance, if a variable x is declared of type $\forall\alpha\alpha$, then it can be *extracted* on two arbitrary types, for instance, $\alpha \to \alpha$ and α. Hence the term $\lambda x.(x)x$ can be given type $\forall\alpha\alpha \to \forall\alpha\alpha$ in System F. At the same time, the reducibility theorem (4.1.1) implies that the auto-applications typable in System F do not lead to "paradoxical", i.e. not normalizing typable terms.

Second order schemes The equational approach to typing in polymorphic type discipline arises as a consequences of two properties $\mathbf{T1}', \mathbf{T2}'$ which are established by induction:

(T1′) *$\Gamma \vdash \lambda x.M : \sigma$ is derivable iff there exist variables $\alpha_1, \dots, \alpha_n$ and types σ_1, σ_2 such that $\sigma = \forall\alpha_1 \dots \forall\alpha_n(\sigma_1 \to \sigma_2)$, $\alpha_1, \dots, \alpha_n$ are bindable in Γ and $\Gamma, (x : \sigma_1) \vdash M : \sigma_2$ is derivable;*

(T2′) *$\Gamma \vdash MN : \sigma$ is derivable iff there exist variables $\alpha_1, \dots, \alpha_n$ and types $\sigma_1, \sigma_2, \tau, \rho$ such that $\sigma = \forall\alpha_1 \dots \forall\alpha_n\rho$, $\Gamma \vdash M : \sigma_1 \to \tau$ and $\Gamma \vdash N : \sigma_2$ are both derivable and moreover $\sigma_1 = \sigma_2$ and $\tau \leq \rho$.*

where the relation $\leq$ is the transitive closure of the relation $\leq_1$ defined, for all types σ, τ, by

$$\forall\alpha\sigma \leq_1 \sigma[\tau/\alpha] \tag{6.1.6}$$

The properties above lead to the definition of a *syntax-directed* type inference system for F (see [GRDR91]): this means that, for all term M having a type in the system, every derivation of a typing of M has a shape depending only on the structure of M.

$$\boxed{\begin{array}{lll}
(\text{var}) & \Gamma, (x:\sigma) \vdash x : \tau \quad \sigma \leq \tau & \\
(\to I) & \dfrac{\Gamma, (x:\sigma) \vdash M : \tau}{\Gamma \vdash \lambda x.M : \forall\overline{\alpha}.\sigma \to \tau} & (\overline{\alpha} \text{ bindable in } \Gamma) \\
(\to E) & \dfrac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma \quad \tau \leq \rho}{\Gamma \vdash MN : \forall\overline{\alpha}.\rho} & (\overline{\alpha} \text{ bindable in } \Gamma)
\end{array}} \tag{6.1.7}$$

The system (6.1.7) is easily seen to be equivalent to the one given in chapter (2). In the following, when referring to a typing derivation in F, we will refer to a derivation in the syntax-directed system above.

In order to define (second order) schemes one needs the following sets:

- a countable set of *sequence variables* $a, b, \ldots$ which correspond, intuitively, to a (possibly empty) finite package of type variables;
- a countable set of *scheme variables* $\phi, \psi, \ldots$;
- for every sequence variable a, a countable set of *pseudo-substitutions of domain a* $I_a, J_a, \ldots$.

The syntax of *pseudo-schemes* is then defined by the following grammar:

$$\Phi := \phi | \Phi \to \Psi | \forall a \Phi | I_a(\Phi) \tag{6.1.8}$$

Two pseudo-schemes are *disjoint* if their sequence variables and scheme variables are pairwise distinct. A pseudo-scheme is *linear* if all its subschemes are pairwise disjoint. The syntax of schemes is the following:

$$\Phi, \Psi := \forall a \phi | \Phi \to \Psi | \forall a \Phi | I_a(\Phi) \tag{6.1.9}$$

Disjointness and linearity for schemes is exactly the same as for pseudo-schemes.

Second order pseudo-schemes can be seen as sets of polymorphic types: let Σ_F be the set of types of System F; a *ground substitution* S is given by

- a map a^S from sequence variables to finite sequences (possibly empty) of type variables such that, if $a \neq a'$, then $a^S \cap a'^S = \emptyset$;
- a map I_a^S from pseudo substitutions of domain a to substitutions (i.e. functions from type variables to Σ_F) of domain a^S
- a map ϕ^S from pseudo-schemes to types commuting with substitutions, $\to$ and $\forall$, i.e. such that

$$\big(I_a(\Phi)\big)^S = \Phi^S I_a^S \tag{6.1.10}$$

$$(\Phi \to \Psi)^S = \Phi^S \to \Psi^S \tag{6.1.11}$$

$$(\forall a.\Phi)^S = \forall a^S.\Phi^S \tag{6.1.12}$$

  where $\forall a^S.\sigma$ is $\forall \alpha_1 \ldots \forall \alpha_n \sigma$, where $a^S = (\alpha_1, \ldots, \alpha_n)$ (remark that a^S can be empty).

Given a pseudo-scheme Φ, the set $[\Phi] \subseteq \Sigma_F$ can be defined as before as the set of all Φ^S, for S a ground substitution. For instance $[\phi]$ is the set Σ_F^- of non externally quantified types, and $[\phi \to \psi]$ is the set of types of the form $\sigma \to \tau$. Remark that, since sequence variable can have an empty interpretation, for all pseudo-scheme Φ, $[\forall a \Phi] \subseteq [\Phi]$. The algebraic structure of the sets of the form $[\Phi]$ is investigated in detail in [Mal92].

The principal typing derivation As in the first order case, we can define, for each λ-term M, a principal typing derivation d_M, defined over schemes and a set $eq(M)$ of equational specifications over pseudo-schemes with substitutions. Moreover, we will define a new set $ct(M)$ of *constraints*, i.e pairs of the form (a, B), where a is a sequence variable and B a finite set of schemes. A constraint $(a, (\Phi_1, \ldots, \Phi_n))$ will be interpreted as follows: the variables in a^S must be bindable in the context $\Gamma = \Phi_1^S, \ldots, \Phi_n^S$.

The sets $eq(M)$ along with $ct(M)$ will be then enough to characterize any typing of M.

- if $M = x_i$, for a certain $1 \leq i \leq n$, then d_M is the derivation $(x_1 : \forall a_1 \phi_1), \ldots, (x_n.\forall a_n \phi_n) \vdash x_i : \forall b \psi$, where $\forall a_1 \phi_1, \ldots, \forall a_n \phi_n, \forall b \psi$ are disjoint scheme variables, $eq(M) = \{I_{a_i}(\phi_i) = \psi\}$ and $ct(M) = \{(b, \{\forall a_1 \phi_1, \ldots, \forall a_n \phi_n\})\}$;

- if $M = \lambda z.M'$, then d_M is the following derivation:

$$\frac{\begin{array}{c}\vdots\ d_{M'}\\ \Gamma, (x : \forall a_1\phi_1) \vdash M' : \forall a_2\phi_2\end{array}}{\Gamma \vdash \lambda z.M' : \forall b\phi} \tag{6.1.13}$$

  where ϕ is a fresh scheme variable, and $eq(M) = eq(M') \cup \{\phi = \forall a_1\phi_1 \to \forall a_2\phi_2\}$ and $ct(M) = ct(M') \cup \{(b, \Gamma)\}$;

- if $M = M_1M_2$, then d_M is the following derivation:

$$\frac{\begin{array}{c}\vdots\ d_{M_1}\\ \Gamma \vdash M_1 : \forall a\phi\end{array} \quad \begin{array}{c}\vdots\ d_{M_2}\\ \Gamma \vdash M_2 : \forall b\psi\end{array}}{\Gamma \vdash M_1M_2 : \forall c\chi} \tag{6.1.14}$$

  where χ and c are fresh, and $eq(M) = eq(M_1) \cup eq(M_2) \cup \{\phi = \forall b\psi \to \forall b'\psi', I_b(\psi') = \chi\}$, where I_b is a fresh pseudo-substition and $ct(M) = ct(M_1) \cup ct(M_2) \cup \{(c, \Gamma)\}$.

We can redefine the notions of linear scheme and linear system: a (pseudo)-scheme Φ is *linear* if all its sub-schemes are pairwise disjoint. A set of equations between pseudo-schemes with substitutions is *linear* if $e = \{\Phi = \Psi_i | 1 \leq i \leq n\}$, where Φ and the Ψ_i are linear schemes, Ψ_i is not externally quantified and $FV(\Phi) \cap FV(\Psi_i) = \emptyset$.

The following lemma describes the structures of the systems $eq(M)$ in terms of its linear subsystems.

Lemma 6.1.1. *For each λ-term M, the system $eq(M)$ is a union of linear systems, and the following hold:*

i. for all sequence variable a, if $\forall a\Phi$ and $\forall a\Psi$ occur in equations in the system, then $\Phi \equiv \Psi$ (where $\equiv$ denotes syntactic equality between schemes);

ii. for all schemes Φ, Ψ occurring in equations in the system, if $FV(\Phi) \cap FV(\Psi) \neq \emptyset$, then either Φ is a subscheme of Ψ or viceversa.

Proof. See [GRDR91]. □

A simple corollary of the point $i.$ above is that all equations of the form $\phi = \forall a_1\Phi \to \forall a_2\psi$, that is all equations in which no pseudo-substitution occurs, so as all equations of the form $\Phi = \phi$ (where ϕ is a scheme variable) can be simplified: we can take such equations as a definition of ϕ, and the lemma above remains valid for the simplified system so obtained, that we call $eq^*(M)$ (in [GRDR91] this is shown in detail by defining a $UNIFY$ algorithm over schemes).

Using the simplified system $eq^*(M)$ we can straightforwardly define a simplified derivation d^*_M, which differs from d_M in the fact that all schemes are replaced by their simplified form.

A ground substitution *satisfies* a set e of equations between pseudo-schemes with substitution if, for all equation $\Phi = \Psi$, $\Phi^S = \Psi^S$ holds; moreover, S satisfies a set c of constraints if, for all constraint $(a, \{\Phi_1, \ldots, \Phi_n\}) \in C$, a^S is bindable in $\{\Phi_1^S, \ldots, \Phi_n^S\}$.

If a ground substitution S satisfies a system $eq^*(M)$, then one can define a typing derivation d^S_M in System F, which is obtained by replacing in d_M every scheme Φ by the type Φ^S.

Let us consider a simple example to introduce the reader to the system $eq^*(M)$:

Example 6.1.1. *Let M be the λ-term $\lambda x.(x)x$. As we saw, this term contains an auto-application forbidden in the simple type discipline; on the contrary, it is quite easy to find solutions to the equation below, which is the only element of $eq^*(M)$:*

$$I_a(\phi) = J_a(\phi) \to \forall b\psi \tag{6.1.15}$$

For instance, take $\Phi^S = \alpha$, $I_a^S(\alpha) = \alpha \to \forall\beta\beta, J_a^S(\alpha) = \alpha$ and $(\forall b\psi)^S = \forall\beta\beta$. Then M can be typed $\forall\alpha\alpha \to \forall\beta\beta$.

For second order principal typing derivations one can prove an analogue of proposition (6.1.1):

Proposition 6.1.2 (principal typing derivations, [GRDR91])**.** *Let M be a λ-term, then the following two hold:*

i. if a ground substitution S satisfies $eq^*(M)$ and $ct(M)$, then d_M^S is a typing derivation of M in F;

ii. if d is a typing derivation of M in F, then there exists a ground substitution S satisfying $eq^*(M)$ and $ct(M)$ and such that $d = d_M^S$.

Proof. Both parts are proved by a straightforward induction on the construction of d_M^S. □

Remark 6.1.1. *To a maximal application*[1] *of the form $(x)P_1 \dots P_k$ there correspond equations in $eq^*(M)$ of the form*

$$I_{c_1}(\Phi) = \Psi_1 \to \psi_1 \tag{6.1.16}$$
$$I_{c_2}(\psi_1) = \Psi_2 \to \psi_2 \tag{6.1.17}$$
$$\vdots \tag{6.1.18}$$
$$I_{c_k}(\psi_{k-1}) = \Psi_k \to \psi \tag{6.1.19}$$

where the Ψ_i, for $1 \leq i \leq k$, are either schemes either pseudo-schemes of the form $I_{d_i}(\Psi_i')$, for a certain pseudo-scheme Ψ_i'. It is convenient to put the equations above together into a single equation of the form

$$I_{c_k}(\dots I_{c_1}(\Phi)\dots) = \Psi_1 \to \dots \to \Psi_k \to \psi \tag{6.1.20}$$

Hence, we can consider $eq^*(M)$ as the system generated by equations like (6.1.20) *which correspond to maximal applications in M.*

Second order unification For solving second order systems two standard approaches based on variants of unification can be found in the literature, depending on how one takes account of type extractions. The first approach (see [Pfe88]) is based on *second order unification*, i.e. unification over a language containing second order variables $F, G, H, ...$ and an abstraction operator λ over first-order variables. In particular, a second order unifier for a system S of equations $t_i = u_i$ between second order terms is a substitution (mapping first-order variables into first-order terms and second order variables into second order terms) θ such that, for all $1 \leq i \leq n$, $t_i\theta$ is α-equivalent to $u_i\theta$ (remark that syntactic identity is replaced by α-equivalence to cope with the

[1] i.e. a sequence of applications $(\dots((x)P_1)\dots)P_k$ in M such that, either $M = (x)P_1 \dots P_k$, either M contains the subterm $\lambda z.(x)P_1 \dots P_k$, for a certain variable z.

fact that first-order variables can be bound). Second order unification is an undecidable problem (see [Gol81]) and moreover does not admit of $m.g.u.$s: for instance, the equation

$$F(f(x,y)) = f(F(x),y) \tag{6.1.21}$$

admits infinitely incomparable unifiers of the form $\theta(F) = \lambda x.f(\dots f(x,y),\dots,y)$.

The second approach is based upon a variant of first-order unification, called *semi-unification* (see [Hen89, Hen88]): in this case one considers a system of *inequations* $t_i \preceq u_i$ between first-order terms; a solution to the semi-unification problem for a system S of inequations $t_i \preceq u_i$ is a set of substitutions $\theta, \theta_1, \dots, \theta_n$ such that, for all $1 \leq i \leq n$, the equality $t_i\theta\theta_i = u_i\theta$ is a syntactic identity.

The idea is that an equation of the form $I_a(\Phi) = \Psi$ is interpreted as the inequation $\forall a\Phi \leq \Psi$. Semi-unification is an undecidable problem too ([KTU93]), and no notion of "most general semi-unifier" holds for solutions to a semi-unification problem.

As a consequence of these approaches, we get that systems of equations over second order schemes do not enjoy the two main properties of their simple type cousins: their solution constitutes an undecidable problem[2] and there are no principal solutions. For instance, all types of the form $\forall\alpha(\alpha \to \psi), \forall\alpha((\alpha \to \psi) \to \psi), \forall\alpha(((\alpha \to \psi) \to \psi) \to \psi)$ are solutions to equation (6.1.15) and are thus types for $\lambda x.(x)x$.

6.1.3 Another scheme system

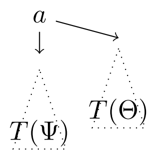
We introduce in this subsection a slightly different formulation of the systems of equations over second order schemes, equivalent to the one above. This formulation is based on the definition of a tree, depending on the derivation d^*_M, which allows to handle the dependencies between sequence variables (i.e. the constraints in $ct(M)$) in a more synthetic way.

Let, for any scheme Φ, a tree $T(\Phi)$ be defined as follows:

- if Φ is $\forall a\phi$, then $T(\Phi)$ is the tree

$$\begin{array}{c} a \\ \downarrow \\ \phi \end{array} \tag{6.1.22}$$

- if Φ is $\forall a(\Psi \to \Theta)$, then $T(\Phi)$ is the tree



(6.1.23)

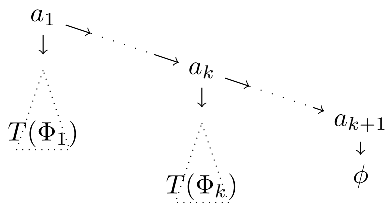
Definition 6.1.1. *By induction on d^*_M we define a tree $T(M)$ whose nodes are labeled by sequence variables and whose leaves are labeled by occurrences of scheme variables. Moreover, let the scheme of M be $\Phi = \forall a_1(\Phi_1 \to \forall a_2(\Phi_2 \to \dots \to \forall a_k(\Phi_k \to \forall a_{k+1}\phi)\dots))$; then $T(M)$ has the following properties:*

- *all the $T(\Phi_i)$, for $1 \leq i \leq k$, occur "appended", in order, to the first k nodes (of label $a_1, \dots, a_k$) occurring in the rightmost path;*

[2] The undecidability of typability for System F was first proved, independently from the equational approach, in [Wel98].

- *the scheme* ϕ *occurs "appended" to the last node (of label* a_{k+1}*) occurring in the rightmost path.*

The tree $T(M)$ *has thus the shape illustrated below:*



(6.1.24)

- *if* d^*_M *is of the form*

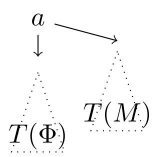
$$\Gamma, (x : \forall a \Phi) \vdash x : \Psi \tag{6.1.25}$$

  for a certain scheme Ψ*, then* $T(M)$ *is simply* $T(\Psi)$*;*

- *if* d^*_M *is of the form*

$$\frac{\begin{array}{c}\vdots\\ \Gamma, (x : \Phi) \vdash M : \Psi\end{array}}{\Gamma \vdash \lambda x.M : \forall a(\Phi \to \Psi)} \tag{6.1.26}$$

  for a certain scheme Φ*, then* $T(\lambda x.M)$ *is of the form*

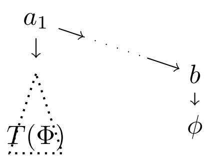


(6.1.27)

- *if* d^*_M *is of the form*

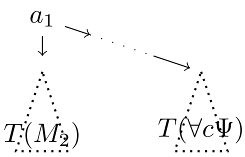
$$\frac{\begin{array}{c}\vdots\\ \Gamma \vdash M_1 : \forall a_1(\Phi \to \forall b\phi)\end{array} \quad \begin{array}{c}\vdots\\ \Gamma \vdash M_2 : \Phi\end{array}}{\Gamma \vdash M_1 M_2 : \forall c \Psi} \tag{6.1.28}$$

  were $eq^*(M)$ *contains the equation* $I_b(\phi) = \Psi$*, for a certain (non externally quantified) scheme* Ψ*, then by induction hypothesis* $T(M_1)$ *is of the form*



(6.1.29)

  then $T(M)$ *is the tree below*



(6.1.30)

Remark that a sequence variable or a scheme variable can occur several times in $T(M)$. However, we can bijectively associate a sequence variable a with the leftmost node of label a, that we call $n(a)$. Hence, given a sequence variable a, we can define its *initial segment* $s_a^{\triangleright}$ as the sequence of sequence variables $(a_0, \ldots, a_k)$ where $a = a_k$ and $a_0, \ldots, a_k$ are the sequence variables which label (in this order) the leftmost oriented path from the root of $T(M)$ to $n(a)$.

Since any equation E in $eq^*(M)$ corresponds to a maximal sequence of application rules, one can associate with it a unique subtree of $T(M)$: let $d(E)$ be the subderivation of d_M^* ending with such a sequence of application rules. Then the construction above associates with $d(E)$ a tree $T(E)$; remark that, since the sequence of application rules is maximal, one can verify that, by construction, the tree $T(E)$ is a subtree of $T(M)$.

We must now reformulate the syntax of schemes. We still use sequence variables and scheme variables but, instead of pseudo-substitutions, we will consider now, for each sequence variable a, a countable set of *substitution variables* $F_a, G_a, \ldots$ which will be symbols for n-ary functions, where n is the length of the initial segment $s_a^{\triangleright}$.

A *substitution term* is a first order term built by using sequence variables and substitution variables.

An *atomic scheme* is an expression of the form $\phi(t_1, \ldots, t_n)$, where ϕ is a sequence variable and $t_1, \ldots, t_n$ are substitution terms. A *substitution scheme* Φ is defined by the grammar:

$$\Phi, \Psi := \phi(t_1, \ldots, t_n) | \Phi \to \Psi | \forall a \Phi \tag{6.1.31}$$

The interest of the tree $T(M)$ is that it allows to define a order relation over sequence variables occurring in $eq^*(M)$: for any two sequence variables a, b let $a \triangleright_1 b$ if there exists in $T(M)$ an oriented edge from $n(a)$ to $n(b)$, and $a \triangleright b$ if there exists in $T(M)$ and oriented path from $n(a)$ to $n(b)$. Since $T(M)$ is a tree, every node is connected to the root by a unique path labeled by sequence variables $a_0, \ldots, a_k$.

The intuition behind the order relation above is that a sequence variable a is to be viewed as "bound" (resp. "free") with respect to a sequence variable b if $b \triangleright a$ (resp. $(a \triangleright b)$). This means that, given a ground substitution S (to be defined below), if a substitution is applied to one of the variables in $(b)^S$ (resp. $(a)^S$), then this substitution can not introduce occurrences of the variables in $(a)^S$ (resp. $(b)^S$), since substitution cannot introduce bound variables.

A *ground substitution* S now is given by:

- a map a^S from sequence variables to finite sequences (possibly empty) of type variables such that, if $a \neq a'$, $a^S \cap b^S = \emptyset$;
- a map F_a^S from substitutions variables of domain a to substitutions (i.e. functions from type variables to Σ_F) of domain a^S. Remark that this induces a map θ_t from substitution terms to substitutions defined as follows:
  - $\alpha\theta_a^S = \alpha$, for $\alpha \in a^S$;
  - $\alpha\theta^S_{F_a(t_1,\ldots,t_n)} := \alpha F_a^S \theta^S_{t_n} \ldots \theta^S_{t_1}$, for $\alpha \in a^S$.
- a map ϕ^S from substitution schemes to types commuting with substitutions, $\to$ and $\forall$, i.e. such that

$$\big(\phi(t_1, \ldots, t_n)\big)^S = \phi^S \theta^S_{t_n} \ldots \theta^S_{t_1} \tag{6.1.32}$$

$$(\Phi \to \Psi)^S = \Phi^S \to \Psi^S \tag{6.1.33}$$

$$(\forall a.\Phi)^S = \forall a^S.\Phi^S \tag{6.1.34}$$

  where $\forall a^S.\sigma$ is $\forall \alpha_1 \ldots \forall \alpha_n \sigma$, where $a^S = (\alpha_1, \ldots, \alpha_n)$ (remark that a^S can be empty).

With the help of $T(M)$ we will translate then the system $eq^*(M)$ into a system $sc(M)$ defined over substitution schemes. In particular, to any pseudo-scheme Φ occurring in $eq^*(M)$ we will associate a substitution scheme $sc(\Phi)$. Moreover, from a ground substitution S satisfying $sc(M)$ we will show how to construct a ground substitution S^* satisfying $eq^*(M)$.

For every scheme variable ϕ, let us consider the leftmost leaf node a_ϕ whose label is ϕ. Let $s_\phi^\triangleright$ be the initial segment $s_a^\triangleright = (a_0, \ldots, a_{k-1}, a)$, where a is the only variable such that $T(\forall a\phi)$ is a subtree of $T(M)$. Then

$$sc(\phi) := \phi(a_0, \ldots, a_{k-1}, a) \tag{6.1.35}$$

Let ι be any injective map from pseudo-substitutions to substitution variables such that, for all sequence variable a, $\iota(I_a)$ is of the form F_a. We define an invertible map i from the pseudo-schemes occurring in $eq^*(M)$ to substitution schemes. If F_a a substitution variable and Φ a substitution scheme, let Φ^{F_a} be obtained from Φ by replacing every occurrence of a in the atomic schemes in Φ by $F_a(a_1, \ldots, a_{k-1}, a)$ (where $(a_1, \ldots, a_{k-1}, a)$ is $s_a^\triangleright$). Now we put:

$$i(\phi) = \phi(a_1, \ldots, a_n) \qquad ((a_1, \ldots, a_n) = s_{\phi_1}^\triangleright) \tag{6.1.36}$$
$$i(I_a(\Phi)) = \big(i(\Phi)\big)^{\iota(I_a)} \tag{6.1.37}$$
$$i(\Phi \to \Psi) = i(\Phi) \to i(\Psi) \tag{6.1.38}$$
$$i(\forall a\Phi) = \forall a\ i(\Phi) \tag{6.1.39}$$

Finally, for any equation

$$I_{c_k}(\ldots I_{c_1}(\Phi)\ldots) = \Psi_1 \to \cdots \to \Psi_k \to \psi \tag{6.1.40}$$

occurring in $eq^*(M)$, the system $sc(M)$ contains the equation

$$i\big(I_{c_k}(\ldots I_{c_1}(\Phi)\ldots)\big) = i(\Psi_1) \to \cdots \to i(\Psi_k) \to i(\psi) \tag{6.1.41}$$

Hence all equations in $sc(M)$ are of the form

$$\Phi^{F_{a_1}\ldots F_{a_k}} = \Psi_1 \to \cdots \to \Psi_k \to \psi \tag{6.1.42}$$

for certain substitution schemes $\Phi, \Psi_1, \ldots, \Psi_k, \psi$, where, for $1 \le i \le k$, Ψ_i is either a simple substitution scheme, either of the form $\Theta_i^{F_{b_i}}$, for a certain substition variable F_{b_i} of domain b_i.

A ground substitution S satisfies the system $sc(M)$ if, for all equation $\Phi = \Psi$ in $sc(M)$, one has $\Phi^S = \Psi^S$ and moreover, for all scheme variable ϕ, the free variables of ϕ^S are among the $a_0^S, \ldots, a_n^S$, where $s_\phi^\triangleright$ is the linear order $a_0 \triangleright_1 \cdots \triangleright_1 a_n$. The latter condition (see proposition (6.1.3) below) allows to get rid of the set $ct(M)$ of constraints.

Let S be a ground substitution (over substitution schemes). We can define a ground substitution S^* over schemes as follows:

Next we put:

$$a^{S^*} := a^S \tag{6.1.43}$$
$$I_a^{S^*} := (\iota(I_a))^S \tag{6.1.44}$$
$$\phi^{S^*} := \phi^S \tag{6.1.45}$$
$$(\Phi \to \Psi)^{S^*} := \Phi^{S^*} \to \Psi^{S^*} \tag{6.1.46}$$
$$(\forall a\Phi)^{S^*} := \forall a^{S^*}\Phi^{S^*} \tag{6.1.47}$$

Proposition 6.1.3. *If S satisfies $sc(M)$, then S^* satisfies $eq^*(M)$.*

Proof. One easily proves, by induction, that for all scheme Φ, $\Phi^{S^*} = \big(i(\Phi)\big)^S$. The only interesting case is $\Phi = I_a(\phi)$:

$$(\phi^{\iota(I_a)})^S = \phi^S(\iota(I_a))^S = \phi^{S^*} I_a^{S^*} = \big(I_a(\phi)\big)^{S^*} \tag{6.1.48}$$

Moreover, one can easily show by induction that, for any constraint $(a, B) \in ct(M)$, and any scheme variable ϕ occurring in a scheme in B, $a \notin s^{\triangleright}_{i(\phi)}$; hence, from the fact that, for all ϕ, $FV(\phi^S) \subseteq \{a_0^S, \dots, a_n^S\}$, it follows that, for any constraint $(a, (\Phi_1, \dots, \Phi_k)) \in ct(M)$, $a^S \notin FV(\Phi_1^S) \cup \dots \cup FV(\Phi_k^S)$ and, as a consequence, $a^{S^*} \notin FV(\Phi_1^{S^*}) \cup \dots \cup FV(\Phi_k^{S^*})$. □

Some properties of $sc(M)$ A derivation d_M^{sc} can be obviously defined starting from d_M^* and replacing every scheme Φ by $i(\Phi)$.

A declaration $(x : \Phi)$ occurring in d_M^{sc} is said *non trivial* if Φ is not of the form $\forall a\phi$, where ϕ is a scheme variable. One can easily see that a non trivial declaration $(x : \Phi)$ in d_M^* must come from a trivial one $(x : \phi)$ in d_M and an equation $\phi = \Phi$ coming from a redex. Indeed one can prove the following:

Lemma 6.1.2. *If M is normal, then all declarations $(x : \Phi)$ occurring in d_M^{sc} are trivial.*

Proof. Induction on the construction of d_M^*. □

Let M be a λ-term and M' be a subterm of M; let us say that M' *has scheme* Φ if a sequent $\Gamma \vdash M' : \Phi$ occurs in the derivation d_M^* (or, equivalently, in d_M^{sc}). Moreover, if Φ is a scheme of the form

$$\forall b_1 \Phi_1 \to \forall a_2(\forall b_2 \Phi_2 \to \dots \to \forall a_k(\forall b_k \Phi_k \to \forall a_{k+1}\phi) \dots) \tag{6.1.49}$$

then a term M is *faithful to* Φ if it is of the form $\lambda z_1. \dots .\lambda z_k.M'$.

Let us define a well-founded order $M' \prec M''$ on the subterms of M as follows: $M' \prec M''$ holds if M'' has a free variable z with a non trivial declaration $(z : \Phi)$, M' has scheme Φ and there exists a redex $(\lambda z.J)M'$ with J containing M''. To see that $\prec$ is well-founded, one uses the fact that, by lemma (6.1.2), the declarations in the conclusion of d_M^* are all trivial. The definition below, so as the proof of proposition (6.1.4), will be given by induction on the order $\prec$.

Definition 6.1.2 (applied terms, becoming)**.** *Let us call a subterm M' of M* applied *if it occurs in a subterm N of the form $M'N'$, and* non applied *if it doesn't.*

If M' is non applied then we say that M' becomes N in M under reduction *if M' contains some free variables $x_1, \dots, x_n$ such that, for all $1 \leq i \leq n$, M contains a redex of the form $(\lambda x_i.J)Q_i$, where M' is a subterm of J, and $N = M'[Q_1'/x_1, \dots, Q_n'/x_n]$, where, for all $1 \leq i \leq n$, Q_i becomes Q_i' under reduction.*

Lemma 6.1.3. *Let M' be a subterm of a term M of scheme Φ and let $x_1, \dots, x_n$ be the free variables of M' which have non trivial declarations $(x_1 : \Phi_1), \dots, (x_n : \Phi_n)$ in d_M^*. Then, if $Q_1, \dots, Q_n$ are terms which are faithful, respectively, for $\Phi_1, \dots, \Phi_n$, the term $M'[Q_1/x_1, \dots, Q_n/x_n]$ is faithful to Φ.*

Proof. Induction on M'. □

In order to study the behavior of redexes in M we introduce the notion of *redex pair*:

Definition 6.1.3 (redex pair)**.** *Given a scheme variable ϕ, a* head redex pair *of base ϕ in sc(M) is a pair of equations*

$$\phi^{F_{a_1}\dots F_{a_k}} = \Psi_1^{G_{a_1}} \to \dots \to \Psi_h^{G_{a_h}} \tag{6.1.50}$$

$$(\Phi_1 \to \dots \to \forall a_1 \phi \to \Phi_p)^{F_{b_1}\dots F_{b_h}} = \Theta \tag{6.1.51}$$

where $\{a_1, \dots, a_k\} \cap \{b_1, \dots, b_h\} = \emptyset$ and $b_1 \rhd a_1$.

A body redex pair *of base ϕ in sc(M) is a pair of equations*

$$\phi^{F_{a_1}\dots F_{a_k}} = \Psi_1^{G_{a_1}} \to \dots \to \Psi_h^{G_{a_h}} \tag{6.1.52}$$

$$\Theta^{F_{v'_1}\dots F_{v'_h}} = \Phi_1^{G_{b_1}} \to \dots \to (\Psi_1 \to \dots \to \forall a_v \phi \to \Psi_p)^{G_{b_k}} \to \Phi_{k+1}^{G_{b_{k+1}}} \tag{6.1.53}$$

where $b_1 \rhd a_1$.

Remark 6.1.2. *From a head redex pair (6.1.50) one can guess the existence of a redex $(\lambda x.P)Q$ in M, where Q has scheme $\forall a_1(\Phi_1 \to \dots \to \forall a_k \phi \to \Phi_k)$ and P contains a subterm of the form $(x)P_1 \dots P_k$.*

From a body redex pair (6.1.52) one can guess the existence of a redex $(\lambda x.P)Q$ in M, where Q has scheme $\forall a_1(\Psi_1 \to \dots \to \forall a_{p-1}\phi \to \Psi_p)$ and P contains a subterm of the form $(y)P_1 \dots P_{k-1}x$, where y is a variable declared of scheme Θ.

Indeed these are the only two possible cases of redex in M, if one excludes "weakening" redexes $(\lambda x.P)Q$, where x does not occur in P.

Remark 6.1.3. *If M is normal, then sc(M) contains no redex pairs and all its equations are of the form*

$$\phi^{F_{a_1}\dots F_{a_k}} = \Psi_1^{F_{b_1}} \to \dots \to \Psi_k^{F_{b_k}} \to \psi \tag{6.1.54}$$

in particular, if $\phi^{F_{a_1}\dots F_{a_k}}$ and $\phi^{F_{a'_1}\dots F_{a'_{k'}}}$, for certain $k, k' \in \mathbb{N}$, occur in the lefthand side of two equations in sc(M), then one must have $a_1 = a'_1, \dots, a_{min(k,k')} = a'_{min(k,k')}$.

Conversely, one can easily show by induction on d^*_M the following fact: suppose a head redex pair occurs in sc(M) and suppose the two schemes $\Phi_1^{F_{a_1}\dots F_{a_k}}$ and $\Phi_2^{F_{b_1}\dots F_{b_h}}$ (hence $a_1 \neq b_1$) occurring in the lefthand side of the equations of the redex pairs are not disjoint (so they have at least a scheme variable ϕ in common); then, for any other occurrence in sc(M) of a scheme $\Psi^{F_{c_1}\dots F_{c_l}}$ which contains ϕ, one has that either $a_1 = c_1, \dots, a_{min(k,l)} = c_{min(k,l)}$, either $b_1 = c_1, \dots, b_{min(h,l)} = c_{min(h,l)}$. Intuitively, this means that in sc(M) there can occur at most two "incompatible" (i.e. containing substitution variables whose nodes are labeled by disjoint sets of sequence variables, see below) occurrences of the substitution scheme ϕ.

Proposition 6.1.4. *Let M be a λ-term. If Q is a non applied sub term of M having scheme*

$$\Phi = \forall b_1 \Phi_1 \to \forall a_2(\forall b_2 \Phi_2 \to \dots \to \forall a_k(\forall b_k \Phi_k \to \forall a_{k+1}\phi)\dots) \tag{6.1.55}$$

then Q becomes Q' in M under reduction, where Q' is faithful to Φ.

Proof. We prove the following fact by induction on the complexity of λ-terms: if Q is non applied and has scheme $\Phi_1 \to \Phi_2$ in d^*_M, then either Q is faithful, either Q becomes faithful under reduction.

We argue by induction on the order $\prec$. If Q is not faithful, since it is not applied it must contain free variables $x_1, \dots, x_n$ with non trivial declarations $(x_1 : \Phi_1), \dots, (x_n : \Phi_n)$ and, for each $1 \leq i \leq n$, there must exist a non applied term Q_i of scheme Φ_i and a redex of the form

$(\lambda x_i.J_i)Q_i$, where J_i contains Q. Now, since, for all $1 \leq i \leq n$, $Q_i \prec Q$ we can apply the induction hypothesis: for all $1 \leq i \leq n$, the Q_i are faithful to Φ_i and Q becomes under reduction the term $Q[Q_1/x_1, \ldots, Q_n/x_n]$, which is faithful by lemma (6.1.3).

□

Corollary 6.1.1. *Suppose that a (head or body) redex pair of base ϕ occurs in $sc(M)$. Then M reduces to a term M' containing a redex of the form $(\lambda x.P)\lambda z_1.\ldots.\lambda z_k.Q$, with $k \geq p$.*

6.2 Vicious circles and typing

In this section we investigate the mathematical structure underlying the recursive equations arising from auto-applications or, in Russell's terminology, "vicious circles". First we recall a "geometrical" investigation of the vicious circles arising from type inference which arises from the so-called Patterson-Wegman algorithm ([PW78]): to the recursive equations arising from the unification algorithm there correspond cyclic paths in a "unification graph"; following this interpretation, in [LC89], some properties of typability in System F are obtained from an analysis of the structure of those graphs.

Next we recall some results, which can be found in [Mal90, Mal92], on the untypability of some pure λ-terms in System F based on a notion of "compatibility" between the constraints induced by recursive equations. The use of this combinatorial notion allows to prove the untypability of certain λ-terms without relying on the reducibility theorem.

Finally Malecki's notion of compatibility is generalized and it is proved (theorem (6.2.1)) that a term forcing two incompatible constraints cannot be normalizing (hence a strong normalizing λ-term cannot force incompatible constraints). This result constitutes the first step towards the abstract characterization of typability that is obtained in the next section.

6.2.1 The geometry of vicious circles

The problem of autoapplications In subsection (6.1.1) it was shown that the first-order unification algorithm, when applied to simple types, fails only when an occur-check is detected, i.e. a recursive equation of the form $\phi = \Phi$, where ϕ occurs in Φ, is produced. Recursive equations come from terms containing an *auto-application* of a variable x, i.e. containing subterms of the form

$$(x)P_1 \ldots P_k x P_{k+1} \ldots P_n \tag{6.2.1}$$

The polymorphic type discipline allows to type some λ-terms containing auto-applications. Indeed, in addition to the already mentioned term δ (containing one auto-application) also the terms

$$(\lambda x.\lambda y.(x)yx)\lambda z.(z)z \tag{6.2.2}$$

$$(\lambda x.(x)xx_1xxx_2)\lambda y_1.\lambda y_2.\lambda y_3.\lambda y_4.(y_4)y_3y_4yy_1y_2 \tag{6.2.3}$$

are typable in F (see [Mal90]), though they contain more than one auto-applications.

On the contrary, the term $(\lambda x.(x)x)\lambda y.\lambda z.(y)zy$, though being strongly normalizable, is not typable in F and the terms $(\lambda x.(x)x)\lambda x.(x)x$ and $(\lambda u.(u)u)\lambda x.\lambda y.(y)xy\lambda x.\lambda y.(y)xy$ are not even normalizable.

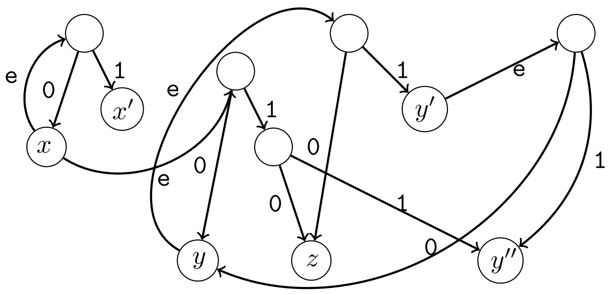
First-order unification and vicious circles In this paragraph we investigate the vicious circles arising in the system $eq^*(M)$ (or, equivalently, $sc(M)$) from the viewpoint of first-order unification. Indeed, by deleting quantifiers, sequence variables and substitution variables in all equations the system $eq^*(M)$ collapses into the system of equation over schemes that one obtains from the first-order type inference technique described in subsection (6.1.1).

The interest of investigating second order systems from a first order perspective is twofold: first, obviously, from a solution to the collapsed system one straightforwardly retrieves a solution to the second order system; second, this collapse enables the use of first-order unification in order to investigate the structure of the recursive (i.e. non unifiable) equations.

In [LC89] a geometrical interpretation of unification (based on the algorithm first described in [PW78]) is given by associating, with a system E of equations between first order terms, a graph $\mathcal{U}(E)$ which is invariant under the transformations defining the unification algorithm. In particular, to the recursive equations derivable from E there correspond simple cycles in the "unification graph" $\mathcal{U}(E)$, so that unifiability can be investigated as an acyclicity problem.

In order to define the unification graph, we first associate to a system E a *dag* (directed acyclic graph) $\mathcal{G}(E)$ defined as follows: first, to each term t occurring in an equation in E we associate its *dag* representation $\mathcal{G}(t)$[3] Then we consider the union graph of all the $\mathcal{G}(t)$, i.e. the graph having as set of vertices the union of the sets of vertices of the $\mathcal{G}(t)$ (remark that these sets are in general not disjoint) and as set of oriented edges the union of the sets of oriented edges of the $\mathcal{G}(t)$. Finally, we obtain the graph $\mathcal{G}(E)$ by adding to the union graph, for any equation $t = u \in E$, an oriented edge (called *equational edge*), with label e, from the root of t to the root of u.

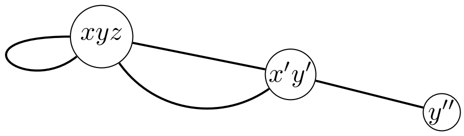
For instance, the graph $\mathcal{G}(E)$ for the system $E = \{x = f(x, x'), y = f(z, y'), y' = f(y, y''), x = f(y, f(z, y''))\}$ is the one below (as reported in [LC89]):



(6.2.4)

The unification graph $\mathcal{U}(E)$ is defined as the quotient of $\mathcal{G}(E)$ under the smallest, downward closed, equivalence relation on vertices generated by the equational edges. Hence, the definition of $\mathcal{U}(E)$ induces an equivalence relation $\sim$ over the variables occurring in E. In the case above, the unification graph is the following:

[3] This is a labeled *dag* (see [PW78]) that can be defined recursively as follows: if $t = x$ then $\mathcal{G}(t)$ has a node for the variable x and no directed edge; if $t = f(t_1, t_2)$ then the nodes of $\mathcal{G}(t)$ are given by the nodes of $\mathcal{G}(t_1)$, the nodes of $\mathcal{G}(t_2)$ (where these two sets might be not disjoint if t_1 and t_2 have some variable in common) and a new node n for the occurrence of the function symbol f; the directed edges of $\mathcal{G}(t)$ are given by the union of the directed edges of $\mathcal{G}(t_1)$ and those of $\mathcal{G}(t_2)$ plus a directed edge with label 0 from n to the root of $\mathcal{G}(t_1)$ and a directed edge with label 1 from n to the root of $\mathcal{G}(t_2)$.



(6.2.5)

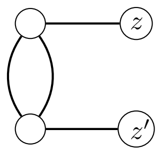
In [LC89] it is shown that, if a system E is transformed into E' by means of an identity, decomposition or variable elimination step (see subsection (6.1.1)), then the graph $\mathcal{U}(E')$ is the same as $\mathcal{U}(E)$. In this sense the unification graph is an invariant of the unification algorithm.

Moreover, it is proved there that the simple cycles in $\mathcal{U}(E)$ correspond exactly to the recursive equations that can be derived from E by applying standard equality rules plus other rules which translate the transformations defining the unification algorithm. Hence simple cycles in $\mathcal{U}(E)$ are in bijection with the recursive equations derivable from E. This result allows to speak in a very general and abstract way of the set of "vicious circles" which are induced by a system of first order equations (which are not limited to the recursive equations in the system, but include also the recursive equations derivable from the system).

When considering first-order systems arising from polymorphic type inference we are not interested in the existence of unifiers. Indeed, since such systems generally contain vicious circle, they have no unifiers. This does not impedes to investigate the structure of those vicious circles from the perspective of first-order unification.

First recall from subsection (6.1.1) that every sequence of the transformations of the unification algorithm terminates, independently of the fact that it finds a unifier of the system. Let us call *irreducible* a system to which no transformation can be applied; the termination of every branch of the unification algorithm implies then that every system can be transformed into a reducible system.

Observe that, as the application of the transformations is performed in a non deterministic way, the same system can be reduced into distinct irreducible systems. For instance, the system $E = \{(x = f(y,z), y = f(x,z')\}$ can be reduced to the two distinct systems $E_1 = \{x = f(f(x,z'),z)\}$ and $E_2 = \{y = f(f(y,z),z')\}$. However, one has that $\mathcal{U}(E) = \mathcal{U}(E_1) = \mathcal{U}(E_2)$, where $\mathcal{U}(E)$ is the graph below:



(6.2.6)

The unification graph allows then to capture the properties which are shared by all of those systems (in particular the equivalence classes of variables and the number of simple cycles).

We end this subsection by proving a result that will be used in subsection (6.3.2). First, since any simple cycle c in $\mathcal{U}(E)$ corresponds to a recursive equation $x = t$ derivable from E, we can associate a pair (x_c, σ_c) made of a variable $x_c = x$ and an *address* σ_c, i.e. a finite sequence of 0 and 1 corresponding to the leftmost path in $\mathcal{G}(t)$ from the root to x. Two cycles c_1, c_2 are said *coherent* when either $x_{c_1} \neq x_{c_2}$, either σ_{c_1} is not a subsequence of σ_{c_2} nor σ_{c_2} is a subsequence of σ_{c_1}.

Let us call a *splitting pair* a pair (c_1, c_2) of coherent cycles such that $x_{c_1} = x_{c_2}$. Geometrically, splitting pairs are pairs of cycles passing through the same vertex. The condition of coherence assures that we can "decompose" the two cycles, by applying to the system the substitution θ which sends x on a *linear* (i.e. all variables occur exactly once) term t defined with fresh variables and such that the two addresses σ_{c_1} and σ_{c_2} are occupied by two (distinct) variables.

More formally, let us first define, for an address σ, a linear term t_σ as follows:

- if $\sigma = \epsilon$ is the empty sequence, then $t_\sigma = x$, where x is a fresh variable;
- if $\sigma = e * \sigma'$, where $e \in \{0,1\}$, then t_σ is $f(t_{\sigma'}, x)$, if $e = 0$, and $f(x, t_{\sigma'})$ if $e = 1$, where x is a fresh variable.

Now, given two distinct addresses σ_1, σ_2, we can define the term t_{σ_1,σ_2} as a *most general unifier* of t_{σ_1} and t_{σ_2} (one can easily verify that such *mgu* is always defined and moreover t_{σ_1,σ_2} is still linear).

Hence, given a splitting pair (c_1, c_2), the decomposition of the system corresponds to the result of applying, to all equations in E, the substitution $\theta(x) = t_{\sigma_{c_1},\sigma_{c_2}}$. The new system $E\theta$ has then the following features:

1. for every simple cycle c in E there exists exactly one corresponding simple cycle $c\theta$ in $E\theta$;
2. every simple cycle in $E\theta$ is of the form $c\theta$ for exactly one cycle c in E;
3. to every splitting pair (c_1', c_2') in E, except for the pair (c_1, c_2), there corresponds a splitting pair $(c_1'\theta, c_2'\theta)$ in $E\theta$ and viceversa, every splitting pair in $E\theta$ comes from a splitting pair in E;
4. the cycles $c_1\theta$ and $c_2\theta$ have no variable in common.

The facts $1 - 3$ are immediate consequences of the fact that the equations in $E\theta$ have the form $t\theta = u\theta$, where $t = u \in E$ and where θ introduces linear terms with fresh variables. The fact 4 comes from the fact that the cycles $c_1\theta$ and $c_2\theta$ pass now through distinct variables (since the paths σ_{c_2} and σ_{c_2} correspond to distinct fresh variables in $t_{\sigma_{c_1},\sigma_{c_2}}$).

In other words, the splitting operation transforms the system E into a system $E\theta$ having the same number of cycles and the same type of intersections between cycles, except for the two cycles c_1, c_2 which no more intersect.

As a consequence of the facts $1 - 4$ and the fact that the number of splitting pairs strictly decreases after splitting, after a finite iteration of the splitting operation one obtains a system with no splitting pairs.

In the next paragraph it will be shown that the splitting operation is legitimate: from the viewpoint of polymorphic typing this operation transforms the system into an equivalent one.

6.2.2 Recursive equations and typing constraints

Malecki's lemma Since all normal terms are typable in System F, the problem with typability must arise from the occurrence of a redex $(\lambda x.P)Q$, which might make two distinct "vicious circles" interact; the term $(\delta)\delta$, where $\delta = \lambda x.(x)x$ provides a well-known example. A second example is the term $(\omega)z\omega$, where ω is $\lambda x.\lambda y.(y)xy$.

The existence of an auto-application in a term M corresponds to the existence of a recursive equation in $sc(M)$, i.e. an equation of the form

$$\Phi^{F_{a_1}\cdots F_{a_k}} = \Phi_1 \to \Phi_2 \to \cdots \to \forall b_l \Phi^{F'_a} \to \cdots \to \Phi_k \tag{6.2.7}$$

Now a simple argument shows that this equation forces a constraint on the form of the types $\sigma, \tau_1, \ldots, \tau_k$ which should satisfy it. In order to describe this argument we introduce *addresses* in a type: for each type σ of System F and positive integer $k \geq 1$ the *address* $\Pi_k(\sigma)$ is (partially) defined by induction as follows:

- if $\sigma \equiv \alpha$, then $\Pi_1(\sigma) = \sigma$ and $\Pi_{k+1}(\alpha) \uparrow$;

- if $\sigma \equiv \tau \to \rho$, then $\Pi_1(\sigma) = \tau$ and $\Pi_{k+1}(\sigma) = \Pi_k(\rho)$;
- if $\sigma \equiv \forall\alpha\tau$ then $\Pi_k(\sigma) = \Pi_k(\tau)$.

Let $lr(\sigma) = max\{k|\Pi_{k+1}(\sigma)\downarrow\}$ and $H(\sigma) = \Pi_{lr(\sigma)+1}(\sigma)$. Finally, let's define, for $k \geq 0$, the *head* $H^k(\sigma)$ of σ at address $k+1$ as

$$H^k(\sigma) = \begin{cases} H^k(\Pi_{k+1}(\sigma)) & \text{if } k \leq lr(\sigma) \\ H(\sigma) & \text{else} \end{cases} \tag{6.2.8}$$

If $H^k(\sigma)$ is the variable α, we say (as in [Urz97]) that α *owns the path* k.

Now, if the types $\sigma, \tau_1, \ldots, \tau_k$ satisfy equation (6.2.7), then for some substitutions θ, θ', one has

$$\sigma\theta = \Pi_l(\sigma)\theta' \tag{6.2.9}$$

where τ is the type $\tau_1 \to \forall\alpha_2(\tau_2 \to \cdots \to \forall\tau_k)$. The lemma below says that, in that case, the *head* at address k in σ must be in the domain of θ'.

Lemma 6.2.1 ([Mal90]). *Let σ be a type which satisfies an equation of the form*

$$\tau\theta = \Pi_k(\tau)\theta' \tag{6.2.10}$$

for certain substitutions θ, θ' and a positive integer k. Then $H^{k-1}(\sigma) \in dom(\theta')$.

Proof. Let $H^{k-1}(\sigma)$ be α; let us define, for any type σ, a notion of *k-depth* $lr^k(\sigma)$, for $k \geq 0$, as follows:

$$lr^k(\sigma) := \begin{cases} lr^k(\Pi_{k+1}(\sigma)) + k + 1 & \text{if } k \leq lr(\sigma) \\ lr(\sigma) & \text{else} \end{cases} \tag{6.2.11}$$

Clearly, if $H^k(\sigma) \notin dom(\theta)$, then $H^k(\sigma\theta) = H^k(\sigma)$ and $lr^k(\sigma\theta) = lr^k(\sigma)$; hence, if we suppose $H^{k-1}(\sigma) \notin dom(\theta')$, we get

$$lr^{k-1}(\Pi_k(\sigma)) = lr^{k-1}(\Pi_k(\sigma)\theta') = lr^{k-1}(\sigma\theta) \geq lr^{k-1}(\sigma) \tag{6.2.12}$$

where in the last step we used the remark that a substitution cannot decrease k-depth. From the fact that $lr^{k-1}(\sigma) = lr^{k-1}(\Pi_k(\sigma)) + k$ we get then a contradiction.

□

Let us see a simple application of this lemma: the system $sc((\delta)\delta)$ contains the two equations

$$\phi^{F_a} = \phi^{F'_a} \to \psi \tag{6.2.13}$$

$$(\forall a\phi \to \psi)^{F_b} = (\forall a\phi \to \psi)^{F'_b} \to \chi \tag{6.2.14}$$

Hence, is S satisfies $sc(M)$, then, by letting $\sigma = \phi^S, \tau = \psi^S$ and $\rho = \chi^S$, one must have (where $\overline{\alpha}$ stands for a finite sequence of type variables)

$$\sigma\theta^S_{F_a} = \sigma\theta^S_{F'_a} \to \tau \tag{6.2.15}$$

$$(\forall\overline{\alpha}\sigma \to \tau)\theta^S_{F_b} = \forall\overline{\beta}(\forall\overline{\alpha}\sigma \to \tau)\theta^S_{F'_b} \to \rho \tag{6.2.16}$$

where the substitutions $\theta^S_{F_a}$ and $\theta^S_{F_b}$ have disjoint domain. Now from lemma (6.2.1) it follows that $H^0(\sigma) \in dom(\theta^S_{F_a}) \cap dom(\theta^S_{F_b}) = \emptyset$, which is absurd.

The argument above shows that the combinator $(\delta)\delta$ is not typable in System F without relying on the reducibility theorem (4.1.1) (a similar argument can be found in [GV09] to show that $(\delta)\delta$ cannot be typed in System U^-). Indeed we are going to generalize this form of argument, in order to obtain results about the untypability of certain (not normalizing) λ-terms, without relying on assumptions about reducibility.

The transport of head constraints Let us call a triple $\kappa = (\phi, k, A)$, where ϕ is a scheme variable, $k \geq 1$ a positive integer and A a finite non empty set of sequence variables occurring in $s_{\phi}^{\triangleright}$, a *head constraint*[4]. Intuitively the head constraint (ϕ, k, A) says that, for all ground substitution S, $H^{k-1}(\phi^S) \in a_1^S \cup \cdots \cup a_n^S$, where $A = \{a_1, \ldots, a_n\}$. Two constraints $\kappa = (\phi, k, A)$ and $\kappa' = (\psi, h, B)$ are *incompatible* if $\phi = \psi$, $k = h$ and $A \cap B = \emptyset$.

We first want to study how, following lemma (6.2.1), constraints (ϕ, k, A) can be forced by a λ-term.

Let us say that a scheme variable ϕ occurs in a scheme Φ at address k if either $\Phi = \phi$, either Φ is of the form

$$\forall a_1(\Phi_1 \to \forall a_2(\Phi_2 \to \cdots \to \forall a_k(\forall b_k \phi \to \Phi_{k+1}) \ldots)) \tag{6.2.17}$$

The simplest case in which a constraint (ϕ, p, A) is forced is when an equation of the form

$$\Phi^{F_{a_1} \ldots F_{a_k}} = \Psi_1^{F_{b_1}} \to \cdots \to \Psi_{p-1}^{F_{b_{p-1}}} \to \Phi^{F'_{a_1}} \to \cdots \to \Psi_k^{F_{b_k}} \to \psi \tag{6.2.18}$$

where ϕ occurs in Φ at address p, occurs in $sc(M)$; however, we have to consider also the fact that, if an equation of the form $\Phi^{F_a} = \Psi$ occurs in $sc(M)$, then a constraint on a subscheme of Φ (resp. of Ψ) can be "transported" to a subscheme of Ψ (resp. of Φ), as implied by the following lemma:

Lemma 6.2.2 (transport of constraints, [Mal90])**.** *Suppose that two types σ, τ satisfy an equation $\sigma\theta = \tau\theta'$, for certain substitutions θ, θ', and that $H^k(\sigma) \notin dom(\theta)$. Then two cases arise:*

1. *if $H^k(\sigma) \in BV(\sigma)$, then either $H^k(\tau) \in dom\theta'$, either $H^k(\tau) \equiv H^k(\sigma)$;*
2. *if $H^k(\sigma) \in FV(\sigma)$, then either $H^k(\tau) \in dom\theta'$, either $H^k(\tau) = H^k(\sigma)$.*

Proof. Both results come from the remark that $H^k(\sigma) = H^k(\sigma\theta) = H^k(\tau\theta')$. □

Following lemma (6.2.2) we get to the following definition:

Definition 6.2.1 (forcing constraints)**.** *Let M be a λ-term. Given a simple substitution scheme Φ, an integer k and a finite set of sequence variables A, M* forces *the constraint κ if one of the following holds:*

i. $\kappa = (\phi, p, \{a_1\})$ and $sc(M)$ contains the equation

$$\Phi^{F_{a_1} \ldots F_{a_k}} = \Psi_1^{F_{b_1}} \to \cdots \to \Psi_{p-1}^{F_{b_{p-1}}} \to \Phi^{F'_{a_1}} \to \cdots \to \Psi_k^{F_{b_k}} \to \psi \tag{6.2.19}$$

where ϕ occurs in Φ at address p;

ii. $\kappa = (\phi, k, A)$ and there exist a scheme variable ψ and simple schemes $\Phi, \Psi, \Phi_1, \ldots, \Phi_{k-1}, \Phi_{k+1}$ such that $sc(M)$ contains the equation

$$\Psi^{F_{a_1} \ldots F_{a_k}} = \Phi_1^{F_{b_1}} \to \cdots \to \Phi_{k-1}^{F_{b_{k-1}}} \to \Phi^{F_{b_k}} \to \Phi_{k+1}^{F_{b_{k+1}}} \tag{6.2.20}$$

where ϕ occurs in Φ at address k, ψ occurs in Ψ at address k and the following holds:

- *$A = \{b_k\} \cup C' \cup D'$ and M forces $(\psi, k, C \cup D)$, where for any sequence variable $c \in C$, $c \triangleright b_k$ and for any sequence variable $d \in D$, $b_k \triangleright d$ and the sets C', D' are defined as follows: $C' \subseteq C$ contains the $c \in C$ such that $c \in s_{\phi}^{\triangleright}$; D' contains, for any $d \in D$, the sequence variable d', if it exists, which occurs in Ψ at the same position as d in Φ;*

[4] Unless there is ambiguity with the previously defined notion of constraint, we will simply call this a "constraint".

iii. $\kappa = (\psi, k, A)$ *and there exist a scheme variable* ϕ *and simple schemes* $\Phi, \Psi, \Phi_1, \ldots, \Phi_{k-1}, \Phi_{k+1}$ *such that* $sc(M)$ *contains the equation*

$$\Psi^{F_{a_1} \cdots F_{a_k}} = \Phi_1^{F_{b_1}} \to \cdots \to \Phi_{k-1}^{F_{b_{k-1}}} \to \Phi^{F_{b_k}} \to \Phi_{k+1}^{F_{b_{k+1}}} \tag{6.2.21}$$

where ϕ *occurs in* Φ *at address* k*,* ψ *occurs in* Ψ *at address* k *and the following holds:*

- $A = \{a_1\} \cup C' \cup D'$ *and* M *forces* $(\phi, k, C \cup D)$*, where for any sequence variable* $c \in C$*,* $c \rhd b_k$ *and for any sequence variable* $d \in D$*,* $b_k \rhd d$ *and the sets* C', D' *are defined as follows:* $C' \subseteq C$ *contains the* $c \in C$ *such that* $c \in s_\psi^{\rhd}$*;* D' *contains, for any* $d \in D$*, the sequence variable* d'*, if it exists, which occurs in* Φ *at the same position as* d *in* Ψ*;*

Accordingly, we define the notion of *constraint path*: this is a finite sequence of pairs $(\phi_i, \kappa_i)_{1 \leq i \leq n}$, where the ϕ_i are distinct scheme variables and the $\kappa_i = (\phi_i, k_i, A_i)$ are constraints such that:

- M forces the constraint $\kappa_1 = (\phi_1, k_1, A_1)$ according to clause *i.* of the definition (6.2.1);
- for all $1 \leq i \leq n-1$, the constraint κ_{i+1} is transported from the constraint κ_i according to clause *ii.* or *iii.* of the definition (6.2.1).

Clearly M forces κ if and only if there exists a constraint path (of finite length k) leading to κ. In such case we say that M k-forces κ.

Example 6.2.1. *The reader can familiarize with the transport of constraints by computing the constraint paths of the (non normalizing) term below*

$$(\lambda u.(u)u)\big(\lambda y.((\lambda z.(z)z)\lambda x.(x)y)\big) \tag{6.2.22}$$

and showing that it induces two incompatible constraints.

A term is said *compatible* if it forces no incompatible constraints, and *incompatible* if it forces some. Thus one has the following proposition, which allows to prove the untypability of non compatible λ-terms without appealing to reducibility:

Proposition 6.2.1. *If* M *is not compatible, then it is not typable in System* F*.*

Proof. Suppose M forces two incompatible constraints $(\phi, k, A), (\phi, k, B)$ and S is a ground substitution which satisfies $sc(M)$. Let A^S, B^S denote, respectively, the union of all the sets of the type variables of the form a^S and b^S, for $a \in A$ and $b \in B$. Then one must have $H^{k-1}(\phi^S) \in A^S \cap B^S = \emptyset$, which is absurd. □

The combinator $(\delta)\delta$ induces two incompatible constraints, namely $(\phi, 1, a)$ and $(\phi, 1, c)$. $(\delta)\delta$ is a fixed point combinator, hence it is not normalizing. Since terms inducing incompatible constraints cannot be typed in System F, as a consequence of Malecki's lemma (6.2.1), it is natural to ask whether there exist normalizing λ-terms inducing incompatible constraints, and thus not normally typable. The main result of this section is theorem (6.2.1), which shows that a λ-term inducing incompatible constraints cannot be normalizing, providing a negative answer to this question.

A consequence of the results presented in this section is the following: if we look at the collapsed first-order system, two compatible constraints $(\phi, k, A), (\phi, h, B)$ correspond to a splitting pair. Now, if $A \cap B = \emptyset$, from lemma (6.2.1) we know that, if S is a ground substitution satisfying $sc(M)$, the two "paths" k, h in ϕ^S must be owned by distinct type variables. This remark justifies, from a collapsed viewpoint, the introduction of the terms $t_{\sigma_{c_1}, \sigma_{c_2}}$ in the previous paragraph and the appeal to the system $E\theta$ obtained by splitting. This technique of splitting compatible constraints will be used in subsection (6.3.2).

6.2.3 Incompatible constraints and untypable terms

In the last subsection we considered an example of an incompatible λ-term, the combinator $(\delta)\delta$, which is not normalizing (nor it has a head normal form). In this subsection we prove a first result on typability which shows in full generality that an incompatible λ-term cannot be normalizing. Remark that, as the λ-term $\lambda z.(z)(\delta)\delta$ shows, an incompatible λ-term can be not normalizing though having a head normal form.

If M forces two incompatible constraints $(\phi, k, A), (\phi, k, B)$, then M must contain a redex. Our aim is to show that such a redex has an infinite reduction path.

Theorem 6.2.1. *If M forces two incompatible constraints $(\phi, k, A), (\phi, k, B)$, then M is not normalizable.*

Proof. We proceed by induction on the sum $p+q$ of the lengths of the constraint paths leading to the two incompatible variable constraints (ϕ, k, A) and (ϕ, k, B). More precisely, we first show that in the case $p+q=2$ (since variable constraints have length at least 1) the term is not normalizable and then we show by induction on $p+q$ that by reducing M all variable paths can be reduced to length 1.

If $p+q=2$, then $sc(M)$ must contain a head redex pair

$$\phi^{F_{a_1}\dots F_{a_k}} = \Psi_1^{F_{c_1}} \to \dots \to \Psi_{k-1}^{F_{c_{k-1}}} \to \phi^{F'_{a_1}} \to \Psi_{k+1}^{F_{c_{k+1}}} \tag{6.2.23}$$

$$\Psi^{F_{b_1}\dots F_{b_k}} = \Psi_1^{F_{d_1}} \to \dots \to \Psi_{k-1}^{F_{d_{k-1}}} \to \Psi^{F'_{b_1}} \to \Psi_{k+1}^{F_{d_{k+1}}} \tag{6.2.24}$$

where $\Psi = \Phi_1 \to \forall b_2(\Phi_2 \to \dots \to \forall b_{k-1}(\Phi_{k-1} \to \forall b_k(\forall a_1\phi \to \forall b_{k+1}\Phi_{k+1}))\dots)$.

From proposition (6.1.4) it follows then that M reduces to a term containing a redex M' of the form

$$(\lambda x.P)\lambda y_1.\dots.\lambda y_k.Q \tag{6.2.25}$$

where P contains a subterm of the form

$$(x)P_1 \dots P_{k-1}x \tag{6.2.26}$$

and Q contains a subterm of the form

$$(y_k)Q_1 \dots Q_{k-1}y_k \tag{6.2.27}$$

one can easily verify then that the head reduction of M' does not terminate.

If $p+q=n+3$, then one of the two constraints, say (ϕ, k, A), is either derived from a constraint (ψ, k, A') through an equation of the form

$$\Psi^{F_{a_1}\dots F_{a_k}} = \Phi_1^{F_{b_1}} \to \dots \to \Phi_{k-1}^{F_{b_{k-1}}} \to \Phi^{F_{b_k}} \to \Phi_{k+1}^{F_{b_{k+1}}} \tag{6.2.28}$$

(where ϕ occurs in Φ at address k and ψ occurs in Ψ at address k), either it is derived from a constraint (Ψ_k, k, A') through an equation of the form

$$\Phi^{F_{a_1}\dots F_{a_k}} = \Psi_1^{F_{b_1}} \to \dots \to \Psi_{k-1}^{F_{b_{k-1}}} \to \Psi_k^{F_{b_k}} \to \Psi_{k+1}^{F_{b_{k+1}}} \tag{6.2.29}$$

(where ϕ occurs in Φ at address k and ψ occurs in Ψ at address k). Here we just consider the first case. The second one can be proved in a similar way.

Now, either Ψ is ψ, either it is of the form

$$\forall a_1(\Psi_1 \to \dots \to \forall a_k(\forall b_k\psi \to \Psi_{k+1})\dots) \tag{6.2.30}$$

Suppose $\Psi = \psi$. By induction hypothesis, M reduces to a term M' such that $sc(M')$ still contains equation (6.2.28) and where (ϕ, k, B) and (ψ, k, A') are 1-forced. This means that $sc(M')$ contains the equations

$$\phi^{F_{a'_1}\dots F_{a'_k}} = \Theta_1^{F_{b'_1}} \to \dots \to \Theta_{k-1}^{F_{b'_{k-1}}} \to \phi^{F'_{a'_1}} \to \Theta_{k+1}^{F_{b'_{k+1}}} \tag{6.2.31}$$

$$(\Psi^*)^{F_{c_1}\dots F_{c_k}} = \Xi_1^{F_{d_1}} \to \dots \to \Xi_{k-1}^{F_{d_{k-1}}} \to (\Psi^*)^{F'_{c_1}} \to \Xi_{k+1}^{F_{d_{k+1}}} \tag{6.2.32}$$

where ϕ occurs in Φ at path k and ψ occurs in Ψ^* at path k. By proposition (6.1.4) M' reduces then to a term M'' containing a redex $(\lambda x.P)\lambda z_1.\dots.\lambda z_k.Q$ where x has scheme $\forall c_1 \Psi^*$, P contains a subterm of the form $(x)P_1 \dots P_{k-1}x$, Q contains a subterm of the form $(z_k)Q_1 \dots Q_{k-1}R$, where R has scheme $\forall e_1 \Phi$ and is of the form $\lambda y_1.\dots.\lambda y_k.R'$ and R' contains a subterm of the form $(y_k)R_1 \dots R_{k-1}y_k$.

M'' has thus the form

$$(\lambda x.\Big(\dots(x)P_1\dots P_{k-1}x\dots\Big))\lambda z_1.\dots.\lambda z_k.\Big(\dots(z_k)Q_1\dots Q_{k-1}(\lambda y_1.\dots.\lambda y_k.\big(\dots(y_k)R_1\dots R_{k-1}y_k\dots\big)\dots\Big) \tag{6.2.33}$$

and it reduces in two steps to a term containing

$$(\lambda y_1.\dots.\lambda y_k.\big(\dots(y_k)R'_1\dots R'_{k-1}y_k\dots\big))Q'_1\dots Q'_{k-1}(\lambda y_1.\dots.\lambda y_k.\big(\dots(y_k)R'_1\dots R'_{k-1}y_k\dots\big)) \tag{6.2.34}$$

which is not normalizable and moreover 1-forces the constraint (ϕ, k, A). Moreover, since the only reduced redexes involved equations (6.2.28) and (6.2.36), all other equations are left unchanged.

Suppose now the ψ occurs in Ψ at path k, i.e. that Ψ is $\forall a_1(\Psi_1 \to \dots \to \forall a_k(\forall b_k \psi \to \Psi_{k+1})\dots)$. Again, by induction hypothesis, M reduces to a term M' such that $sc(M')$ still contains equation (6.2.28) and where (ϕ, k, B) and (ψ, k, A') are 1-forced. This means that $sc(M')$ contains equations

$$\phi^{F_{a'_1}\dots F_{a'_k}} = \Theta_1^{F_{b'_1}} \to \dots \to \Theta_{k-1}^{F_{b'_{k-1}}} \to \phi^{F'_{a'_1}} \to \Theta_{k+1}^{F_{b'_{k+1}}} \tag{6.2.35}$$

$$\psi^{F_{c_1}\dots F_{c_k}} = \Xi_1^{F_{d_1}} \to \dots \to \Xi_{k-1}^{F_{d_{k-1}}} \to \psi^{F'_{c_1}} \to \Xi_{k+1}^{F_{d_{k+1}}} \tag{6.2.36}$$

where, again, ϕ occurs in Φ at path k. By proposition (6.1.4) M' reduces then to a term M'' containing a redex $(\lambda x.P)\lambda z_1.\dots.\lambda z_k.Q$ where x has scheme $\forall a_1 \Psi$, P contains a subterm of the form $(x)P_1 \dots P_{k-1}R$, where R has scheme $\forall e_1 \Phi$, is of the form $\lambda y_1.\dots.\lambda y_k.R'$ and R' contains a subterm of the form $(y_k)R_1 \dots R_{k-1}y_k$, and finally Q contains a subterm of the form $(z_k)Q_1 \dots Q_{k-1}z_k$.

M'' has thus the form

$$(\lambda x.\Big(\dots(x)P_1\dots P_{k-1}(\lambda y_1.\dots.\lambda y_k.\big(\dots(y_k)R_1\dots R_{k-1}y_k\dots\big))\dots\Big))\lambda z_1.\dots.\lambda z_k.\big(\dots(z_k)Q_1\dots Q_{k-1}z_k\dots\big) \tag{6.2.37}$$

which reduces in one step to a term containing

$$(\lambda z_1.\dots.\lambda z_k.\big(\dots(z_k)Q_1\dots Q_{k-1}z_k\dots\big))P_1\dots P_{k-1}(\lambda y_1.\dots.\lambda y_k.\big(\dots(y_k)R_1\dots R_{k-1}y_k\dots\big)) \tag{6.2.38}$$

which is not normalizable and 1-forces the constaint (ϕ, k, A). Again, since the only reduced redexes involved the equation (6.2.28), all other equations are left unchanged.

□

Contextual typing In the derivation d^{sc}_M the schemes assigned to the free variables of M are all of the form $\forall a\phi$, where ϕ is a sequence variable. We were indeed interested in the *typability problem* for λ-terms, i.e. the problem of finding an arbitrary type for the terms.

In order to consider also the *type checking* problem, we have to consider a *scheme assignment* $\mathcal{S}(x)$, which associates a (not necessarily linear) scheme $\mathcal{S}(x)$ with every free variable x of M. Indeed, for every free variable x, of scheme $\forall a\phi$, we must add to the system $eq(M)$ the equations $\forall a\phi = \mathcal{S}(x)$. Let us call this system $eq_{\mathcal{S}(x)}(M)$. One can define the systems $eq^*(M)$ and $sc_{\mathcal{S}(x)}(M)$ in the same way as in the subsections (6.1.2) and (6.1.3).

The following proposition is an immediate consequence of proposition (6.1.4):

Proposition 6.2.2. *Let M be a λ-term and $\mathcal{S}(x)$ a scheme assignment for M. Let $FV(M) = \{x_1, \dots, x_n\}$ and let, for all $1 \leq i \leq n$, M_i be a λ-term faithful to $\mathcal{S}(x_i)$. If Q is a non applied sub term of M having scheme*

$$\Phi = \forall b_1\Phi_1 \to \forall a_2(\forall b_2\Phi_2 \to \cdots \to \forall a_k(\forall b_k\Phi_k \to \forall a_{k+1}\phi)\dots) \tag{6.2.39}$$

then $Q[M_1/x_1, \dots, M_n/x_n]$, as a subterm of $M[M_1/x_1, \dots, M_n/x_n]$, becomes faithful to Φ under reduction.

Let M be a λ-term and $\mathcal{S}(x)$ a scheme assignment for M. Let us say that M $\mathcal{S}(x)$*-forces a constraint* κ if κ is obtained from $sc_{\mathcal{S}(x)}(M)$ by means of the clauses $i. - iii.$ of definition (6.2.2). M is *compatible relative to* $\mathcal{S}(x)$ if it forces no incompatible constraint. In subsection (6.3.3), given a term M with just a free variable x, we will consider the scheme assignment which associates $\mathcal{S}(x) = \forall a_1((\phi \to \phi) \to \forall a_2(\phi \to \phi)$, which allows to investigate a specific case of type-checking, i.e. whether $\lambda x.M$ can be given type $\mathbf{N} \to \mathbf{N}$.

6.3 A conjecture on typability

In this section we investigate, from a technical viewpoint, the following conjecture:

Conjecture 6.3.1. *Compatible λ-terms are typable in System U^-.*

The section is organized as follows: in subsection (6.3.1), we introduce the type inference of System U^-, as a generalization of the type inference introduced for System F, and we extend to the former system lemma (6.2.1) and the notion of head constraint. In subsection (6.3.2) we present some partial results and some examples intended to introduce the reader to the technical content of the conjecture. In particular we try to show why System U^- seems a good candidate for a combinatorial characterization of typability. Finally, in subsection (2.4.3), we discuss some technical consequences which would arise from it and which constitute its main motivations.

6.3.1 Type inference in System U^-

A syntax-directed type inference system for U^- can be devised, similarly to System F. The system below differs from (6.1.7) only in the definition of the relation $\leq$, which must take account of the β-equivalence of propositions. Moreover, we assume expressions σ, τ, ρ to be well-typed propositions and denote by $\overline{\alpha}$ finite sequences of constructor variables $\alpha_1, \dots, \alpha_n$, where, for

$1 \leq i \leq n$, α_i has a well-specified universe κ_i.

$$\boxed{\begin{array}{lll}
(\text{var}) & \Gamma, (x:\sigma) \vdash x : \tau \quad \sigma \leq \tau & \\
(\to I) & \dfrac{\Gamma, (x:\sigma) \vdash M : \tau}{\Gamma \vdash \lambda x.M : \forall \overline{\alpha}.\sigma \to \tau} & (\overline{\alpha} \text{ bindable in } \Gamma) \\
(\to E) & \dfrac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma \quad \tau \leq \rho}{\Gamma \vdash MN : \forall \overline{\alpha}.\rho} & (\overline{\alpha} \text{ bindable in } \Gamma)
\end{array}} \tag{6.3.1}$$

where $\overline{\alpha}$ denotes a finite (possibly empty) sequence of variables $\alpha_1, \ldots, \alpha_n$ (and $\forall\overline{\alpha}\sigma$ stands for $\forall\alpha_1 \ldots \forall\alpha_n\sigma$) and the relation $\sigma \leq \tau$ is the transitive closure of the relation $\leq_1$ defined by

$$\forall\alpha.\sigma \leq_1 \sigma' \qquad \Leftrightarrow \qquad \sigma' =_\beta \sigma[C/\alpha] \tag{6.3.2}$$

where σ and σ' are well-typed propositions, α has universe κ and C is a well-typed constructor of universe κ.

A *ground substitution* S for System U^- is defined similarly to the case of System F:

- a^S is a finite sequence (possibly empty) of constructor variables (of a certain universe) and, if $a \neq a'$, thene $a^S \cap a'^S = \emptyset$;
- F_a^S is substitution (i.e. a function which maps a constructor variable of universe κ into a well-typed constructor of universe κ) of domain a^S. This induces a map θ_t from substitution terms to substitutions defined as follows:
  - $\alpha\theta_a^S = \alpha$, for $\alpha \in a^S$;
  - $\alpha\theta^S_{F_v(t_1,\ldots,t_n)} := \alpha F_v^S \theta^S_{t_n} \ldots \theta^S_{t_1}$, for $\alpha \in a^S$.
- Φ^S is a well-typed proposition and one has

$$\big(\phi(t_1, \ldots, t_n)\big)^S = \phi^S \theta^S_{t_n} \ldots \theta^S_{t_1} \tag{6.3.3}$$

$$(\Phi \to \Psi)^S = \Phi^S \to \Psi^S \tag{6.3.4}$$

$$(\forall a.\Phi)^S = \forall a^S.\Phi^S \tag{6.3.5}$$

  where $\forall a^S.\sigma$ is $\forall\alpha_1 \ldots \forall\alpha_n\sigma$, where $a^S = \{\alpha_1, \ldots, \alpha_n\}$ (remark that a^S can be empty).

Since the system (6.3.1) is very similar to (6.1.7), the proposition (6.1.2) can be extended to System U^-:

Proposition 6.3.1 (principal typing derivations in System U^-)**.** *Let M be a λ-term, then the following two hold:*

i. if a ground substitution S satisfies $eq^*(M)$ and $ct(M)$, then d_M^S is a typing derivation in U^- of M in F;

ii. if d is a typing derivation in U^- of M in F, then there exists a ground substitution s satisfying $eq^*(M)$ and $ct(M)$ and such that $d = d_M^S$.

Proof. The two parts are straightforwardly proved by induction on the derivation d_M^S.

□

Compatibility Lemma (6.2.1) can be easily adapted to System U^-, by relying on the fact that all propositions in U^- have a (unique) normal form. Let us define *addresses* in a proposition: for each proposition σ and positive integer $k \geq 1$ the *address* $\Pi_k(\sigma)$ is defined by induction on the normal form σ' of σ as follows

- if $\sigma' \equiv (\alpha)\sigma_1 \dots \sigma_n$, then $\Pi_1(\sigma) = \alpha$ and $\Pi_{k+1}(\sigma) = \uparrow$;
- if $\sigma' \equiv (\sigma_1)\sigma_2$, then $\Pi_1(\sigma) = \uparrow$ and $\Pi_{k+1}(\alpha) = \uparrow$;
- if $\sigma' \equiv \tau \to \rho$, then $\Pi_1(\sigma) = \tau$ and $\Pi_{k+1}(\sigma) = \Pi_k(\rho)$;
- if $\sigma' \equiv \forall\alpha\tau$ then $\Pi_k(\sigma) = \Pi_k(\tau)$.

$lr(\sigma)$, $H(\sigma)$ and $H^k(\sigma)$ are defined (on the normal form σ' of σ) as in subsection (6.2.1).

Lemma 6.3.1. *Let σ be a proposition which satisfies an equation of the form*

$$\sigma\theta = \Pi_k(\sigma)\theta' \tag{6.3.6}$$

for certain substitutions θ, θ' and a positive integer k. Then $H^{k-1}(\sigma) \in dom(\theta')$.

Proof. The argument proceeds exactly like for lemma (6.2.1). □

Lemma (6.2.2), being a consequence of lemma (6.2.1), can be immediately transported to System U^-. Hence one obtains:

Proposition 6.3.2. *If M is incompatible, then it is not typable in U^-.*

6.3.2 Around the conjecture

In this subsection we present some partial results and some examples which will help the reader understand the content of conjecture (6.3.1) as well as some technical problems which must be solved in order to prove it.

The discussion is divided in three parts:

1. the simplification of the system $sc(M)$ by means of first-order unification can produce recursive equations of the form
$$\sigma\theta = \Pi_{k_1}(\Pi_{k_2}(\dots(\Pi_{k_n}(\sigma)\theta_n)\dots)\theta_2)\theta_1 \tag{6.3.7}$$
i.e. where the address associated with the vicious circle is given by a finite sequence of the form $(k_1, \dots, k_n)$, where $k_1, \dots, k_n$ are positive non zero integers. Hence, we introduce a generalized notion of constraint (ϕ, s, A), where s is a finite sequence of integers, to be interpreted as the infinite periodic path $s * s * s * \dots$. Theorem (6.3.3) assures that one can always find a set of generalized constraints which are pairwise compatible;
2. if two independent constraints $(\phi, k, A), (\phi, h, B)$, i.e. such that $A \cap B = \emptyset$, are forced by M, then, for any ground substitution satisfying $sc(M)$, one must have $H^{k-1}(\phi^S) \neq H^{h-1}(\phi^S)$, hence ϕ^S must contain at least $max\{k, h\}$ distinct non trivial addresses. We discuss a "splitting" operation which performs this decomposition at the level of schemes, in order to reduce $sc(M)$ to a "completely split" system $\mathbf{sc}(M)$, in which every scheme occurs in constraints whose sets of sequence variables are pairwise non disjoint. This "splitting" algorithm should implement the "splitting" of cycles that was sketched in subsection (6.2.1) in the case of first-order unification.
3. we investigate the definition of a ground substitution S satisfying a "completely split" system. This allows to indicate the role of impredicative universes to provide a uniform solution to recursive equations.

1. Generalized constraints In order to analyze the possible solutions to the system $sc(M)$, we must consider a generalized notion of constraint which naturally arises from the analysis of scheme equation systems. This notion is an immediate consequence of the lemma below, which generalizes lemma (6.3.1).

First we have to extend the notion of address: now an address s is a finite sequence $(p_0, \ldots, p_{n-1})$, with $n \geq 1$, of positive non zero integers; given a proposition σ, the *head* $H^s(\sigma)$ of σ at address s is defined as follows:

$$H^{k*s}(\sigma) = \begin{cases} H^{s*k}(\Pi_k(\sigma)) & \text{if } k \leq lr(\sigma) \\ H(\sigma) & \text{otherwise5} \end{cases} \tag{6.3.8}$$

Intuitively, $H^s(\sigma)$ looks for the variable which owns the infinite periodic path

$$p_0, \ldots, p_{n-1}, p_0, \ldots, p_{n-1}, \ldots \tag{6.3.9}$$

Remark that, if $p_0 = p_1 = \cdots = p_{n-1} = k$, then $H^s(\sigma)$ is just $H^k(\sigma)$.

We can now state the lemma which leads to the notion of generalized head constraint.

Lemma 6.3.2. *Let $\sigma_0, \ldots, \sigma_{n-1}$ be propositions satisfying a set of equations of the form*

$$\begin{aligned} \sigma_1\theta_0 &= \Pi_{k_{n-1}}(\sigma_0)\theta'_0 \\ \sigma_2\theta_1 &= \Pi_{k_0}(\sigma_1)\theta'_1 \\ &\vdots \\ \sigma_{n-1}\theta_{n-2} &= \Pi_{k_{n-3}}(\sigma_{n-2})\theta'_{n-2} \\ \sigma_0\theta_{n-1} &= \Pi_{k_{n-2}}(\sigma_{n-1})\theta'_{n-1} \end{aligned} \tag{6.3.10}$$

for certain substitutions $\theta_0, \theta'_0, \ldots, \theta_{n-1}, \theta'_{n-1}$ and integers $k_0, \ldots, k_{n-1}$. Then, one of the following holds:

$$\begin{aligned} H^{(k_{n-1},k_0,\ldots,k_{n-2})}(\Pi_{k_{n-1}}(\sigma_0)) &\in dom(\theta'_0) \cup \cdots \cup dom(\theta'_{n-1}) \\ H^{(k_0,\ldots,k_{n-1})}(\Pi_{k_0}(\sigma_1)) &\in dom(\theta'_0) \cup \cdots \cup dom(\theta'_{n-1}) \\ &\vdots \\ H^{(k_{n-2},k_{n-1},k_0,\ldots,k_{n-3})}(\Pi_{k_{n-2}}(\sigma_{n-1})) &\in dom(\theta'_0) \cup \cdots \cup dom(\theta'_{n-1}) \end{aligned} \tag{6.3.11}$$

Proof. Let us suppose that all the conditions (6.3.11) are false. We define a notion of *$s-$depth* $lr^s(\sigma)$, for σ a type and s a finite (non empty) sequence of positive non zero integers, as a "cyclic" generalization of the notion of k-depth:

$$lr^{k*s}(\sigma) := \begin{cases} lr^{s*k}(\Pi_{k+1}(\sigma)) + k + 1 & \text{if } k \leq lr(\sigma) \\ lr(\sigma) & \text{otherwise} \end{cases} \tag{6.3.12}$$

Clearly, if $H^s(\sigma) \notin dom(\theta)$, then $lr^s(\sigma\theta) = lr^s(\sigma)$. By using the remark that $lr^{(k-1)*s}(\sigma) = lr^{s*(k-1)}(\Pi_{k+1}(\sigma)) + k$ we get the following list of disequations

$$\begin{aligned} & lr^{(k_{n-1},k_0,\ldots,k_{n-2})}(\sigma_0) > lr^{(k_0,\ldots,k_{n-1})}(\Pi_{k_{n-1}}(\sigma_0)) = lr^{(k_0,\ldots,k_{n-1})}(\Pi_{k_{n-1}}(\sigma_0)\theta'_0) = \\ & lr^{(k_0,\ldots,k_{n-1})}(\sigma_1\theta_1) \geq lr^{(k_0,\ldots,k_{n-1})}(\sigma_1) > lr^{(k_1,\ldots,k_{n-1},k_0)}(\Pi_{k_0}(\sigma_1)) = lr^{(k_1,\ldots,k_{n-1},k_0)}(\Pi_{k_0}(\sigma_1)\theta'_1) = \\ & lr^{(k_1,\ldots,k_{n-1},k_0)}(\sigma_2\theta_2) \geq lr^{(k_1,\ldots,k_{n-1},k_0)}(\sigma_2) > \ldots \\ & \cdots \geq lr^{(k_{n-2},k_{n-1},\ldots,k_{n-3})}(\sigma_{n-1}) > lr^{(k_{n-1},k_0,\ldots,k_{n-2})}(\Pi_{k_{n-2}}(\sigma_{n-1})) = lr^{(k_{n-1},k_0,\ldots,k_{n-2})}(\Pi_{k_{n-2}}(\sigma_{n-1})\theta'_{n-1}) = \\ & lr^{(k_{n-1},k_0,\ldots,k_{n-2})}(\sigma_0\theta_0) \geq lr^{(k_{n-1},k_0,\ldots,k_{n-2})}(\sigma_0) \end{aligned} \tag{6.3.13}$$

which is absurd.

□

Definition 6.3.1. *A* generalized head constraint *is a triple (ϕ, s, A), where ϕ is a scheme variable, s is a finite non empty sequence of positive non zero integers and A is a finite set of sequence variables.*

A usual constraint (ϕ, k, A) can be considered as a special case of a generalized constraint, where $s = (k)$.

We can now generalize the definition of (6.2.1) for generalized constraints: remark that, due to the fact that lemma (6.3.2) proves a disjunction of conditions, a term does not force a constraint, but rather a finite set of constraints.

Definition 6.3.2 (forcing generalized constraints)**.** *A set K of generalized constraints is said a* constraint set *if its elements are all of the form $(\phi, (p_{\gamma(0)}, \ldots, p_{\gamma(n-1)}), A)$ for a fixed sequence $(p_0, \ldots, p_{n-1})$, a fixed set A of sequence variables and γ a cyclic permutation of n elements.*

Let M be a λ-term and K a constraint set. M forces *K if one of the following holds:*

i. $K = \{(\phi_0, s_0, A\}), \ldots, (\phi_{n-1}, s_{n-1}, A)\}$, where $A = \{a_0^1, \ldots, a_{n-1}^{h_{n-1}}\}$ and $sc(M)$ contains the following non recursive equations:

$$\begin{aligned}
\Phi_0^{F_{a_0^1} \ldots F_{a_0^{h_0}}} &= \Psi_0^1 \to \cdots \to \Psi_0^{p_1 - 1} \to \Phi_1^{F'_{a_1^1}} \to \Psi_0^{p_1+1} \\
&\vdots \\
\Phi_j^{F_{a_j^1} \ldots F_{a_j^{h_j}}} &= \Psi_j^1 \to \cdots \to \Psi_j^{p_j - 1} \to \Phi_{j+1}^{F'_{a_{j+1}^1}} \to \Psi_j^{p_j+1} \\
&\vdots \\
\Phi_{n-1}^{F_{a_{n-1}^1} \ldots F_{a_{n-1}^{h_{n-1}}}} &= \Psi_{n-1}^1 \to \cdots \to \Psi_{n-1}^{p_n - 1} \to \Phi_0^{F'_{a_1^1}} \to \Psi_{n-1}^{p_n+1}
\end{aligned} \tag{6.3.14}$$

where, for $0 \leq i \leq n-1$, either Φ_i is ϕ_i and $s_i = (p_i, p_{\gamma_i(0)}, \ldots, p_{\gamma_i(n-1)})$ (where $\gamma_i(x) = x + i \mod n$), either Φ_i is of the form

$$\forall b_1(\forall c_1 \Theta_1 \to \cdots \to \forall c_{p_i}(\forall d_{p_i} \phi_i \to \forall d_{p_i+1} \Theta_{p_i+1}) \ldots) \tag{6.3.15}$$

and $s_i = (p_{\gamma_i(0)}, \ldots, p_{\gamma_i(n-1)}, p_i)$.

ii. the constraints in K are of the form (ϕ_j, s_j, A) for $0 \leq j \leq n-1$, where $s_0 = s' * k$, and there exist a scheme variable ψ and simple schemes $\Phi, \Psi, \Phi_1, \ldots, \Phi_{k-1}, \Phi_{k+1}$ such that $sc(M)$ contains the equation

$$\Psi^{F_{a_1} \ldots F_{a_k}} = \Phi_1^{F_{b_1}} \to \cdots \to \Phi_{k-1}^{F_{b_{k-1}}} \to \Phi^{F_{b_k}} \to \Phi_{k+1}^{F_{b_{k+1}}} \tag{6.3.16}$$

where ϕ_1 occurs in Φ at address k, ψ occurs in Ψ at address k and and one of the two holds:

- *for all constraint $(\phi_j, s_j, A) \in K$, $A = \{b_k\} \cup C' \cup D'$ and M forces the constraint set made of $(\psi, k * s', C \cup D)$ and $(\phi_j, s_j, C \cup D)$, for $1 \leq j \leq n-1$, where for any sequence variable $c \in C$, $c \rhd b_k$ and for any sequence variable $d \in D$, $b_k \rhd d$ and the sets C', D' are defined as follows: $C' \subseteq C$ contains the $c \in C$ such that $c \in s_\phi^\rhd$; D' contains, for any $d \in D$, the sequence variable d', if it exists, which occurs in Ψ at the same position as d in Φ;*

iii. the constraints in K *are* $(\psi, s' * k, A)$ *(where* $s_0 = k * s'$*) and the* (ϕ_j, s_j, A)*, for* $1 \leq j \leq n-1$ *and there exist simple schemes* $\Phi_1, \dots, \Phi_{k-1}, \Phi_{k+1}$ *such that* $sc(M)$ *contains the equation*

$$\Psi^{F_{a_1} \dots F_{a_k}} = \Phi_1^{F_{b_1}} \to \dots \to \Phi_{k-1}^{F_{b_{k-1}}} \to \Phi^{F_w} \to \Phi_{k+1}^{F_{b_{k+1}}} \tag{6.3.17}$$

and one of the two holds:

- *for all constraint* $(\phi_j, s_j, A) \in K$*,* $A = \{b_k\} \cup C' \cup D'$ *and* M *forces the constraint set made of the* $(\phi_j, s_j, C \cup D)$*, for* $0 \leq j \leq n-1$*, where for any sequence variable* $c \in C$*,* $c \rhd b_k$ *and for any sequence variable* $d \in D$*,* $b_k \rhd d$ *and the sets* C', D' *are defined as follows:* $C' \subseteq C$ *contains the* $c \in C$ *such that* $c \in s_{\psi}^{\rhd}$*;* D' *contains, for any* $d \in D$*, the sequence variable* d'*, if it exists, which occurs in* Φ *at the same position as* d *in* Ψ*;*

In order to generalize the notion of compatibility, we must take into account the infinite periodic paths which are coded by addresses: the *path* $\pi(s)$ of an address $s = (p_0, \dots, p_{n-1})$ is the infinite periodic sequence $p_0, \dots, p_{n-1}, p_0, \dots, p_{n-1}, \dots$. Two constraints (ϕ, s, A) and (ψ, s', A') are *incompatible* if $\phi = \psi$, $\pi(s) = \pi(s')$ and $A \cap A' = \emptyset$.

Hence, for two incompatible constraints $(\phi, s, A), (\phi, s', A')$, two possibilities arise: either one between s and s', say s, is of the form $s' * s' * \dots * s'$ (where $*$ here indicates concatenation of sequences), either s and s' are of the form $(\underbrace{k, \dots, k}_{n \text{ times}}), (\underbrace{k, \dots, k}_{m \text{ times}})$, for some $n, m \in \mathbb{N}$.

Let us call a generalized constraint *strict* if it is of the form (ϕ, s, A), where s has length ≥ 2.

We will now prove that a term M never forces incompatible strict generalized constraints. First we show two combinatorial lemmas, that will be used in the proof of proposition (6.3.3):

Lemma 6.3.3. *Let* m *be a* $n \times p$ *matrix, where* $n, p \geq 2$ *and suppose* b *is a coloring of the slots of* m *(i.e. a map* $b : n \times p \to S$*, where* S *is a set of* $m \geq n$ *colors) such that:*

1. *slots in the same column have different colors, i.e.* $b(i, j) \neq b(i, j')$*, for all* $1 \leq i \leq n$ *and* $1 \leq j, j' \leq p$*;*

2. *each color occurs at most twice, i.e., for all* $x \in S$*,* $\sharp(b^{-1}(x)) \leq 2$*.*

Then, there exists an injective function $f : n \to S$ *such that, for all* $1 \leq i \leq n$ *there exists* $1 \leq j \leq p$ *such that* $f(i) = b(i, j)$*.*

Proof. We prove the result by induction on n. If $n = 2$ the result is obvious. If $n = n' + 3$, then, by induction hypothesis there exists an injective function $f : n' + 2 \to S$ such that, for all $1 \leq i \leq n + 2$ there exists a $1 \leq j \leq p$ such that $f(i) = b(i, j)$. Let, for $1 \leq i \leq n$, R_i be the set $R_i := \{b(i, j) | 1 \leq j \leq p\}$. Remark that, by the hypothesis 1., one has $\sharp R_i \geq 2$, for $1 \leq i \leq n$. If R_n is not contained in $Im(f)$, then we can choose a color c_0 in $R_n - Im(f)$ and define an injective function $f' : n \to S$ as $f(i)$ if $i < n$ and c_0 otherwise.

Suppose then $R_n \subseteq Im(f)$ and choose a color $c_0 \in R_n$; as c_0 occurs at most twice, there exists exactly a $k_1 < n$ such that $f(k_1) = c_0$. If $R_{k_1} \subsetneq Im(f) - \{f(k_1)\}$, then we can pick a $c_1 \in R_{k_1} - Im(f) - \{c_0\}$ and define an injective function $f' : n \to S$ as $f(i)$ if $i < n$ and $i \neq k_1$, as c_1 if $i = k_1$ and c_0 if $i = n$. Otherwise, we pick $c_1 \in R_{k_1} - \{c_0\}$ and there exists exactly a $k_2 < n$ such that $k_2 \neq k_1$ and $f(k_2) = c_1$.

If the procedure does not produce an injective function $f' : n \to S$ after $1 < q < n - 1$ iterations, we find a color $c_{q-1} \in R_{k_{q-1}}$ and a $k_q < n$ such that $f(k_q) = c_{q-1}$. If $R_{k_q} \subsetneq Im(f) - \{f(k_1), \dots, f(k_{q-1})\}$, then we can pick a $c_1 \in R_{k_q} - Im(f) - \{c_0, \dots, c_{q-1}\}$ (which is non empty as all the occurrences of the $c_0, \ldots, c_{q-1}$ are in the sets $R_n, R_{k_1}, \ldots, R_{k_{q-1}}$ and because $\sharp R_{k_1} \geq 2$) and define an injective function $f' : n \to S$ as follows

$$f'(i) = \begin{cases} c_0 & \text{if } i = n \\ c_r & \text{if } i = k_r \qquad (1 \leq r \leq q) \\ f(i) & \text{otherwise} \end{cases} \tag{6.3.18}$$

In the worst case, i.e., at the $n-1$-th iteration, we find a color $c_{n-2} \in R_{k_{n-2}}$ and a $k_{n-1} < n$ such that $f(k_{n-1}) = c_{n-2}$. Now one must have $R_{k_n} \subsetneq Im(f) - \{f(k_1), \ldots, f(k_{n-1})\} = Im(f) - Im(f) = \emptyset$, hence we can pick a $c_{n-1} \in R_{k_n} - \{c_0, \ldots, c_{n-2}\}$ and define $f' : n \to S$ by $f'(i) = k_i$, if $i < n$ and $f'(n) = c_0$.

□

Lemma 6.3.4. *Let S be a finite set and $\sim$ be a symmetric non reflexive relation over S. Then there exists a partition $P_1, \ldots, P_n$ of S such that*

1. *for all $1 \leq i \leq n$ and for all $x, y \in P_i$, $x \sim y$;*

2. *for all $1 \leq i \neq j \leq n$ and for all $x \in P_i, y \in P_j$, $x \nsim y$.*

Proof. Let $cl(S) \subseteq \wp(S)$ be the set of *cliques* of S, i.e. the set of all subsets $R \subseteq S$ such that, for all $x, y \in R$, $x \sim y$. Set inclusion defines an order relation over the finite set $cl(S)$. We define the partition $P_1, \ldots, P_n$ recursively as follows:

1. let $S_0 := S$ and P_0 be a maximal element of $cl(S_0)$;

2. let $S_{k+1} := S_k - P_k$ and P_{k+1} be a maximal element of $cl(S_{k+1})$.

Property 1. is immediately verified by the P_i as they are cliques. For property 2. we argue as follows: for all $1 < k \leq n$, let $1 \leq i < k$, $k \leq j \leq n$ and $x \in P_i, y \in P_j$; since P_i is maximal in $cl(S_i)$ and $P_i \cap P_j = \emptyset$, it follows that $x \nsim y$.

□

Proposition 6.3.3. *Let $(K_i)_{1 \leq i \leq k}$ enumerate the constraint sets forced by a λ-term M and suppose that M forces no incompatible (non generalized) constraint. Then there exists a choice function f such that, for all $1 \leq i \leq k$, $f(i) \in K_i$ and the image of f is a set of compatible constraints.*

Proof. Let κ and κ' be two incompatible constraints; if one of the two is not strict, then the incompatibility will be called *simple*; otherwise, it will be called *non simple.*

First we show that there is no simple incompatibility: suppose κ and κ' are incompatible, where $\kappa = (\Phi, (k), A)$ and $\kappa' = (\Phi, (k, \ldots, k), B)$, with $A \cap B = \emptyset$); if κ' is 1-forced, let $\Phi_1, \ldots, \Phi_n$ (where $n \geq 1$) be the schemes occurring in the left in the equations giving rise to the constraint, where $\Phi = \Phi_1$. We claim that for all $2 \leq p \leq n$ no constraint of the form $(\Phi_p, (k), C)$, with $A \cap C = \emptyset$ is forced by M: if for some $1 \leq p \leq n$, M forces the constraint $(\Phi_p, (k), C)$, then, by transporting the constraint along the non recursive equations we obtain that M forces $(\Phi_1, (k), C \cup B')$, where $B' \subseteq B$, contradicting the (non generalized) compatibility of the system.

The case where κ' is $n-$forced, for $n \geq 1$, is treated in a similar way, by considering the fact that the constraint $(\Phi_p, (k), C)$ can be transported (in the sense of definition (6.2.1)) through the equations through which κ' is transported, in the sense of definition (6.3.2) (since the address is constant).

It remains to show the existence of a choice function over constraint sets. Two generalized constraints $(\Phi, s, A), (\Psi, s', A')$ are independent when $A \cap A' = \emptyset$. Two constraint sets are independent where their associated sets of sequence variables are disjoint.

Since independence between constraint sets is a symmetric non reflexive relation, we can apply lemma (6.3.4) and find a partition $P_1, \dots, P_n$ of the constraint sets such that, for all $1 \leq i \neq j \leq n$, the constraint sets in P_i are pairwise independent and two arbitrary constraint sets, respectively in P_i and P_j, are not independent.

It suffices then to show how to define a choice function over a set of pairwise independent constraint sets; indeed a choice function f over all constraint sets can be obtained by gluing together choice functions $f_1, \dots, f_n$ defined over the classes of the partition $P_1, \dots, P_n$: if $1 \leq i \neq j \leq n$, $f_i(p_1) = (\phi, s, A)$ and $f_j(p_2) = (\phi', s', A')$, one must have $A \cap A' \neq \emptyset$, so the image of f is a set of compatible constraints.

Let then P be a set of pairwise independent constraint sets. We can assume w.l.o.g. $n := \sharp P \geq 2$ (the case $\sharp P = 1$ is trivial); let $p \geq 2$ be the minimum dimension of the systems of equations associated with the constraint sets in P. Let S be the set of the scheme variables which occur in the lefthand side of the equations in all the systems associated with the constraint sets in P and $b : n \times p \to S$ a function which associates, with $1 \leq i \leq n$ and $1 \leq j \leq p$, the scheme variable occurring at the j-th equation of the i-th system (we assume given a linear ordering of the systems and, for each system, a linear ordering of its equations).

The function $b(i, j)$ satisfies the hypotheses of lemma (6.3.3): property 1. is immediate, whereas property 2. follows from remark (6.1.3). Hence, there exists an injective choice function $f : n \to S$.

□

From now on, we will say that a system of equations forces a set of (compatible) generalized constraints, rather than a set of constraint sets. That is, we will tacitly suppose that a choice function from constraints sets to (compatible) generalized constraints is given.

2. (I) Simplifying $sc(M)$ by first-order unification We define a variant of the first-order unification algorithm (section (6.1.1)), in order to decompose non recursive equations in $sc(M)$.

Let us define the notion of *semi-congruence* between substitution schemes inductively as follows:

1. any two atomic substitution schemes are semi-congruent;
2. if ϕ is a scheme variable and Φ a substitution scheme in which ϕ occurs, then, for every $a_1, \dots, a_k$, $\phi^{F_{a_1} \cdots F_{a_k}}$ and Φ are semi-congruent;
3. $\Phi \to \Psi$ and $\Phi' \to \Psi'$ are semi-congruent if Φ and Φ' are semi-congruent and Ψ and Ψ' are semi-congruent;
4. $\forall a \Phi$ and $\forall b \Psi$ are semi-congruent if Φ and Ψ are semi-congruent.

The notion of *congruence* is obtained by eliminating the clause 2. of the definition of semi-congruence.

An equation $\Phi^{F_a} = \Psi_1^{F_{b_1}} \to \cdots \to \Psi_k^{F_{b_k}}$, where Φ^{F_a} and $\Psi_1^{F_{b_1}} \to \cdots \to \Psi_k^{F_{b_k}}$ are semi-congruent substitution schemes, induces an obvious map f from the subschemes of Φ to the subschemes of Ψ and a map g from the sequence variables occurring (free or bound) in Φ to the sequence variables occurring (free or bound) in Ψ. If Φ is of the form

$$\Phi = \forall b_1 \Phi_1 \to \forall a_2 (\forall b_2 \Phi_2 \to \cdots \to \forall a_k (\forall b_k \Phi_k \to \forall a_{k+1} \phi_{k+1}) \dots) \tag{6.3.19}$$

and Ψ of the form

$$\Psi = \forall d_1 \Psi_1 \to \forall c_2(\forall d_2 \Psi_2 \to \cdots \to \forall c_k(\forall d_k \Psi_k \to \forall c_{k+1}\psi_{k+1})\dots) \tag{6.3.20}$$

then, for $2 \leq i \leq k+1$, $g(a_i) = c_i$ and, for $1 \leq i \leq k$, $g(b_i) = d_i$. The map g is then recursively extended to the sequence variables occurring in the Φ_i, for $1 \leq i \leq k$ and in ϕ_i.

We define then a variant of the first-order unification algorithm sketched in section (6.1.1), by which we will obtain a system $UNIF(sc(M))$ containing equations between semi-congruent substitution schemes.

The variant of unification is obtained by taking, as inference rules, the transformations below over a set e of equations over substitution schemes:

decomposition if e contains an equation of the form $(\Phi \to \Psi)^{F_a} = \Phi'^{F_{b_1}} \to \Psi'^{F_{b_2}}$, then we replace this equation by the two equations $\Phi^{F_a} = \Phi'^{F_{b_1}}$ and $\Psi^{F_a} = \Psi'^{F_{b_2}}$;

variable elimination if e contains an equation of the form $\phi^{F_{a_1}\dots F_{a_k}} = \Phi$, where $\Phi = \Psi_1^{F_{b_1}} \to \dots \to \Psi_k^{F_{b_k}}$, then two cases arise: if ϕ does not occur in Φ, then eliminate the equation and replace, in the remaining equations in $sc(M)$, every occurrence of the scheme variable ϕ with the substitution scheme Φ (with amount at replacing atomic substitution schemes $\phi^{F_{b_1}\dots F_{b_k}}$ by $\Phi^{F_{b_1}\dots F_{b_k}}$).

If ϕ occurs in Φ, leave the system unchanged.

The main difference between the algorithm above and the usual first-order unification algorithm is that the former does not take the "occur-check" as a case of failure. It simply leaves recursive equations of the form $\phi^{F_{a_1}\dots F_{a_k}} = \Phi_1^{F_{b_1}} \to \dots \to \Phi_p^{F_{b_p}} \to \phi^{F'_{a_1}} \to \Phi_{p+1}^{F_{b_{p+1}}}$ unchanged. In a sense, this algorithm performs all the first-order operations that can be done.

Let us call a system e *irreducible* if no one of the rules above can be applied to e. If an equation $\Phi = \Psi$ belongs to an irreducible system, then Φ and Ψ must be semi-congruent.

It is clear that the transformations above preserve solutions, in the sense that, once e is transformed in e' by means of one of the two rules, then a solution to e' is still a solution to e. Moreover, the termination of all transformation sequences is a direct consequence of the termination of first-order unification. As in that case, distinct irreducible systems can be obtained as the result of distinct transformation sequences: for instance, for a system containing the two equations

$$\phi^{F_a} = \psi^{F_b} \to \chi \tag{6.3.21}$$
$$\psi^{F_c} = \phi^{F_d} \to \chi' \tag{6.3.22}$$

the algorithm produces two distinct solutions, depending on whether it applies variable elimination to the first or to the second equation.

Remark 6.3.1. *The system $UNIF(sc(M))$ induces a new derivation d_M^{UNIF} and a new tree $T(M)^{UNIF}$. The tree $T(M)$ is a subtree of $T(M)^{UNIF}$: the latter is indeed obtained by replacing some leaves of $T(M)$ by trees of the form $T(\Phi)$.*

Finally, the result of the previous paragraph (proposition (6.3.3)) assures that the transformations above preserve compatibility: from a set of equations of the form

$$
\begin{aligned}
\sigma_1\theta_0 &= \Pi_{k_{n-1}}(\sigma_0)\theta'_0 \\
\sigma_2\theta_1 &= \Pi_{k_0}(\sigma_1)\theta'_1 \\
&\vdots \\
\sigma_{n-1}\theta_{n-2} &= \Pi_{k_{n-3}}(\sigma_{n-2})\theta'_{n-2} \\
\sigma_0\theta_{n-1} &= \Pi_{k_{n-2}}(\sigma_{n-1})\theta'_{n-1}
\end{aligned}
\tag{6.3.23}
$$

as in the case of lemma (6.3.2), several applications of variable elimination and decomposition allow indeed to derive, non deterministically, recursive equations of the form

$$
\sigma_i\theta_i = \Pi_{k_{\gamma_i(0)}}(\Pi_{k_{\gamma_i(1)}}(\dots(\Pi_{k_{\gamma_i(n-1)}}(\sigma_i)\theta'_i)\dots)\theta'_{(\gamma_i(1)-1 \mod n)})\theta'_{(\gamma_i(0)+1 \mod n)} \tag{6.3.24}
$$

where, for $0 \leq i \leq n-1$, γ_i is the cyclic permutation over n elements given by $\gamma_i(x) = x + i \mod n$.

Remark that the choice function f of proposition (6.3.3) univocally determines one among the several irreducible systems produced by the $UNIF$ algorithm. Indeed, in any case in which the algorithm can choose (i.e. when a constraint set like (6.3.23) occurs) the choice function in a sense "chooses for him".

2. (II) Decomposing schemes along compatible constraints The second transformation we describe is based on the remark that, if a scheme variable ϕ occurs in k compatible constraints, then the distinct addresses in the constraint must correspond to distinct subtypes of ϕ^S; hence we can replace, in the scheme system, the variable ϕ by a more complex scheme Φ where the distinct addresses correspond to distinct subschemes of Φ, without altering solutions.

Given an address s and a scheme variable ϕ, we define the linear scheme ϕ_s, in which the address s corresponds to a subscheme of ϕ_s:

$$
\begin{aligned}
\phi_{(k)} &:= \forall a_1(\forall b_1\phi_1 \to \forall a_2(\forall b_2\phi_2 \to \dots \to \forall a_{k-1}(\forall b_{k-1}\phi_{k-1} \to \forall a_k(\phi_k)))\dots) \\
\phi_{k*s'} &:= \forall a_1(\forall b_1\phi_1 \to \forall a_2(\forall b_2\phi_2 \to \dots \to \forall a_{k-1}(\forall b_{k-1}\phi_{k-1} \to \forall a_k(\phi_{s'} \to \forall a_{k+1}\phi_{k+1}))\dots)
\end{aligned}
\tag{6.3.25}
$$

where, at any stage, the $a, a_1, \dots, a_{k+1}, b_1, b_{k+1}$ and $\phi_1, \dots, \phi_{k+1}$ denote, respectively, fresh sequence variables and fresh scheme variables.

Given n distinct addresses $s_1, \dots, s_n$, the scheme $\phi_{s_1,\dots,s_n}$ can be defined as a most general unifier of $\phi_{s_1}, \dots, \phi_{s_n}$ (which is always defined and linear).

Given a scheme variable ϕ, let $add(\phi)$ be the set of all the addresses s_1, s_2 which occur in two constraints $(\phi, s_1, A), (\phi, s_2, B) \in \kappa_\phi$, with $A \cap B = \emptyset$.

Let then, for a system E, $SPLIT(E)$ be the system obtained by replacing each occurrence of a scheme variable ϕ by the scheme $\phi_{s_1,\dots,s_n}$, if $add(\phi) = \{s_1, \dots, s_n\}$ is non empty. Remark that, if two distinct addresses s_1, s_2 occur in constraints $(\phi, s_1, A), (\phi, s_2, B) \in \kappa_\phi$, with $A \cap B \neq \emptyset$, we do not split ϕ on those addresses.

This splitting operation corresponds to the the transformation defined on splitting pairs (subsection (6.2.1)) for first-order unification. In particular, all properties of that transformation can be transported to the splitting operation just defined.

The constraints forced by $SPLIT(E)$ can be easily defined: if $(\phi, k * s, A)$ (resp. $(\phi, (k), A)$) is forced by E, then $(\phi_k, s * k, A)$ (resp. $(\phi_k, (k), A)$) is forced by $SPLIT(E)$; if $\psi \neq \phi$, then (ψ, s, A) is forced by E if and only if (ψ, s, A) is forced by $SPLIT(E)$ (we use here the remark that the recursive equations in $SPLIT(E)$ are in bijection with those in $UNIF(M)$, which is a consequence of the remarks on splitting in subsection (6.2.1)).

Hence both the $UNIF$ and the $SPLIT$ transformation preserve compatibility. Moreover, as a consequence of the termination of the alternate iteration of unification and splitting for first-order unification, we get that, after a finite number of alternate iteration of $UNIF$ and $SPLIT$, we end up with a (non unique) system $\mathbf{sc}(M)$ with the following properties:

- an equation in $\mathbf{sc}(M)$ is either a recursive one, either an equation between atomic substitution schemes;

- for any scheme variable ϕ and for any two constraints $(\phi, s, A), (\phi, s', B)$ forced by $\mathbf{sc}(M)$, $A \cap B \neq \emptyset$.

4. Typing a compatible term in System U^- We investigate some of the aspects involved in the typing of a compatible λ-term in System U^-; in particular we highlight the necessity of an impredicative universe in order to solve recursive equations in a uniform and general way. In the construction sketched below we will make an essential use of the impredicative universe $\mathcal{U} := \forall \mathcal{X} \mathcal{X}$.

We will assume given a compatible λ-term M along with a fully reduced system $\mathbf{sc}(M)$. Moreover we will assume that a choice function c is given, which assigns, with every set A of sequence variables occurring in a constraint (ϕ, s, A) forced by M, a sequence variable $a \in A$ in such a way that, if M forces two constraints $(\phi, s, A), (\phi, s, B)$, then $c(A) = c(B) \in A \cap B$. Given such a function c, we can replace every constraint (ϕ, s, A) by a *singlet constraint*, i.e. a constraint of the form $(\phi, s, \{c(A)\})$. Remark that the existence of this choice function was not shown in the previous paragraphs.

Let H be the number of scheme variables occurring in $\mathbf{sc}(M)$. With each scheme variable ϕ and each sequence variable a we associate a constructor variable α_a^ϕ of universe $\mathcal{U}$. Moreover, for every sequence variable a, we denote by $\overline{\alpha}_a$ the sequence of the α_a^ϕ, for an arbitrarily chosen ordering of the set of scheme variables.

For all atomic scheme ϕ, the proposition ϕ^S is defined as follows: let $s_\phi^\triangleright = (a_1, \dots, a_n)$ and κ_ϕ be the set of all constraints (forced by $\mathbf{sc}(M)$) of the form $(\phi, k, \{a\})$.

- if κ_ϕ is empty, then ϕ^S is the proposition below
$$\forall \overline{\alpha}_{a_n} \big(\alpha_{a_n}^\phi\big) \tag{6.3.26}$$
which is well-typed, since $\alpha_{a_n}^\phi$ belongs to the universe *prop*.

- if κ_ϕ contains the (unique) constraint $(\phi, k, \{b\})$ (remark that one must have $b = a_i$, for a certain $1 \leq i \leq n$), then
$$\forall \overline{\alpha}_{a_n} \big((\alpha_{a_i}^\phi)\overline{\alpha}_{a_{i+1}} \dots \overline{\alpha}_{a_n}\big) \tag{6.3.27}$$

which is well-typed since $\alpha_{a_i}^\phi$ belongs to the universe $\underbrace{\mathcal{U} \to \dots \to \mathcal{U}}_{(n-i)\times H} \to prop$.

The propositions Φ^S, for every simple substitution scheme Φ, are defined inductively by
$$(\forall a \Phi \to \Psi)^S := \forall \overline{\alpha}_a (\Phi^S \to \Psi^S) \tag{6.3.28}$$

1. for any sequence variable a, we put $a^S := \{\alpha_a^\phi | \phi \text{ scheme variable}\}$;

2. suppose $sc(M)$ contains an equation of the form

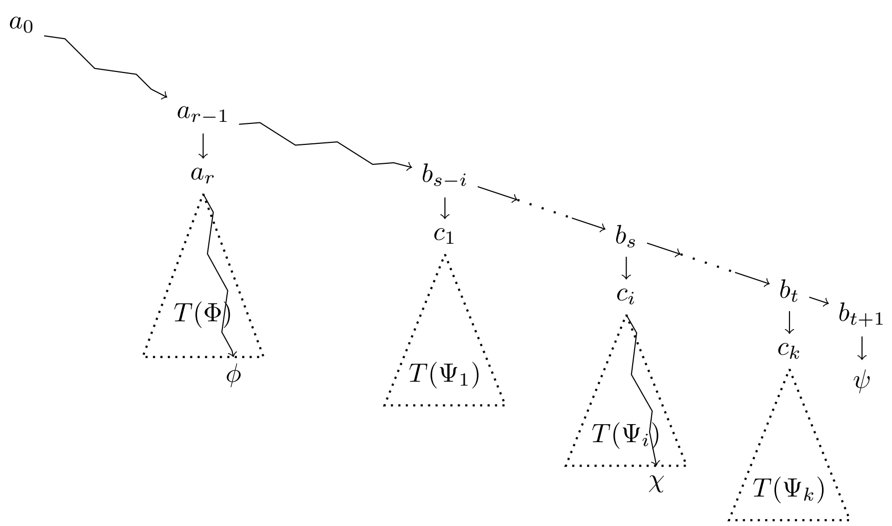
$$\Phi^{F_{c_1}\ldots F_{c_k}} = \Psi_1^{F_{d_1}} \to \cdots \to \Psi_k^{F_{d_k}} \to \psi \tag{6.3.29}$$

The substitutions $\theta^S_{F_{c_{1+l}}}$, for $1 \leq l \leq k-1$ will be identity substitutions. We need to define the substitutions $\theta^S_{F_{c_1}}, \theta^S_{F_{d_1}}, \ldots, \theta^S_{F_{d_k}}$. If the equation (6.3.29) is not recursive, then the two equated schemes are semi-congruent and we can decompose the equation into a finite set of *atomic equations* of the form

$$\phi^{F_{c_1}\ldots F_{c_k}} = \chi^{F_{d_i}} \tag{6.3.30}$$

for a certain $1 \leq i \leq k$ and for ϕ a scheme variable occurring in Φ and χ a scheme variable either occurring in one of the Ψ_i, either equal to ψ. Let $s_\phi^\triangleright = (a_0, \ldots, a_n)$ and let c_1 be a_r, for a certain $1 \leq r \leq n$. Let $s_\chi^\triangleright = (b_0, \ldots, b_m)$ and let the node d_i be b_s, for a certain $1 \leq s \leq m$.

Since equation (6.3.29) comes from a sequence of applications of the form $(x)P_1 \ldots P_k$, by the construction of $T(M)$ we must have either that $m \geq n$ and $a_0 = b_0, \ldots, a_{r-1} = b_{r-1}$ (i.e. the paths from a_0 to ϕ and from a_0 to χ must split exactly at a_{r-1}, as in the figure below)



(6.3.31)

either $n \geq m$ and $a_0 = b_0, \ldots, a_{s-1} = b_{s-1}$ (i.e. the paths a_0 to ϕ and from a_0 to χ must split exactly at b_{s-1}).

We will consider below only the first hypothesis, as the second one can be treated similarly.

The length of the sequence $a_{r-1}, \ldots, a_n$ is equal to the length of the sequence $b_s, \ldots b_m$ (this comes from the fact that the two schemes in (6.3.29) are semi-congruent).

We consider some relevant cases:

(a) there are no constraints on ϕ and χ. Then $\phi^S = \chi^S$ by definition and moreover they are closed types, so we can define $\theta^S_{F_{c_1}}$ and $\theta^S_{F_{d_i}}$ arbitrarily;

(b) there is a constraint (ϕ, k, a_p), for a certain sequence variable a_p, with $1 \leq p \leq r-1$. Then we look for substitutions $\theta^S_{F_{c_1}}, \theta^S_{F_{d_i}}$ solving the equation

$$(\alpha^\phi_{a_p})\overline{\alpha}_{a_{p+1}} \ldots \overline{\alpha}_{a_{r-1}}(\overline{\alpha}_{a_r}\theta^S_{F_{c_1}})^\phi \overline{\alpha}_{a_{r+1}} \ldots \overline{\alpha}_{a_n} = (\alpha^\chi_{b_s}\theta^S_{F_{d_i}})\overline{\alpha}_{b_{s+1}} \ldots \overline{\alpha}_{b_m} \tag{6.3.32}$$

where $(\overline{\alpha}_{a_r}\theta^S_{F_{a_1}})^\phi$ stands for the sequence of the α^ψ_a, for all scheme variable ψ, where α^ϕ_a is replaced by $\alpha^\phi_a\theta^S_{F_{c_1}}$. Then we can put

$$\alpha^\phi_{a_r}\theta^S_{F_{c_1}} = \alpha^\phi_{a_r} \tag{6.3.33}$$

$$\alpha^\chi_{b_s}\theta^S_{F_{d_i}} = \lambda\overline{\gamma}_1.\dots.\lambda\overline{\gamma}_{m-s}.(\alpha^\phi_{a_p})\overline{\alpha}_{a_{p+1}}\dots\overline{\alpha}_{a_{r-1}}\overline{\alpha}_{a_r}\overline{\gamma}_1\dots\overline{\gamma}_{m-s} \tag{6.3.34}$$

where $\overline{\gamma}_i$ stands for sequence of H distinct variables $\gamma^1_i,\dots,\gamma^H_i$ and $\lambda\overline{\gamma}_i.C$ is an abbreviation for $\lambda\gamma^1_i.\dots\lambda\gamma^H_i.C$. Remark that the constructors above are all well-typed.

(c) there is a constraint (ϕ,k,a_p), for a certain sequence variable a_p, with $r-1<p\leq n$. Then $p=r+l$, for a certain integer $0\leq l<m-s$. We have to find substitutions $\theta^S_{F_{c_1}},\theta^S_{F_{d_i}}$ solving

$$(\alpha^\phi_{a_p})\overline{\alpha}_{a_{p+1}}\dots\overline{\alpha}_{a_n} = (\alpha^\chi_{b_s}\theta^S_{F_{d_i}})\overline{\alpha}_{b_{s+1}}\dots\overline{\alpha}_{b_m} \tag{6.3.35}$$

and we put

$$\alpha^\chi_{b_s}\theta^S_{F_{d_i}} = \lambda\overline{\gamma}_1.\dots.\lambda\overline{\gamma}_{m-(s+l+1)}.(\gamma^\phi_{s+l})\overline{\gamma}_1\dots\overline{\gamma}_{m-(s+l+1)} \tag{6.3.36}$$

Again, the constructors above are well-typed.

(d) there is a constraint (ϕ,k,a_r). Then, since equation (6.3.29) is not recursive, there must be a constraint $(\chi,k,b_{r'})$ and we must consider three cases:

i. if $0\leq r'\leq r-1$, then $b_{r'}=a_{r'}$ and we must solve

$$(\alpha^\phi_{a_r}\theta^S_{F_{c_1}})\overline{\alpha}_{a_{r+1}}\dots\overline{\alpha}_{a_n} = (\alpha_{a_{r'}})\overline{\alpha}_{a_{r'+1}}\dots\overline{\alpha}_{a_r}\overline{\alpha}_{b_{r+1}}\dots\overline{\alpha}_{b_{s-1}}(\overline{\alpha}_{b_s}\theta^S_{F_{d_i}})^\chi\overline{\alpha}_{b_{s+1}}\dots\overline{\alpha}_{b_m} \tag{6.3.37}$$

where $(\overline{\alpha}_{b_s}\theta^S_{F_{d_i}})^\chi$ stands for the sequence of the $\alpha^\psi_{b_s}$, for all scheme variable ψ, where α^χ_a is replaced by $\alpha^\chi_a\theta^S_{F_{d_i}}$. Hence we put

$$\alpha^\phi_{a_r}\theta^S_{F_{c_1}} = \lambda\overline{\gamma}_1.\dots.\lambda\overline{\gamma}_{n-r}.(\alpha^\phi_{a_{r'}})\overline{\alpha}_{a_{r'+1}}\dots\overline{\alpha}_{a_r}\overline{\alpha}_{b_{r+1}}\dots\overline{\alpha}_{b_{s-1}}\alpha_{b_s}\overline{\gamma}_1\dots\overline{\gamma}_{n-r} \tag{6.3.38}$$

$$\alpha_{b^\chi_s}\theta^S_{F_{d_i}} = \alpha_{b_s} \tag{6.3.39}$$

ii. if $r-1\leq r'\leq s$, then $r'=r+l$ and we must solve

$$(\alpha^\phi_{a_r}\theta^S_{F_{c_1}})\overline{\alpha}_{a_{r+1}}\dots\overline{\alpha}_{a_n} = (\alpha^\chi_{b_{r'}})\overline{\alpha}_{b_{r'+1}}\dots\overline{\alpha}_{b_{s-1}}(\overline{\alpha}_{b_s}\theta^S_{F_{d_i}})^\chi\overline{\alpha}_{b_{s+1}}\dots\overline{\alpha}_{b_{n-q}} \tag{6.3.40}$$

and we put

$$\alpha^\phi_{a_r}\theta^S_{F_{c_1}} = \lambda\overline{\gamma}_1.\dots.\lambda\overline{\gamma}_{n-r}.(\alpha^\chi_{a_{r'}})\overline{\alpha}_{b_{r'+1}}\dots\overline{\alpha}_{b_{s-1}}\alpha_{b_s}\overline{\gamma}_1\dots\overline{\gamma}_{n-r} \tag{6.3.41}$$

$$\alpha^\chi_{b_s}\theta^S_{F_{d_i}} = \alpha_{b_s} \tag{6.3.42}$$

iii. Finally, if $s<r'\leq m$, then $r'=r+l$ for a certain $0<l<n-s$ and we must solve

$$(\alpha_{a_r}\theta^S_{F_{c_1}})\overline{\alpha}_{a_{r+1}}\dots\overline{\alpha}_{a_n} = (\alpha^\chi_{b_{r+l}})\overline{\alpha}_{b_{s+l+1}}\dots\overline{\alpha}_{b_m} \tag{6.3.43}$$

and we put

$$\alpha^\phi_{a_r}\theta^S_{F_{c_1}} = \lambda\overline{\gamma}_1.\dots.\lambda\overline{\gamma}_{n-r}.(\gamma^\chi_{l-1})\overline{\gamma}_l\dots\overline{\gamma}_{n-r} \tag{6.3.44}$$

The constructors defined above are all well-typed. Moreover, they could all have been typed in F^ω: given the full sequence $a_0, \ldots, a_n$ of sequence variables occurring in a sequence variable ϕ, one could type the constructor variables α_{a_i}, for $1 \leq i \leq n$, with a universe κ_i defined as follows:

$$\kappa_n := prop \tag{6.3.45}$$

$$\kappa_{n-i} := \kappa_{n-i+1} \to \cdots \to \kappa_n \to prop \tag{6.3.46}$$

By the way, the appeal to the impredicative universe $\mathcal{U}$ is fundamental when dealing with recursive equations, as shown below.

If equation (6.3.29) is cyclic, then $\Phi^{F_{c_1}}$ is atomic and one has a constraint (ϕ, k, a_r). We must find $\theta, \theta_1, \ldots, \theta_k$ such that

$$(\alpha_{a_r}\theta)\alpha_{a_{r+1}} \ldots \alpha_{a_n} = \Psi_1^S\theta_1 \to \cdots \to \Psi_k^S\theta_k \to \psi^S \tag{6.3.47}$$

For all $1 \leq i \leq k$ a constructor $D_i := \lambda\overline{\gamma}_1. \ldots .\lambda\overline{\gamma}_{n-r}.C_i$, of universe $\underbrace{\mathcal{U} \to \cdots \to \mathcal{U}}_{(n-r)\times H} \to prop$, such that $(D_i)\overline{\alpha}_{a_{r+1}} \ldots \overline{\alpha}_{a_n} = \Psi_i^S$ can be defined. To do that we make use of the map g associated with equation (6.3.29); remark that g sends the tree $T(\Phi)$ into an isomorphic subtree of the tree $T(\Xi)$, where Ξ is $\forall b_{s-1}(\forall e_1\Psi_1 \to \forall d_2(\forall e_2\Psi_2 \to \cdots \to \forall e_k\Psi_k \to \forall d_{k+1}\psi)$.

Let χ be a leaf of Ξ; the linear order of the free sequence variables of χ is of the form

$$a_0, \ldots, a_{r-1}, b_r \ldots b_{s-1} f(a_r) \ldots f(a_n) b_{s+n} \ldots b_m \tag{6.3.48}$$

Let χ^C be then like χ^S, but with the variables $\overline{\alpha}_{f(a_i)}$, for $r \leq i \leq n$, replaced by the variables $\overline{\gamma}_{i-r}$. We define then

$$(\Phi \to \Psi)^C := \Phi^C \to \Psi^C \tag{6.3.49}$$

$$(\forall\overline{\alpha}_{b_{s+l}}\Phi)^C := \forall\overline{\alpha}_{b_{s+l}}\Phi^C \tag{6.3.50}$$

for $1 \leq l \leq n-r$. Finally we can put $C_i := \Psi_i^C$.

We can thus take as $\theta_{b_i}^S$, for all $1 \leq i \leq k$, the identical substitution and put

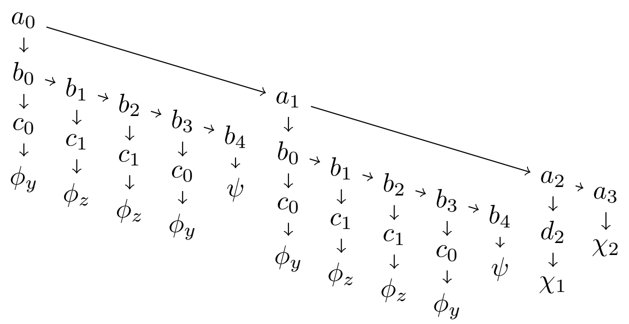
$$\alpha_{a_r}^\phi \theta_{F_{c_1}}^S = \lambda\overline{\gamma}_1. \ldots .\lambda\overline{\gamma}_{n-r}.\big(C_1 \to \cdots \to C_{k+1}\big) \tag{6.3.51}$$

The typing of the propositions C_i cannot be done in System F^ω: the variables α_{r+l}^ψ must belong to the two distinct universes $\underbrace{\mathcal{U} \to \cdots \to \mathcal{U}}_{(n-(r+l))\times H} \to prop$ and $\underbrace{\mathcal{U} \to \cdots \to \mathcal{U}}_{(m-(r+l))\times H} \to prop$. This is possible only if they are assigned a universally quantified type like $\mathcal{U}$ or, for instance, $\forall\mathcal{X}(\underbrace{\mathcal{U} \to \cdots \to \mathcal{U}}_{(n-(r+l))\times H} \to \mathcal{X})$.

Two remarks on the solution of recursive equations can be done at this point. First, that the substitution θ in equation (6.3.47) must have access to all the bound variables $\overline{\alpha}_{r+1}, \ldots, \overline{\alpha}_n$, that might occur in the righthand side type and that it could note introduce otherwise (since a substitution cannot introduce bound variables). Hence, a completely uniform solution to such equations cannot be devised in System F, where one cannot "stock" bound variables in the body of an atomic type (since atomic types cannot contain applications).

Second, that simple universes are not enough to manage the application of a variable α to a linear order of bound variables, as these linear orders (which correspond to the descending paths from the sequence variable a such that $\alpha \in a^S$ to the leafs of $T(M)$) can vary in length. The impredicative universe $\mathcal{U}$ provides then a very simple solution to this problem.

Example 6.3.1. *The λ-term $M = (\lambda x.(x)x)\lambda y.\lambda z.(y)zy$ was shown in [Mal90] to be untypable in System F. The tree $T(M)$ is the following (where $\forall c_0\phi_y$ and $\forall c_1\phi_z$ denote, respectively, the schemes of the variables y and z):*



(6.3.52)

The system $sc(M)$ is made of the following two equations:

$$\phi_y^{F_{c_0} F_{c'_0}} = \phi_z^{F_{c_1}} \to \phi_y^{F'_{c_0}} \to \psi \tag{6.3.53}$$

$$\forall c_0 \phi_y^{F_{b_0}} \to \forall c_1 \phi_z^{F_{b_1}} \to \psi = \forall b_0 \big(\forall c_0 \phi_y^{F'_{b_0}} \to \forall c_1 \phi_z^{F'_{b_0}} \to \psi\big) \to \forall d_2\chi_2 \to \forall a_3\chi_3 \tag{6.3.54}$$

Which force the two compatible constraints $(\phi_y, (2), \{c_0\})$ and $(\phi_y, (1), \{b_0\})$; hence we must split $\forall c_0\phi_y$ into the scheme $\Phi_{(1),(2)}$ below

$$\Phi_{(1),(2)} = \forall c_0\big(\forall e_0\phi_0 \to \forall c'_1(\forall e_1\phi_1 \to \forall c'_2\phi_2)\big) \tag{6.3.55}$$

with the new constraints $(\phi_1, 2, \{c_0\})$ and $(\phi_0, 1, \{b_0\})$. Equations (6.3.53) *become now*

$$\Phi_{(1),(2)}^{F_{c_0} F_{c'_0}} = \phi_z^{F_{c_1}} \to \Phi_{(1),(2)}^{F'_{c_0}} \to \psi \tag{6.3.56}$$

$$\forall c_0 \Phi_{(1),(2)}^{F_{b_0}} \to \forall c_1 \phi_z^{F_{b_1}} \to \psi = \forall b_0 \big(\forall c_0 \Phi_{(1),(2)}^{F'_{b_0}} \to \forall c_1 \phi_z^{F'_{b_0}} \to \psi\big) \to \forall d_2\chi_2 \to \forall a_3\chi_3 \tag{6.3.57}$$

which induce, after decomposition, the following equations

$$\forall e_0 \phi_0^{F_{c_0}} = \phi_z^{F_{c_1}} \tag{6.3.58}$$

$$\forall e_1 \phi_1^{F_{b_0}} = \forall c_1 \phi_z^{F'_{b_0}} \tag{6.3.59}$$

The equations above force, by transport, the compatible constraints $(\phi_z, 1, \{b_0, c_1\})$ and $(\phi_z, 2, \{b_0, c_0\})$. Since $\{b_0, c_1\} \cap \{b_0, c_0\} \neq \emptyset$ we do not split the scheme ϕ_z and we can define our ground substitution by picking $H = 2$:

$$\phi_y^S = \forall\overline{\alpha}_{c_0}(\alpha_{b_0}^{\phi_y})\overline{\alpha}_{c_0} \to \forall\overline{\alpha}_{c'_1}\big(\forall\overline{\alpha}_{e_1}(\alpha_{c_0}^{\phi_y})\overline{\alpha}_{c'_1}\overline{\alpha}_{e_1} \to \forall\overline{\alpha}_{c'_2}\alpha_{c'_2}^{\phi_y}\big) \tag{6.3.60}$$

$$\phi_z^S = \forall\overline{\alpha}_{c_1}(\alpha_{b_0}^{\phi_y})\overline{\alpha}_{c_1} \tag{6.3.61}$$

The complete definition of S can now be obtained from the definition of ϕ_y^S and ϕ_z^S.

6.3.3 Some consequences of the conjecture

We present three interesting applications of conjecture (6.3.1) which constitute its main motivations.

A combinatorial characterization of typability The interest of the notion of compatibility in [Mal90] is that it provides a purely combinatorial way to treat some cases of non typability in System F. The general notion of compatibility presented in this chapter was developed in order to generalize this aspect. In particular, all the results and the arguments discussed so far are of a purely combinatorial nature. In particular, the property of being compatible can be easily shown to be decidable.

The validity of conjecture (6.3.1) would then yield an entirely combinatorial characterization of the typability problem. Moreover, since from theorem (6.2.1) it follows that a strongly normalizing λ-term must be compatible, we would get that every strongly normalizable λ-term is typable in System U^-.

Remark that, as the property of strong normalization is undecidable, we cannot expect that the notion of compatibility characterizes normalization. Indeed, an example of a not normalizing term typable in System U^- (and hence, by proposition (6.3.2), compatible) is given in [Coq94]; similarly, Girard's paradox (appendix (C)) provides an example of a not normalizing term which is typable in System U.

Moreover, we remarked in the last section that the notion of compatibility does not characterize solvable terms either, as the λ-term $\lambda z.(z)(\delta)\delta$ is incompatible just like the term $(\delta)\delta$ though being in head normal form, contrarily to the latter.

Typing recursive functions in System U^- We show an important application of conjecture (6.3.1): we show that, if compatible terms are typable, then for every total unary recursive function f, there exists a λ-term which computes f and which has type $\mathbf{N} \to \mathbf{N}$ in System U^-.

To this end we rely on a representation of partial recursive functions in λ-calculus which comes essentially from [BGP94], where it is shown that every partial recursive function can be simulated by a λ-term of the form $\lambda x.M$, where M is in head normal form (indeed M is normal). A slightly simplified version of this construction is presented in appendix (C).

In order to investigate the typability of recursive functions in System U^-, we consider then the typability of a λ-term M in head normal form with a single free variable x, with the constraints that both x and M must receive type $\mathbf{N}$ (in other words, we consider a subcase of the type checking problem).

We first show a simple lemma:

Lemma 6.3.5. *Let M be a λ-term in head normal form which does not start by an abstraction; if M is typable in U^- then, for any type σ, there exists a context Γ such that $\Gamma \vdash M : \sigma$ is derivable in U^-.*

Proof. If M is a variable z, then it suffices to put $\Gamma = (z : \sigma)$.

If M is an application $(z)M_1 \dots M_k$, then the scheme of M is a scheme variable ϕ that occurs in only one equation of $sc(M)$ of the form

$$\phi_z^{F_{a_1} \dots F_{a_k}} = \Phi_1 \to \dots \to \Phi_k \to \phi \tag{6.3.62}$$

hence, given an arbitrary typing of M in U^-, we can choose $\phi^S = \sigma$.

□

We investigate now how the assignment $\mathcal{S}(x) = \forall a((\phi \to \phi) \to (\phi \to \phi))$ reflects on the compatibility of the induced system.

Let $\forall a_1 \phi$ be the scheme of the variable x; we have to consider the assignment $\mathcal{S}(x) = \forall a_1((\phi \to \phi) \to (\phi \to \phi))$; furthermore, we must add to the set of forced constraints a constraint $(\phi, 1, \{a_1\})$. Let us first remark that, either a constraint (ϕ, k, A), with $a_1 \in A$, is already forced by M, either M forces no constraint on ϕ: since x is a free variable, the scheme ϕ can only occur in equations of the form

$$\phi^{F_{a_1} \dots F_{a_k}} = \Psi_1^{G_{b_1}} \to \dots \to \Psi_k^{G_{b_k}} \to \psi \tag{6.3.63}$$

or of the form

$$\Psi^{F_{a_1} \dots F_{a_k}} = \Phi_1^{G_{b_1}} \to \dots \to \phi^{F_{a_1}} \to \dots \to \Psi_k^{G_{b_k}} \to \psi \tag{6.3.64}$$

Hence, all constraints on ϕ must be of the form (ϕ, k, A), with $a_1 \in A$.

Due to the *non linearity* of the scheme $\forall a_1((\phi \to \phi) \to (\phi \to \phi))$, by clauses *ii.*, *iii.* of definition (6.2.2), the adjunction of the constraint $(\phi, 1, \{a_1\})$ might induce new constraints: in case M contains a subterm of the form $(x)PQ$, then $sc(M)$ must contain an equation of the form

$$\phi^{F_{a_1} F_{a_2}} = \Psi_1^{G_{b_1}} \to \Psi_2^{G_{b_2}} \to \psi \tag{6.3.65}$$

If ϕ is replaced by $\mathcal{S}(x)$ one gets

$$((\phi \to \phi) \to (\phi \to \phi))^{F_{a_1} F_{a_2}} = \Psi_1^{G_{b_1}} \to \Psi_2^{G_{b_2}} \to \psi \tag{6.3.66}$$

By decomposition this implies

$$(\phi \to \phi)^{F_{a_1}} = \Psi_1^{G_{b_1}} \to \Psi_1^{G_{b_1}} \tag{6.3.67}$$

$$(\phi \to \phi)^{F_{a_1} F_{a_2}} = (\Psi_1^{G_{b_1}} \to \Psi_1^{G_{b_1}})^{F_{a_2}} = \Psi_2^{G_{b_2}} \to \psi \tag{6.3.68}$$

Hence, a constraint (ψ_2, k, A) (where ψ_2 occurs in Ψ_2 at path k) will be transported into a constraint $\kappa = (\psi_1, k, A' \cup \{b_1, a_2\})$, (where ψ_1 occurs in Ψ_1 at path k), where A' is obtained from A following definition (6.2.1).

We wish to show that, if M already forces a constraint $\kappa' = (\psi_1, k, B)$ incompatible with κ, then, *for a certain* Church integer $\mathbf{k}$, the term $M[\mathbf{k}/x]$ is not normalizing: indeed for any $k \geq 1$, $M[\mathbf{k}/x]$ contains the subterm $(\mathbf{k})PQ$ which reduces to $P^k Q = P^{k-1}(PQ)$. One easily verifies then that the term PQ forces the two incompatible constraints κ and κ' and cannot then, by theorem (6.2.1), be normalizing.

Remark that if, moreover, M is in head normal form and does not begin by an abstraction, then, by proposition (6.3.5), a ground substitution S for M such that $(\forall a_1 \phi)^S = \mathbf{N}$ and $\Phi_M^S = \mathbf{N}$ (where Φ_M is the scheme of M) can be defined following the examples discussed in the previous subsection, with some slight modifications, due to the non linearity of the scheme $\forall a_1((\phi \to \phi) \to (\phi \to \phi))$.

In particular, we must consider a new case (e) for the definition at pag. 175:

(e) there is a constraint (ϕ, k, a_r) which was transported from a constraint $(\psi, k, b_{r'})$ *on another occurrence of* ϕ *in* Φ. Then, by clause (d), the substitution $\theta^S_{F_{c_1}}$ has already been defined, and we must find a substitution $\theta^S_{F_{d_i}}$ satisfying

$$(\alpha^\phi_{a_r} \theta^S_{F_{c_1}}) \overline{\alpha}_{a_{r+1}} \dots \overline{\alpha}_{a_n} = (\alpha^\chi_{b_s} \theta^S_{F_{d_i}}) \overline{\alpha}_{b_1} \dots \overline{\alpha}_{b_m} \tag{6.3.69}$$

and we can put

$$\alpha^\chi_{b_s} \theta^S_{F_{d_i}} = \lambda \overline{\gamma}_1 . \dots . \lambda \overline{\gamma}_m . (\alpha^\phi_{a_r} \theta^S_{F_{c_1}}) \overline{\gamma}_1 \dots \overline{\gamma}_m \tag{6.3.70}$$

In definitive, the construction sketched should convince the reader that, if conjecture (6.3.1) is true, then one should be able to prove the following fact: let M be a λ-term with exactly one free variable x, in head normal form and not starting by an abstraction; suppose further that, for all integer k, $M[\mathbf{k}/x]$ is strongly normalizable. Then, $\lambda x.M$ can be given type $\mathbf{N} \to \mathbf{N}$ in System U^-.

On the basis of the representation of recursive functions in λ-calculus given in (C), this implies then that, for every total unary recursive function f there exists a λ-term $\hat{\mathbf{f}}$ which computes f and which has type $\mathbf{N} \to \mathbf{N}$ in System U^-.

Typability in System N The type inference of System N is directly inherited from the one of System U^-. The syntax-directed type inference system is just the one in (6.3.1), when one drops the requirement of well-typedness for propositions and constructors and in the definition of the relation $\leq$. A ground substitution S is defined exactly like for System U^-, again by dropping well-typedness.

As a consequence, one can prove the analogue of proposition (6.1.2) also for naïve type theory:

Proposition 6.3.4 (principal typing derivations in System N)**.** *Let M be a λ-term, then the following two hold:*

i. if a ground substitution S satisfies $eq^*(M)$ and $ct(M)$, then d_M^S is a typing derivation in U^- of M in F;

ii. if d is a typing derivation in U^- of M in F, then there exists a ground substitution s satisfying $eq^*(M)$ and $ct(M)$ and such that $d = d_M^S$.

Proof. Once more, the two parts are straightforwardly proved by induction on the derivation d_M^S.

□

As the main source of expressivity of System N is provided by fixed-point types (see subsection (2.4.3)), we first recall some well-known results on typability in the presence of fixed point types, then we will turn our attention towards types having a normal form.

A quite general result connecting fixpoint operators with the typability of not normalizing combinators is in [Men87]: Mendler considers an extension S_{rec} of simple type theory with types satisfying recursive equations. This means that, for all type σ containing a free variable α, a type $\mu\alpha.\sigma$ is admitted with the typing rules:

$$\frac{\Gamma \vdash M : \sigma[\mu\alpha.\sigma/\alpha]}{\Gamma \vdash M : \mu\alpha.\sigma}\ (\mu - I) \qquad\qquad \frac{\Gamma \vdash M : \mu\alpha.\sigma}{\Gamma \vdash M : \sigma[\mu\alpha.\sigma/\alpha]}\ (\mu - E) \tag{6.3.71}$$

Remark that the existence in N of a fixpoint operator fix allows the definition of a type $fix(\lambda\alpha.\sigma)$ satisfying the same rules as $\mu\alpha.\sigma$.

Given a set of k equations of the form

$$\sigma_i = \tau_i \tag{e_i}$$

for $1 \leq i \leq k$, such that a cyclic equation $\sigma_i = \tau$, where σ_i occurs *negatively* in σ is derivable from it, Mendler assumes given a ground substitution s (i.e. a map from variables to types in S_{rec}) that satisfies all equations (e_i); then he shows how to build a not normalizing λ-term which is typable by means of these types.

This result shows that, as soon as a negative recursive equation occurs, the existence of a fixpoint solution implies the existence of a "paradoxical term" in the type system. In particular this implies that one will not find fixpoint solutions for negative recursive equations in reducible type systems like F or F^ω.

Since not normalizing types allow the typing of all λ-terms in a trivial way, it is natural to turn to consider typability by means of types having a normal form. Though the type discipline of System N is less restrictive than the one of System U^-, a consequence of conjecture (6.3.1) is that the terms typable in N by means of types having a normal form are exactly those that are already typable in System U^-.

In order to adapt Lemma (6.2.1) to the case System N, we must take into account the fact that a proposition might not have a normal form. However, if σ has a normal form, we can keep the definitions of $lr(\sigma), H(\sigma), H^k(\sigma)$ given for System U^- and prove:

Lemma 6.3.6. *Let σ be a type of System N satisfying an equation of the form*

$$\sigma\theta = \Pi_k(\sigma)\theta' \tag{6.3.72}$$

for certain substitutions θ, θ' and a positive integer k. Then either $H^{k-1}(\sigma) \in dom(\theta')$ either σ has no normal form.

Proof. It suffices to reproduce the argument of lemma (6.3.1). $\square$

Lemma (6.2.2), being a consequence of lemma (6.2.1), can be immediately transported to System N. Hence one has

Proposition 6.3.5. *If M is incompatible, then it is not typable in N by types having a normal form.*

For the converse side, it suffices to remark that a term that is typable in System U^- must be typable by means of types having a normal form in System N; hence, by the conjecture (6.3.1) we get that a term is typable in N by means of types having a normal form if and only if it is compatible.

This simple result shows an apparently counter-intuitive fact: if conjecture (6.3.1) is true, then, as soon as one does not consider fixpoint (i.e. not normalizing) types, the impredicativity of System U^- is exactly as powerful as the one of full naïve type theory.

Part IV

Perspectives

Chapter 7

Towards a proof theory of "uncertain" proofs

In this final chapter we sketch some possible lines of research which arise from the perspectives and results of this thesis. The ideas here presented go in the direction, indicated throughout the text, of a proof-theoretic investigation of "uncertain" proofs: these are proofs whose computational content can be entirely described in a recursive way, but whose logical meaning (validity, reducibility) demands for complex and somehow circular (see section (4.3.1)) arguments, whose reliability can be endlessly questioned (and, in some unfortunate cases - see prelude at page 7 and section (4.3.2) -, disproven).

In the first section we try to highlight the subtle difference between the combinatorial and non combinatorial aspects connected with typing pure λ-terms. In the second section we give a (very sketchy) idea of the proof-theoretical perspectives which arise from the approach developed in chapter (6) on System U^-.

From the viewpoint of "how proof theory", we sketch an argument to derive, from the typability of total recursive functions in U^-, derivations of totality for total recursive functions in an extension of higher order arithmetics **UA**, which reflects the type structure of U^-. The argument is thought of as an extension of the usual technique (described in section (4.3.1)) of expressing the reducibility of single typed λ-terms by means of arithmetical predicates. Remark that, in order to extend the argument to U^-, one must adopt a notion of reducibility which cannot, *globally*, be defined in set-theory (as it leads to the paradoxes discussed in section (4.3.2) and section (5.1.1)).

From the viewpoint of "why proof theory" we sketch some possible directions to answer the question: how much of System U^- can we actually justify?

7.1 The why and the how of typing

The acknowledgement of the recursive content of proofs constituted one of the spines of last century proof theory. In chapter (2) and chapter (3) we reconstructed the correspondence by which proofs can be seen as programs. This correspondence is illustrated by the table below:

PROOF THEORY	TYPE THEORY
proofs	programs
formulae	types
rules	typing rules
Gentzen transformations	execution of programs

From the interactionist perspective (chapter (3)) it is through typing that we attach meaning to programs, by getting to know how we can use them (by applying them to other programs or by applying other programs to them).

Part (III) contains the characterization of some recursive aspects related to typing in polymorphic type systems. In chapter (5) it is shown that terms having a universally quantified type must satisfy certain “genericity equations” which allow to characterize the shape of those terms in a finite, combinatorial, way. In chapter (6) the property of having a type in System U^- is investigated by means of a property concerning the “compatibility” of the vicious circles present in the λ-term.

By exposing some combinatorial features of typing, those investigations indicate then that the line between the “how” content of typing and its “why” content (giving meaning, assuring validity) is indeed quite subtle.

As we noted in chapter (6), the problem of *verifying* whether a program has a given type (*type-checking*), reduces to the *prima facie* more complex problem of *finding* a type for a program (*typability*). Indeed, the problem of verifying whether a given λ-term codes a recursive function, i.e. has type $\mathbf{N} \to \mathbf{N}$, reduces to the problem of *finding* types on which to extract the type variable occurring in the type $\mathbf{N}$.

From an arithmetical viewpoint this corresponds to the fact that a proof of $\forall nA$ can be constructed by appeal to induction axioms over *arbitrarily complex* formulae. From a proof-theoretical viewpoint this corresponds to the fact that a proof of $\forall x(N(x) \Rightarrow A)$ can be constructed by appeal to comprehension rules (i.e. $(\forall E)$ rules) containing witnesses of arbitrary logical complexity. In a word, this is linked to the loss of the subformula property in second order logic.

Hence we encounter a well-known (see [Poh89, Lei90, Lei01]) phenomenon which should interest the why proof theorist: to a growth in the logical complexity of the comprehension rules admitted (or of the formula occurring in induction axioms) there corresponds a growth in the expressive power of the logic obtained.

A very instructive example of this phenomenon is at work in the trick used by Gentzen in [Gen69] to prove transfinite inductions of growing complexity in first-order arithmetics (section (2.2.3)). Remark that, whereas Gentzen has to use more and more complex formulae, so that his proofs in the end exploit all the logical strength of first-order arithmetics, the “trick” by which he can construct more and more complex proofs is of a combinatorial nature. The moral to be drawn is the following: on the one hand, transfinite induction for more and more complex ordinal numbers requires more and more complex arithmetical predicates to be proved; on the other hand, the fact that by appealing to more and more complex inductions one can construct more and more complex proofs can be explained in purely combinatorial terms.

The *Curry-Howard* content of this phenomenon lies in the fact that we can control the computational complexity of the typable (resp. provably total) functions by controlling the logical complexity of the types (resp. formulae) occurring in the extractions over the type $\mathbf{N}$ (resp. in induction axioms) - for a detailed description see [Lei01, Lei90]. Hence, on the one hand, in order to justify the totality of these typed functions one has to rely over more and more logically complex notions of reducibility; on the other hand, the fact that more and more complex functions can be typed by means of more and more complex types can be justified in purely combinatorial terms (an interesting example of this fact within System F^ω can be found in the introduction of the paper [Urz97]).

In the light of the equational characterization of typing described in chapter (6), it might be then of interest to investigate the following question: can the appeal to types (i.e., logically, predicates) of growing complexity be explained by a growth in number and nesting of the *recursive* equations induced by λ-terms?

Indeed, the coding in λ-calculus of recursive functions growing faster and faster results in an augmentation of the number and nesting of auto-applications in the λ-terms obtained. Hence, the systems of equational specification of types arising from such λ-terms will force more and more head constraints (section (6.2.1)); as a consequence it is natural to expect, in the types arising from the solutions to such systems, a growth in the number of addresses and, consequently, in the number of occurrences of quantifiers and implication symbols: in short, a growth in logical complexity.

7.2 A Curry-Howard perspective on System U

In chapter (6) we investigated a recursive characterization of the typability problem for λ-terms within an inconsistent type system which extends System F. The appeal to inconsistent systems was justified by the major uniformity that such extremely expressive systems offer for the investigation of the solvability of systems of equational specifications of types.

7.2.1 System U^- and "how-proof theory"

The proof that, from a term M of type $\mathbf{N} \to \mathbf{N}$ computing a function f, one can recover a proof of the totality of f in second order arithmetics constitutes a wonderful example of the "kaleidoscopic" nature of second order logic (see section (4.3.1)). Indeed, one relies on the fact that the reducibility of M, depending on a finite number of comprehension instances, can be expressed directly within second order logic.

Here we provide a sketch of how an extension of this well-known technique to System U^- could be developed. First, one defines an (inconsistent) extension of second order Heyting arithmetics $\mathbf{HA}^2$, that we might call $\mathbf{UA}$. $\mathbf{UA}$ must have a notion of universe, defined like for System U^-, as well as predicate variables and quantifiers for any universe. Typed predicates are defined by means of typing rules obtained from the rules for typing constructors in U^-. In addition to typing rules for predicates $\mathbf{UA}$ must contain the two logical schemas:

$$\frac{\Gamma \vdash A \quad A \text{ bindable in } \Gamma}{\Gamma \vdash \forall^\kappa X A}\ (\forall^\kappa I) \qquad\qquad \frac{\Gamma, A[C^\kappa/X] \vdash \Delta}{\Gamma, \forall^\kappa X A \vdash \Delta}\ (\forall^\kappa E) \tag{7.2.1}$$

We retain for $\mathbf{UA}$ the notation $P \in Q$ to state that the predicate Q over objects of universe κ, holds of P (of universe κ).

Second, one has to devise a notion of reducibility for System U^-. This can be done along the lines of Martin-Löf's type theory or along those of Reynolds' set theoretical interpretation: in section (2.4) it was remarked that the reducibility interpretation of System U^- corresponds to the set-theoretic interpretation of System F.

Remark that neither Martin-Löf's reducibility nor Reynolds's interpretation can be formalized in set-theory, as they entail paradoxical results. By the way, for our purpose it is of no importance that reducibility be defined in a consistent and set-theoretically acceptable way: all that matters is that, *locally* (i.e. for any specific term of System U^-), reducibility be definable in $\mathbf{UA}$ (which is inconsistent). Here is one of the most striking aspects of how-proof theory: one can use inconsistent theories and inconsistent proofs to obtain (valid) results.

The idea of the construction comes from Martin-Löf's proof in section (4.3.1): an *extended reducibility candidate e.r.c.* can be now thought of as a pair (κ, S) made of a universe κ and a set S over κ (i.e. a constructor of type $\kappa \to prop$) which satisfies the properties below:

$\mathbf{R}^{\kappa}$1) *if C belongs to κ and $C \in S$, then C is strongly normalizing;*

$\mathbf{R}^{\kappa}$2) *if C belongs to κ, $C \in S$ and C reduces to C', then $C' \in S$;*

$\mathbf{R}^{\kappa}$3) *if C belongs to κ and, for all its immediate reducts C', $C' \in S$, then $C \in S$.*

For a fixed universe κ and a variable X of type $\kappa \to prop$, the properties $\mathbf{R}^{\kappa}\mathbf{1}-\mathbf{3}$ can be expressed in **UA** by means of a formula $\mathbf{CR}^{\kappa}[X]$, with parameter X.

Let $\mathcal{U}$ be the impredicative universe $\forall\mathcal{X}\mathcal{X}$; then, if S is a set over κ, for an arbitrary universe κ, since S also belongs to the universe $\mathcal{U} \to prop$, $\mathbf{CR}^{\mathcal{U}}[S]$ is well typed and expresses, intuitively, the fact that, for a certain universe κ, S satisfies the properties $\mathbf{R}^{\kappa}\mathbf{1} - \mathbf{3}$. Hence, in a sense, $\mathbf{CR}^{\mathcal{U}}[S]$ expresses the fact that, for a certain κ, (κ, S) is an *e.r.c.*.

With this impredicative construction, one should then associate, with each closed universe κ, a predicate $\overline{\kappa}$ in **UA**, of universe $\kappa \to prop$ such that, for all constructor C, C belongs to κ if and only if $\overline{C} \in \overline{\kappa}$ is derivable in **UA** (where $\overline{C}$ denotes a coding of the constructors of System U^- in **UA** which is sketched below). The predicates $\overline{\kappa}$, parametrized by a set $Z_1, \dots, Z_n$ of fresh variables of universe $\kappa' \to prop$, for κ' arbitrary, should be defined by inductive clauses resembling the ones below:

$$\overline{prop}[Z_1, \dots, Z_n] := \lambda X.\mathbf{CR}^{prop}[X] \tag{7.2.2}$$

$$\overline{\mathcal{X}_i}[Z_1, \dots, Z_n] := Z_i \tag{7.2.3}$$

$$\overline{\kappa \to \kappa'}[Z_1, \dots, Z_n] := \lambda X^{\kappa \to \kappa'}.\forall^{\kappa} Y \big(Y \in \overline{\kappa}[Z_1, \dots, Z_n] \Rightarrow (XY) \in \overline{\kappa}'[Z_1, \dots, Z_n]\big) \tag{7.2.4}$$

$$\overline{\forall\mathcal{X}.\kappa}[Z_1, \dots, Z_n] := \lambda X^{\forall\mathcal{X}.\kappa}.\forall^{\mathcal{U}\to prop} Y \big(\mathbf{CR}^{\mathcal{U}}[Y] \Rightarrow X \in \overline{\kappa}[Z_1, \dots, Z_n, Y]\big) \tag{7.2.5}$$

Now, to any constructor C of System U^-, of universe κ, we can associate a constructor $\overline{C}$ in **UA** such that $\overline{C} \in \overline{\kappa}$ is (hopefully) derivable in **UA**. The case in which C is a proposition σ corresponds essentially to the one analyzed in section (4.3.1) for System F (since $\overline{C}$ is $\overline{Red_{\sigma}}[Z_1, \dots, Z_n]$ and $\overline{\kappa}[X]$ is $\mathbf{CR}[X]$); a hypothesis to define the remaining cases could be the following:

$$\overline{\lambda\gamma_i.C} := \lambda X_i.\overline{C} \tag{7.2.6}$$

$$\overline{CD} := \overline{C}\ \overline{D} \tag{7.2.7}$$

It would remain then to show that, for any proposition σ and any (closed) λ-term M of (closed) type σ, the argument for the reducibility of M can be entirely coded in **UA** by means of the notions introduced above.

Given a reducibility interpretation for System U^-, one could then reproduce the usual argument to show that, if M codes a *partial* recursive function f and has type $\mathbf{N} \to \mathbf{N}$ in System U^-, then we can recursively extract from it a derivation in **UA** of the totality of f.

If the argument just sketched works, we would get (by relying on conjecture (6.3.1)), through the systems U^- and **UA**, a recursive battery of sequent calculus derivations which is complete (at least) for true Π^0_2 formulae. Hence one would get a new proof theoretic realization of the idea of the library of Babel, with a direct application to the understanding of Gödel's theorems: the fact that one cannot recursively describe a consistent system in which all true Π^0_2 formulae are provable does not imply that one cannot recursively describe the proofs of those true statements in an (inconsistent) Curry-Howard extension of second order arithmetics.

7.2.2 System U^- and "why-proof theory"

From the viewpoint of the why-proof theorist one is interested in drawing a clear line between correct and incorrect proofs. Since U^- is inconsistent, it must contain trivial terms inhabiting any type. Proof-theoretically, this should imply that **UA** contains trivial proofs of every proposition. One would like then, at least, a criterion telling the "untrivial" terms from the trivial ones which arise from paradoxes. Similarly, one would like a criterion telling the "untrivial" derivations in **UA** from the trivial ones.

A first option comes from the Curry-Howard correspondence: from a derivation of the totality of (possibly partial) recursive function obtained from a paradox one cannot extract, by the forgetful functor, a term computing the function. Hence, a derivation of a certain formula should be rejected as trivial unless it comes from a typed λ-term computing the right function (determined by the type associated with the formula). In this sense, the approach of chapter (6) leads towards an extension of the Curry-Howard connection between proofs and programs to a class of incorrect derivations.

By the way, this approach does not allow to characterize valid derivations: from a term M of type $\mathbf{N} \to \mathbf{N}$ we get, by the procedure sketched in the previous paragraph, a derivation d of the totality of a certain recursive function, computed by M. Hence, this derivation would satisfy the Curry-Howard criterion but, since reducibility fails for System U^-, we cannot rule out that M computes, indeed, a *partial* function, that is, that for some integer k, the term $M\mathbf{k}$, of type $\mathbf{N}$, is not reducible. In a word, the derivation d would be Curry-Howard, but its conclusion would be false!

Here we stumble once more against the fact that, whereas proof-theoretic validity is a logically complex notion, the Curry-Howard criterion exclusively concerns the recursive content of the derivations.

In addition to the Curry-Howard criterion, one might then ask that the term extracted from the proof be reducible. For instance, the term extracted from a totality proof should be in the reducibility $Red_{\mathbf{N}\to\mathbf{N}}$. This would be enough to exclude, in the Π^0_2 case, totality proofs for *partial* functions, as in the case just examined.

By the way, if we wish to extend this criterion to all **UA** derivations, we stumble upon the fact that reducibility for System U^- is an inconsistent notion. Thus, we would be trying to commit the task of evaluating possibly inconsistent derivations to a yet more problematic, since inconsistent, judge.

A third option comes from the remark that, in order to avoid Curry-Howard proofs of totality for partial recursive functions, appeal to the whole reducibility theory is not necessary: all that is needed is the result that typed λ-terms are (strongly) normalizing. This requirement can be expressed by a Π^0_2 arithmetical statement. For instance one could require that, if M has type $\mathbf{N} \to \mathbf{N}$ in System U^- and, moreover, M codes a *total* recursive function, then its typing must be done within a *reducible* subsystem of U^-.

Hence, our criterion would become: a derivation is correct if its extracted λ-term is typable in a reducible subsystem of System U^-. In a word, the why-proof theory of System U^- could well correspond to the question: how much of it can we justify?

The search for a hierarchy of more and more complex reducible subsystems of U^- (or of reducible extensions of System F) is compatible with the principles of why-proof theory: to the reducibility of more and more complex systems there should correspond the use of logical principles (e.g. comprehension axioms) of growing logical complexity.

However, such investigations would be of limited interest from an epistemological viewpoint: they would not provide a reduction of complex problems to simpler ones, as the totality of a recursive function would be vindicated by appeal to logical principles of complexity well beyond

Π^0_2 [1]. One could here make a comparison with Gentzen's consistency proof for arithmetics and quote the remark that Girard reports from Kreisel in [Gir00]

> Gentzen a établi la cohérence de l'induction jusqu'á ω au moyen de l'induction jusqu'á ϵ_0. [Gir00]

[1] One could cite Feferman's work on transfinite progressions of arithmetical theories [Fef62] as an example of a similar enterprise.

Part V

Appendices

Appendix A

Properties of System N

We list without proof some basic properties of system N (all provable by induction on the term M) which will be implicitly adopted in the remaining proofs of this section:

Proposition A.0.1 (basic properties)**.** *i. If $\Gamma \vdash M : \sigma$ is derivable, then $\Gamma' \vdash M : \sigma$, with $\Gamma \subseteq \Gamma'$, is too;*

ii. If $\Gamma \vdash M : \sigma$ is derivable, then, if $x \in FV(M)$, $(x : \tau) \in \Gamma$, for some type σ;

iii. If $\Gamma \vdash x : \sigma$ is derivable, then $(x : \sigma') \in \Gamma$ for some σ' such that $\sigma' \preceq \sigma$;

iv. Properties 1 and 2 of the previous section hold for N;

v. If $\Gamma \vdash M : \sigma$ is derivable and M' is a subterm of M, then $\Gamma \vdash M' : \tau$ is derivable for some τ.

Remark that system N explicitly takes account of reduction over the types only in the case of extractions; anyway, for the case of types containing a redex the following holds:

Proposition A.0.2 (Redex elimination)**.** *If $\Gamma \vdash M : \sigma$ is derivable in N and $\sigma \rightsquigarrow \sigma'$, then there exists a type σ'' such that $\sigma' \rightsquigarrow \sigma''$ and $\Gamma \vdash M : \sigma''$ is derivable in N.*

Before proceeding to the proof of the lemma, we prove the following:

Lemma A.0.1. *If $\Gamma, (x : \tau) \vdash M : \sigma$ is derivable in N and $\tau \rightsquigarrow \tau'$ then there exists σ' such that $\sigma \rightsquigarrow \sigma'$ and $\Gamma, (x : \tau') \vdash M : \sigma'$ is derivable in N.*

Proof. By induction on M:

(var) Let $\Gamma, (x : \tau) \vdash x : \sigma$ be derivable; then $\tau \preceq \sigma$, i.e. $(\tau)\rho \rightsquigarrow \sigma$ for a certain ρ. By applying confluence we obtain then a σ' such that $\sigma \rightsquigarrow \sigma'$ and $(\tau')\rho \rightsquigarrow \sigma'$:

$$\begin{array}{ccc} (\tau)\rho & \rightarrow & \sigma \\ \downarrow & & \downarrow \\ (\tau')\rho & \rightarrow & \sigma' \end{array} \tag{A.0.1}$$

We finally have $\tau' \preceq \sigma'$ and thus $\Gamma, (x : \tau') \vdash M : \sigma'$.

(λ) We have $\Gamma, (x : \tau) \vdash \lambda y.M : \forall\beta(\tau \to \sigma)$. It is enough to show that if $\Gamma, (x : \tau), (y : \rho) \vdash M : \sigma$ is derivable, then $\Gamma, (x : \tau'), (y : \rho) \vdash M : \sigma$ is derivable, which is true by induction hypothesis.

(@) We have $\Gamma, (x : \tau) \vdash MN : \sigma$, which implies $\Gamma, (x : \tau) \vdash M : \rho \to \sigma'$ (with $\sigma' \preceq \sigma$) and $\Gamma, (x : \tau) \vdash N : \rho$; again, we apply the induction hypothesis and obtain $\Gamma, (x : \tau') \vdash M : \rho \to \sigma*$, with $\sigma' \rightsquigarrow \sigma^*$; now, since $\sigma' \preceq \sigma$ means that $(\sigma')\mu \rightsquigarrow \sigma$ for a certain μ. By the same argument of the case (var) we find then a σ'' such that $\sigma \rightsquigarrow \sigma''$ and $\sigma^* \preceq \sigma''$.

□

Proof of proposition (A.0.2). Again, by induction on M:

(var) We have $\Gamma, (x : \tau) \vdash x : \sigma$; frome $\tau \preceq \sigma$ and $\sigma \rightsquigarrow \sigma'$, one has $\tau \preceq \sigma'$, and thus $\Gamma, (x : \tau) \vdash x : \sigma'$.

(λ) We have the following derivation:

$$\frac{\begin{array}{c}\vdots\, d \\ \Gamma, (x : \tau) \vdash M : \sigma \quad \tau \preceq \sigma\end{array}}{\Gamma \vdash \lambda x.M : \forall\beta(\tau \to \sigma)} \tag{A.0.2}$$

We have $\forall\beta(\tau \to \sigma) \rightsquigarrow \forall\beta(\tau' \to \sigma')$, with $\tau \rightsquigarrow \tau'$ and $\sigma \rightsquigarrow \sigma'$. From lemma (A.0.1) it follows that we can replace d with the derivation d' below

$$\frac{\begin{array}{c}\vdots\, d' \\ \Gamma, (x : \tau') \vdash M : \sigma^* \quad \tau' \preceq \sigma^*\end{array}}{\Gamma \vdash \lambda x.M : \forall\beta(\tau' \to \sigma^*)} \tag{A.0.3}$$

With $\sigma \rightsquigarrow \sigma^*$. Since $\sigma \preceq \sigma^*$ and $\sigma \rightsquigarrow \sigma'$, by confluence there exists σ'' such that $\sigma \rightsquigarrow \sigma''$ and $\sigma^* \rightsquigarrow \sigma''$; since $\tau' \preceq \sigma^*$ and $\sigma^* \rightsquigarrow \sigma''$ implies $\tau' \preceq \sigma''$ we can finally derive $\Gamma \vdash \lambda x.M : \forall\beta(\tau' \to \sigma'')$.

(@) We have $\Gamma \vdash MN : \sigma$ and thus $\Gamma \vdash M : \tau \to \sigma$ and $\Gamma \vdash N : \tau$; by induction hypothesis there exists τ' and σ'' such that $\sigma' \rightsquigarrow \sigma''$, $\tau \rightsquigarrow \tau'$ and $\Gamma \vdash M : \tau' \to \sigma''$ is derivable; again, by induction hypothesis there exists τ'' such that $\tau \rightsquigarrow \tau''$ and $\Gamma \vdash N : \tau''$ is derivable. By confluence we finally find τ''' and σ''' such that $\tau', \tau'' \rightsquigarrow \tau'''$, $\sigma'' \rightsquigarrow \sigma'''$ and $\Gamma \vdash MN : \sigma'''$ is derivable.

□

Lemma A.0.2. *If $\Gamma \vdash M : \sigma$ is derivable in N, then $\Gamma[\tau/\alpha] \vdash M : \sigma[\tau/\alpha]$ is derivable in N.*

Proof. Induction on M:

(var) From $\Gamma, (x : \sigma_1) \vdash x : \sigma$ it follows that $\sigma_1 = \forall\overline{\beta}.\sigma_1'$ and $\sigma_1'[\overline{\rho}/\overline{\beta}] \rightsquigarrow \sigma$ for a certain $n \in \mathbb{N}$ and types $\rho_1, \ldots, \rho_n$; remark that, for $\overline{\beta} = \beta_1, \ldots, \beta_k$, for $1 \leq i \leq k$, $\beta_i \notin FV(\tau)$. As a consequence, $\sigma_1'[\tau/\alpha][\overline{\rho}[\tau/\alpha]/\overline{\beta}] \equiv \sigma_1'[\overline{\rho}/\overline{\beta}][\tau/\alpha]$; from general lambda calculus consideration we know that $\sigma_1'[\overline{\rho}/\overline{\beta}] \rightsquigarrow \sigma$ implies $\sigma_1'[\overline{\rho}/\overline{\beta}][\tau/\alpha] \rightsquigarrow \sigma[\tau/\alpha]$; we finally deduce $\sigma_1'[\tau/\alpha] \preceq \sigma[\tau/\alpha]$, hence $\Gamma[\tau/\alpha], (x : \sigma_1[\tau/\alpha]) \vdash x : \sigma[\tau/\alpha]$.

(λ) From $\Gamma \vdash \lambda x.M : \sigma$ we deduce $\sigma = \forall\overline{\beta}(\sigma_1 \to \sigma_2)$ and $\Gamma, (x : \sigma_1) \vdash M : \sigma_2$ and $\sigma_1 \preceq \sigma_2$; by induction hypothesis we deduce $\Gamma[\tau/\alpha], (x : \sigma_1[\tau/\alpha]) \vdash M : \sigma_2[\tau/\alpha]$; remark, again, that $\overline{\beta} \notin FV(\tau)$ (abuse of notation), hence we derive $\Gamma[\tau/\alpha] \vdash \lambda x.M : \sigma[\tau/\alpha]$.

(@) From $\Gamma \vdash MN : \sigma$ we deduce $\Gamma \vdash M : \sigma_1 \to \sigma_2$, $\Gamma \vdash N : \sigma_1$ and $\sigma_2 \preceq \sigma$; by induction hypothesis we have $\Gamma[\tau/\alpha] \vdash M : \sigma_1[\tau/\alpha] \to \sigma_2[\tau/\alpha]$ and $\Gamma[\tau/\alpha] \vdash N : \sigma_1[\tau/\alpha]$; since $\sigma_2 = \forall\overline{\beta}.\sigma_2'$ and there exist types $\overline{\rho}$ such that $\sigma_2'[\overline{\rho}/\overline{\beta}]$; remark that $\overline{\beta} \notin FV(\tau)$, hence $\sigma_2'[\tau/\alpha][\overline{\rho}[\tau/\alpha]/\overline{\beta}] \equiv \sigma_2'[\overline{\rho}/\overline{\beta}][\tau/\alpha]$; from general lambda calculus consideration we know that $\sigma_2'[\overline{\rho}/\overline{\beta}] \rightsquigarrow \sigma$ implies $\sigma_2'[\overline{\rho}/\overline{\beta}][\tau/\alpha] \rightsquigarrow \sigma[\tau/\alpha]$; we finally deduce $\sigma_2'[\tau/\alpha] \preceq \sigma[\tau/\alpha]$, hence $\Gamma[\tau/\alpha] \vdash MN : \sigma[\tau/\alpha]$.

□

Since in system N one cannot assume types to be already in normal form (since such a normal form may not exists), the *substitution lemma* and the *subject reduction lemma* (see [BAGM92]) must be reformulated as stating that typability is preserved (under substitution or reduction) up to some reduction of the types. Clearly, if types are already in normal form, the reformulation below of these results becomes equivalent to the usual one.

Lemma A.0.3. *If $\Gamma \vdash M : \sigma$ is derivable in N and $\sigma \preceq \sigma'$, then there exists σ'' such that $\sigma' \rightsquigarrow \sigma''$ and $\Gamma \vdash M : \sigma''$.*

Proof. Induction on M:

(var) From $\Gamma, (x : \tau) \vdash x : \sigma$ one has $\tau \preceq \sigma$ and $\sigma \preceq \sigma'$, hence $\tau \preceq \sigma'$ by transitivity and thus $\Gamma, (x : \tau) \vdash x : \sigma'$.

(λ) From $\Gamma \vdash \lambda x.M : \sigma$ it follows $\sigma = \forall\overline{\alpha}(\sigma_1 \to \sigma_2)$ and $\Gamma, (x : \sigma_1) \vdash M : \sigma_2$. From $\sigma \preceq \sigma'$ it follows that $\sigma_1[\overline{\tau}/\overline{\alpha}] \to \sigma_2[\overline{\tau}/\overline{\alpha}] \rightsquigarrow \sigma'$ for a certain $n \in \mathbb{N}$ and types $\overline{\tau} = \tau_1, \ldots, \tau_n$; by applying the lemma (A.0.2) and remarking that $\overline{\alpha} \notin FV(\Gamma)$ we find $\Gamma, (x : \sigma_1[\overline{\tau}/\overline{\alpha}]) \vdash M : \sigma_2[\overline{\tau}/\overline{\alpha}]$, hence $\Gamma \vdash \lambda x.M : \sigma_1[\overline{\tau}/\overline{\alpha}] \to \sigma_2[\overline{\tau}/\overline{\alpha}]$; by proposition (A.0.2) we have the thesis.

(@) From $\Gamma \vdash MN : \sigma$ it follows $\Gamma \vdash M : \sigma_1 \to \sigma_2$, $\Gamma \vdash N : \sigma_1$ and $\sigma_2 \preceq \sigma$; by transitivity, $\sigma_2 \preceq \sigma'$ and we have the thesis.

□

Proposition A.0.3 (substitution lemma in [BAGM92]). *If $\Gamma, (x : \sigma) \vdash M : \tau$ and $\Gamma \vdash N : \sigma$ are derivable, then $\Gamma \vdash M[N/x] : \tau'$ is derivable for some τ' such that $\tau \rightsquigarrow \tau'$.*

Proof. By induction on the generation of $\Gamma, (x : \sigma) \vdash M : \tau$.

(var) We have $\Gamma, (x : \sigma) \vdash x : \tau$, $\sigma \preceq \tau$ and $\Gamma \vdash N : \sigma$; by lemma (A.0.3) we derive $\Gamma \vdash N : \tau'$ with $\tau \rightsquigarrow \tau'$.

(λ) We have $\Gamma, (x : \sigma) \vdash \lambda y.M : \tau$, and thus $\tau = \forall\overline{\alpha}.(\tau_1 \to \tau_2)$ and $\Gamma, (x : \sigma), (y : \tau_1) \vdash M : \tau_2$; by induction hypothesis we find $\Gamma, (y : \tau_1) \vdash M[N/x] : \tau_2'$ for a certain τ_2' such that $\tau_2 \rightsquigarrow \tau_2'$, and thus $\Gamma \vdash \lambda y.M[N/x] : \forall\overline{\alpha}(\tau_1 \to \tau_2')$.

(@) We have $\Gamma, (x : \sigma) \vdash M'M'' : \tau$, hence $\Gamma, (x : \sigma) \vdash M' : \tau_1 \to \tau_2$, $\Gamma, (x : \sigma) \vdash M'' : \tau_1$ and $\tau_2 \preceq \tau$. By induction hypothesis we have $\Gamma \vdash M'[N/x] : \tau_1' \to \tau_2'$, for τ_1', τ_2' such that $\tau_1 \rightsquigarrow \tau_1'$ and $\tau_2 \rightsquigarrow \tau_2'$; similarly we find τ_1'' such that $\tau_1 \rightsquigarrow \tau_1''$ and $\Gamma \vdash M''[N/x] : \tau_1''$; remark that, by confluence, we can replace τ_1' and τ_1'' by a type τ_1^* such that $\tau_1' \rightsquigarrow \tau_1^*$ and $\tau_1'' \rightsquigarrow \tau_1^*$, thus concluding $\Gamma \vdash M'[N/x] : \tau_1^* \to \tau_2'$ and $\Gamma \vdash M''[N/x] : \tau_1^*$; since $(\tau_2)\rho_1 \ldots \rho_n \rightsquigarrow \tau$, for certain $\rho_1, \ldots, \rho_n$, and $\tau_2 \rightsquigarrow \tau_2'$, by confluence there is a type τ' such that $\tau \rightsquigarrow \tau'$ and $\tau_2' \preceq \tau'$. We conclude $\Gamma \vdash M[N/x] : \tau'$.

□

Proof of Proposition 2.4.1 . We first show the theorem for the relation $\rightsquigarrow_1$; the extension is made by induction on the lenght of the reduction.

We consider the prime case, i.e. $M = (\lambda x.P)Q$ and $M' = P[Q/x]$. From $\Gamma \vdash M : \sigma$ it follows $\Gamma \vdash \lambda x.P : \tau \to \sigma'$ for certain τ, σ' such that $\sigma' \preceq \sigma$ and $\Gamma \vdash Q : \tau$; again, we deduce $\Gamma, (x : \tau) \vdash \sigma'$ and, from lemma (A.0.3) we find $\Gamma \vdash P[Q/x] : \sigma''$ for a certain σ'' such that $\sigma' \rightsquigarrow \sigma''$; as usual, by confluence, we find a type σ^* such that $\sigma \rightsquigarrow \sigma^*$ and $\sigma'' \preceq \sigma^*$.

For the general case, we use the fact that if a redex $(\lambda x.P)Q$ occurs in M, then there exists a subderivation of the type derivation d with conclusion $\Gamma' \vdash (\lambda x.P)Q : \tau$ with $\Gamma \subseteq \Gamma'$; by the argument above we find $\Gamma' \vdash P[Q/x] : \tau'$ with $\tau \rightsquigarrow \tau'$. By repeatedly applying proposition (A.0.2), we build a derivation d^* with conclusion $\Gamma \vdash M' : \sigma^*$ for a certain σ^* such that $\sigma \rightsquigarrow \sigma^*$.

□

Appendix B

Girard's paradox

We already encountered Girard's paradox for System U in chapter (4). In this appendix we describe a (simplified) version of the paradox in order to investigate its computational content: in particular, we extract from the paradox specifications for typing a not normalizing λ-term. This analysis is intended as an introductory example of the approach that is developed in full generality in chapter (6).

The interest of this paradox, with respect to Russell's antinomy, is that the typing of a diverging combinator is obtained in a type system (System U) whose types satisfy a strong normalization theorem (indeed the types of U are essentially terms of System F). In particular, such a paradox shows that the fact that a term is typed by means of types in normal form does not assure the normalization of the typed term. This is yet another clue to the fact that, when investigating typability in an abstract way, one is led to make abstraction from the normalization of typed terms (that is why, indeed, inconsistent systems like U or N are of especial interest to the matter).

Burali-Forti's paradox and the "powerful universe" Girard's original argument (see [Gir72]) is obtained from a variant of Burali-Forti's paradox: the latter is a paradox found in naïve set theory in 1897 by Cesare Burali-Forti ([BF97]) based on the ordinal numbers.

An ordinal number was there defined as the order type of a well-ordered set. Any element x of a well-ordered set a induces an *initial segment* $a_x := \{y \in a | y < x\} \in \wp(a)$. One can show then that the "set" of order types of well-ordered sets is well-ordered by the relation $\alpha < \beta$ which holds if there exists a monotone function from α to an initial segment of β. Let then Ω be the order type of this well-ordered "set"; then, for every well-ordered set a, there exists a monotone function from the order type α of a to an initial segment of Ω, so that one has $\alpha < \Omega$. In particular, one has then $\Omega < \Omega$, which contradicts the fact that Ω is a well-order.

Remark that in the argument above one passes from the "set" of order types Ω to the "set" of all initial segments of Ω (contained in $\wp\wp\Omega$) and back. In the argument below, we define a non set-theoretic universe $\mathcal{V}$ (in the sense of Reynolds, see section (5.1.1)) and we define two maps s, t which allow to pass from $\mathcal{V}$ to $\wp\wp\mathcal{V}$ and back. If we think of the map s as the map which associates, with each element $x \in \mathcal{V}$, the set of all sets $X \in \wp\mathcal{V}$ which contain all the *predecessors* of x, then we can define, following [Hur95], a order relation $x < y$ over $\mathcal{V}$ given by

$$x < y \textit{ iff } \forall X \in \wp\mathcal{V}(X \in sx \Rightarrow y \in X) \tag{B.0.1}$$

that is, x is less than y if y belongs to any set which contains all predecessors of x. We can then define a notion of *inductive set*:

$$Ind(X) := \forall x \in CV(X \in sx \Rightarrow x \in X) \tag{B.0.2}$$

that is, X is inductive if all its elements belong to any set which contains all their predecessors. Finally, an element $x \in \mathcal{V}$ can be said *well-founded* when, as usually, it is in the intersection of all inductive sets. Formally,

$$WF(x) := \forall X \in \wp\mathcal{V}(Ind(X) \Rightarrow x \in X) \tag{B.0.3}$$

The universe $\mathcal{V}$ we consider is called "powerful" in [Hur95] and is the following:

$$\mathcal{V} := \forall \mathcal{X}((\wp\wp\mathcal{X} \to \mathcal{X}) \to \wp\wp\mathcal{X}) \tag{B.0.4}$$

In the following we adopt the following conventions: we use small variables $x, y, \ldots$ for elements of $\mathcal{V}$ and capital variables $X, Y, \ldots$ for elements of $\wp\mathcal{V}$. In order to avoid confusion, we will use letters $u, v, w, \ldots$ to denote term variables, i.e. variables occurring in "proof-like" λ-terms. Moreover, $c \not< d$ we will indicate the type $c < d \to \bot$, where $\bot := \forall^{prop}\alpha\alpha$.

$\mathcal{V}$ is paradoxical in the sense of Reynolds' result, since we can build a constructor t in the universe $\wp\wp\mathcal{V} \to \mathcal{V}$:

$$t := \lambda f.\lambda x.\lambda y.(x)\lambda z.(y(f(zf))) \tag{B.0.5}$$

Moreover we can build a constructor s in the universe $\mathcal{V} \to \wp\wp\mathcal{V}$

$$s := \lambda x.(x)\lambda y.ty \tag{B.0.6}$$

such that, for all set X in $\wp\wp\mathcal{V}$, the following holds

$$stX = \lambda y.(X)\lambda z.(y)tsz \tag{B.0.7}$$

or, in set notation

$$stX = \{y : \wp\mathcal{V} | \{z : \mathcal{V} | tsz \in Y\} \in X\} \tag{B.0.8}$$

The notions of *predecessor*, *inductive set* and *well-founded element* are defined follows:

$$x < y := \forall^{\wp\mathcal{V}} X((sx)X \to Xy) \tag{B.0.9}$$
$$Ind(X) := \forall^{\mathcal{V}} x((sx)X \to Xx) \tag{B.0.10}$$
$$WF(x) := \forall^{\wp\mathcal{V}} X(Ind(X) \to Xx) \tag{B.0.11}$$

We have now all the elements to proceed to the paradoxical argument.

The paradox The analysis that follows is organized in this way: we describe the paradoxical argument in three steps; at each step we make the reasoning correspond to the typing of a combinator (by relying on the analysis in [Hur95]); next we show that the types used in the argument must satisfy some equational specifications which constitute necessary (and sufficient) conditions for the typing of the combinator. The application of the three combinators will produce in the end a typable, though not normalizing, λ-term, the following:

$$W := (\lambda w.w(\lambda u.\lambda v.(v)uv)w)\lambda u.(u)u \tag{B.0.12}$$

Remark that W has no head normal form and reduces to itself in a finite number of steps.

Let Ω be $t\{X : \wp\mathcal{V} | Ind(X)\}$, which is an element of $\mathcal{V}$. In the first step, corresponding to the combinator $\lambda u.(u)u$, we show that the set Ω is well-founded; in the second step, corresponding to the combinator $\lambda u.\lambda v.(v)uv$, we show that the set $e = \{y : \mathcal{V} | tsy \not< y\}$ is inductive. Finally, in the third step, corresponding to the combinator $\lambda w.w(\lambda u.\lambda v.(v)uv)w$ we show that Ω is not well-founded (by relying on the inductivity of e). As a consequence, one has that the term W is well-typed in System U. Moreover, in the end we can describe the conditions for the typing of W by a finite set of equational specifications.

Step 1 We show that Ω is well-founded: suppose X is inductive; in order to prove that $\Omega \in X$, it suffices to prove $X \in s\Omega$. From equation (B.0.8) if follows that

$$s\Omega = \{X : \wp\mathcal{V} | Ind(\{y : \mathcal{V} | tsy \in X\})\} \tag{B.0.13}$$

and thus we have to show that the set $\{y \in \mathcal{V} | tsy \in C\}$ is inductive. Let then x be in $\mathcal{V}$; since X is inductive, if $X \in stsx$, then $tsx \in X$.

We show then that from the argument above we can extract a typing of the combinator $\lambda u.(u)u$:

$$\dfrac{\dfrac{(X : \wp\mathcal{V}), (u : Ind(X)) \vdash Ind(X) \quad \begin{matrix}\vdots \\ (X : \wp\mathcal{V}), (u : Ind(X)) \vdash \Omega : \mathcal{V}\end{matrix}}{(X : \wp\mathcal{V}), (u : Ind(X)) \vdash u : X \in s\Omega \to \Omega \in X}(\forall E)_{\mathcal{V}} \quad \dfrac{(X : \wp\mathcal{V}), (u : Ind(X)) \vdash u : Ind(X) \quad \begin{matrix}\vdots \\ (X : \wp\mathcal{V}), (u : Ind(X)) \vdash tsx : \wp\mathcal{V}\end{matrix}}{(X : \wp\mathcal{V}), (u : Ind(X)) \vdash u : X \in s\Omega}(\forall E)_{\mathcal{V}}}{\dfrac{\dfrac{(X : \wp\mathcal{V}), (u : Ind(X)) \vdash (u)u : \Omega \in X}{(X : \wp CV) \vdash \lambda u.(u)u : Ind(X) \to \Omega \in X}}{\vdash \lambda u.(u)u : WF(\Omega)}(\forall I)_{\wp\mathcal{V}}} \tag{B.0.14}$$

The crucial part in the typing above is represented by the two extractions performed over the variable u of type $Ind(X)$. Indeed, in order to type the auto-application $(u)u$ one has to extract the variable u over two different types σ_1, σ_2 satisfying an equation of the form

$$\sigma_1 = \sigma_2 \to \tau \tag{B.0.15}$$

for a certain type τ. In the next section we'll develop this intuition in full generality. For the moment we can remark that the type $Ind(X)$ can be written under the form $\forall^{\mathcal{V}} y \Phi(y, X)$, to stress the fact that the open type Φ depends on the variables y (of universe $\mathcal{V}$) and X (of universe $\wp\mathcal{V}$). Since an extraction corresponds to the application of a *substitution* F over a bound variable, we can rewrite the constraint above under the following form

$$\Phi(F_1(y), X) = \Phi(F_2(y), X) \to \tau \tag{B.0.16}$$

Now, the constraint above is satisfied by the type $\Phi(y, X) := X \in sy \to y \in X$ and the two substitutions $F_1(y) = \Omega$ and $F_2(y) = tsy$.

Step 2 Before we actually prove that Ω *is not* well-founded, so giving rise to the antinomy, we show that the set $e := \{y | tsy \not< y\}$ is inductive. Suppose $E \in sx$, for a certain $x \in \mathcal{V}$; then we show that $tsx \not< x$: indeed, suppose $tsx < x$; this means that, for all $X \in \wp\mathcal{V}$, if $X \in sx$, then $tsx \in X$. In particular, since $e \in sx$, one has $tsx \in e$, hence $tstsx \not< tsx$. On the other hand, we show that $tstsx < tsx$, so that we can conclude that $tsx \not< x$. From the assumption $tsx < x$ it follows that, by letting $d := \{y | tsy \in X\}$, if $d \in sx$, then $tsx \in d$; by equation (B.0.8), this means that, if $X \in stsx$, then $tstsx \in X$, i.e. that $tstsx < tsx$.

From the argument above we can extract the following typing of the combinator $\lambda u.\lambda v.(v)uv$:

$$\dfrac{\dfrac{\dfrac{\dfrac{(x : \mathcal{V}), (u : e \in sx), (v : tsx < x) \vdash v : tsx < x}{(x : \mathcal{V}), (u : e \in sx), (v : tsx < x) \vdash v : e \in sx \to tstsx \not< tsx}(\forall E)_{\wp\mathcal{V}} \quad (x : \mathcal{V}), (u : r \in sx), (v : tsx < x) \vdash u : e \in sx}{(x : \mathcal{V}), (u : e \in sx), (v : tsx < x) \vdash (v)u : tstsx \not< tsx}(@) \quad \dfrac{(x : \mathcal{V}), (u : r \in sx), (v : tsx < x) \vdash v : tsx < x}{(x : \mathcal{V}), (u : e \in sx), (v : tsx < x) \vdash v : tstsx < tsx}(\forall E)_{\wp\mathcal{V}}}{(x : \mathcal{V}), (u : e \in sx), (v : tsx < x) \vdash (v)uv : \bot}(@)}{\dfrac{\dfrac{(x : \mathcal{V}), (u : e \in sx) \vdash \lambda v.(v)uv : tsx \not< x}{(x : \mathcal{V}) \vdash \lambda v.\lambda u.(v)uv : e \in sx \to x \in e}(\lambda)}{\vdash \lambda v.\lambda u.(v)uv : Ind(e)}(\forall I)_{\mathcal{V}}}(\lambda) \tag{B.0.17}$$

Similarly to the case above, the crucial part in the typing is constituted by the two extractions performed over the variable v of type $tsx < x$, that is, $\forall^{\wp\mathcal{V}(x \in stsx \to tsx \in x)}$. Given a type $\Psi(x)$, the constraint for the typing of the combinator has the following form:

$$\Psi(G_1(x)) = \Theta(x) \to \Psi(G_2(x)) \to \bot \tag{B.0.18}$$

for two substitutions G_1, G_2. Then the choice $\Psi(x) = x \in stsx \to tsx \in x$, $\Theta(x) := e \in sx$ and $G_1(x) = e$, $G_2(x) = d$ provides a solution to the equation above.

Step 3 We finally prove that Ω is not well-founded: suppose $WF(\Omega)$ holds; since the set E is inductive, it follows that $\Omega \in E$, i.e. that $ts\Omega \not< \Omega$. On the other hand, from the assumption $WF(\Omega)$, i.e. $\forall^{\wp\mathcal{V}} X(Ind(X) \to \Omega \in X)$ it follows that if the set F is inductive, then it is in Ω; this means, by equation (B.0.8), that if $X \in s\Omega$, then $ts\Omega \in X$, i.e. $ts\Omega < \Omega$.

From the argument above we extract the following typing of the combinator $\lambda w.w(\lambda u.\lambda v.(v)uv)w$:

$$\dfrac{\dfrac{\dfrac{(w : WF(\Omega)) \vdash w : WF(\Omega)}{(w : WF(\Omega)) \vdash Ind(e) \to ts\Omega \not< \Omega}\,(\forall E)_{\wp\mathcal{V}} \qquad \begin{matrix}\vdots \\ (w : WF(\Omega) \vdash \lambda u.\lambda v.(v)uv : Ind(e)\end{matrix}}{(w : WF(\Omega) \vdash w(\lambda u.\lambda v.(v)uv) : ts\Omega \not< \Omega}\,(@) \qquad \dfrac{(w : WF(\Omega)) \vdash w : WF(\Omega)}{(w : WF(\Omega) \vdash w : ts\Omega < \Omega}\,(\forall E)_{\wp\mathcal{V}}}{\dfrac{(w : WF(\Omega)) \vdash w(\lambda u.\lambda v.(v)uv)w : \bot}{\vdash \lambda w.w(\lambda u.\lambda v.(v)uv)w : \neg WF(\Omega)}\,(\lambda)}\,(@) \tag{B.0.19}$$

Once more, the core of the typing above lies in the extractions performed over the variable w of type $WF(\Omega)$. $WF(\Omega)$ can be written under the form $\forall^{\wp\mathcal{V}} X(\forall^{\mathcal{V}} x\Phi(x, X) \to \Xi(X))$, and the constraint has then the form

$$\forall^{\mathcal{V}} x\Phi(x, H_1(X)) \to \Xi(H_1(X)) = \forall^{\mathcal{V}} y(\Theta(y) \to \Psi(y) \to \bot) \to (\forall^{\mathcal{V}} x\Phi(x, H_2(X)) \to \Xi(H_2(X))) \to \bot \tag{B.0.20}$$

and one can choose the substitutions $H_1(X) = e$ and $H_2(X) = d$.

In definitive, we can sum up the three steps by saying that a typing of the not normalizing combinator W arises as soon as one can find types $\Phi(x, X), \Psi(x), \Theta(x), \Xi(X)$ (under the assumptions $x \in \mathcal{V}$ and $X \in \wp\mathcal{V}$) and substitutions $F_1, F_2, H_1, H_2, G_1, G_2$ such that the following specifications are satisfied:

$$\Phi(F_1(x), X) = \Phi(F_2(x), X) \to \tau \tag{B.0.21}$$
$$\Psi(G_1(x)) = \Theta(x) \to \Psi(G_2(x)) \to \bot \tag{B.0.22}$$
$$\Phi(x, H_1(X)) = \Theta(x) \to \Psi(x) \to \bot \tag{B.0.23}$$
$$\Xi(H_1(X)) = \forall^{\mathcal{V}} x(\Phi(x, H_2(X)) \to \Xi(H_2(X)) \to \bot \tag{B.0.24}$$

Appendix C

Simulating recursive functions by normal λ-terms

The representation of recursive function in lambda calculus that we'll adopt is a slight variation of the one in [BGP94].

C.1 A modified HGK-*computability*

In what follows, first we present a definition of recursive functions (which can be seen as a modified version of *Herbrand-Gödel-Kleene computability*) and we show its equivalence with the usual definition by means of minimalization; next, we show how to build a λ-representation of a recursive function defined in this way.

Definition C.1.1 (canonical system of equations)**.** *Let Σ be the union $\Sigma_0 \cup \Sigma_1$ of two disjoint sets of function symbols (of fixed arity m) $\Sigma_0 = \{\overline{0}, \overline{s}\}$ (of arity respectively* 0 *and* 1*) and $\Sigma_1 = \{f_1, \ldots, f_k\}$ that we call, respectively, the* data constructors *and the* programs*. Let $\mathcal{L}(\Sigma)$ be the language made of a countable set of variables $x, y, z, \ldots$ and the function symbols in Σ. A* canonical *system of equations $\mathcal{E}$ in $\mathcal{L}(\Sigma)$ is given by, for all $1 \le u \le k$, two equations $e_{u,1}, e_{u,2}$ of the form*

$$f_u(\overline{0}, y_1, \ldots, y_m) = b_{u,1} \qquad (e_{u,1})$$
$$f_u(\overline{s}(x), y_1, \ldots, y_m) = b_{u,2} \qquad (e_{u,2})$$

where $1 \le u \le k$, $f_u \in \Sigma_1$, $n \ge 0$ and, for $p \in \{1, 2\}$ and $b_{u,p}$ is a term in $\mathcal{L}(\Sigma)$ depending on the (all distinct) variables $y_1, \ldots, y_n$ (plus x in case $p = 2$) which belongs to the set $\mathcal{B} \subseteq \mathcal{L}(\Sigma)$ inductively defined below:

i. $\overline{0} \in \mathcal{B}$*;*

ii. $\overline{s}(b) \in \mathcal{B}$*, if* $b \in \mathcal{B}$*;*

iii. $y_l \in \mathcal{B}$*, for* $1 \le l \le m$*;*

iv. $f_u(x, b_1, \ldots, b_m) \in \mathcal{B}$*, if* $b_1, \ldots, b_m \in \mathcal{B}$*;*

v. $f_v(b, b_1, \ldots, b_m) \in \mathcal{B}$*, if* $b, b_1, \ldots, b_m \in \mathcal{B}$ *and* $1 \le v \le k$*.*

*For all $1 \leq u \leq k$, we call the first variable in the definition of f_u the $u-$*recursive variable*, and the other variables $y_1, \ldots, y_m$ the u*-parameters*.*

Remark that the definition of the terms in $\mathcal{B}$ is such that the $u-$recursive variable x appears as first argument of f_v only for $v = u$: this means that f_u is the only function having the right to perform recursion over x.

We say that a recursive function f is *defined* by a canonical system of equations $\mathcal{E}$ if, for all positive integers n, m, the equation $f(\underline{n}) = \underline{m}$ is derivable from the equations in $\mathcal{E}$ (in the sense of *Herbrand-Gödel-Kleene computability*, see [Lei90] Appendix I) if and only if $f(n)$ is defined and equal to m.

Proposition C.1.1. *For every partial recursive function f there exists a canonical system of equations $\mathcal{E}(f)$ defining it.*

Proof. First remark that it is enough to show the theorem without requiring that the function symbols f_u have a fixed arity (as in the definition), since any system of this kind can be turned into a canonical system by introducing "dummy" parameters in the equation so to fix an arity $m = max\{arity(f_u) | 1 \leq u \leq k\}$.

The cases of the zero, the successor and the projection functions are trivial. The composition $h(z_0, z_1, \ldots, z_m)$ of $f(x, y_1, \ldots, y_n)$ with $g_0(z_0, \vec{z}), \ldots, g_n(z_0, \vec{z})$ is obtained by adding to the equations of f and the g_i, $1 \leq i \leq n$ the equations

$$\begin{aligned} h(\overline{0}, \vec{z}) &= f(g_0(\overline{0}, \vec{z}), \ldots, g_n(\overline{0}, \vec{z})) \\ h(\overline{s}(x), \vec{z}) &= f(g_0(\overline{s}(x), \vec{z}), \ldots, g_n(\overline{s}(x), \vec{z})) \end{aligned} \tag{C.1.1}$$

Let us show how to define the minimalization $\mu f(y_1, \ldots, y_n)$ of a function $f(x, y_1, \ldots, y_n)$, for $n \geq 1$: let's define a new function $h(x, y_1, \ldots, y_n, y_{n+1})$ as follows

$$\begin{aligned} h(\overline{0}, y_1, \ldots, y_n, y_{n+1}) &= \overline{0} \\ h(\overline{s}(x), y_1, \ldots, y_n, y_{n+1}) &= \overline{s}\big(h(f(\overline{s}(y_{n+1})), y_1, \ldots, y_n, \overline{s}(y_{n+1}))\big) \end{aligned} \tag{C.1.2}$$

we can now define $\mu f(y_1, \ldots, y_n)$ by composition as $h(f(\overline{0}), y_1, \ldots, y_n, \overline{0})$.

□

C.2 Recursive functions by normal λ-terms

We pass now to show how to build normal solutions to canonical systems in pure lambda calculus. By a solution we mean a representation of the signature Σ by normal lambda terms $\mathbf{0}_B, \mathbf{s}_B, \mathbf{f}_1, \ldots, \mathbf{f}_k$ satisfying the equations in $\mathcal{E}(f)$. The crucial aspect of this representation is that also numerals will receive a non standard representation (as one can guess, the translation of *Church* numerals into such numerals will correspond to the "up" part of the accessibility proof).

The terms $\mathbf{0}_B, \mathbf{s}_B$ are defined as follows

$$\begin{aligned} \mathbf{0}_B &:= \lambda e.(e)U_1^2 \\ \mathbf{s}_B &:= \lambda x.\lambda e.(e)U_2^2 x \end{aligned} \tag{C.2.1}$$

For every $n \in \mathbb{N}$, the *BGP integers* $\mathbf{n}_B$ (from the authors of [BGP94]) is then the term below:

$$\mathbf{n}_B := \underbrace{\lambda e.(e)U_2^2\big(\lambda e.(e)U_2^2\big(\ldots \lambda e.(e)U_2^2}_{n \text{ times}}\,\mathbf{0}_B\big)\ldots\big) \tag{C.2.2}$$

which are the normal forms of the terms $(\mathbf{s}_B)^n\mathbf{0}_B$.

We define now, for every term $t \in \mathcal{B}$, a representation $\mathbf{t}$ which will be a normal term in lambda calculus. Let's fix $k+m$ distinct variables $z_1,\dots,z_k,y_1,\dots,y_m$ (remark here again an abuse of notation, since y_l is at the same time a first-order variable and a variable in λ-calculus); we proceed by induction on the terms t and show that, unless $\mathbf{t}$ is a *Church numeral*, it does not begin with a lambda:

i. if t is $\overline{0}$, then $\mathbf{t}$ is just $\mathbf{0}$;

ii. if t is $\overline{s}(t')$ for a certain term t', then $\mathbf{t}$ is $\lambda f.\lambda x.\mathbf{t}'f(fx)$, i.e. the *Church successor* of t', if t is not a *Church numeral*, else it is just its successor; remark that, if t is not a Church numeral, then by i.h. $\mathbf{t}'$ does not begin with a λ, so $\mathbf{t}$ is normal;

iii. if t is y_l, for a certain $1 \leq l \leq m$, then $\mathbf{t}$ is y_l;

iv. if t is $f_u(x,t_1,\dots,t_m)$ for certain terms $t_1,\dots,t_m$, then $\mathbf{t}$ is $(x)z_u z_1\dots z_k\mathbf{t}_1\dots\mathbf{t}_m$ (which does not begin with a λ);

v. if t is $f_v(t_0,t_1,\dots,t_m)$ for certain terms $t_0,t_1,\dots,t_m$ and $1 \leq v \leq k$, then three subcases arise:

- if $t_0 = \overline{0}$, then $\mathbf{t}$ is $(z_v)U^2_1 z_1\dots z_k\mathbf{t}_1\dots\mathbf{t}_m$ (which is the normal form of $(\mathbf{0}_B)z_v z_1\dots z_k\mathbf{t}_1\dots\mathbf{t}_m$);
- if $t_0 = (\overline{s})t'_0$, then $\mathbf{t}$ is $(z_i)U^2_2\mathbf{t}'_0 z_1\dots z_k\mathbf{t}_1\dots\mathbf{t}_m$ (which is the normal form of $(\mathbf{s}_B\mathbf{t}_0)z_v z_1\dots z_k\mathbf{t}_1\dots\mathbf{t}_m$);
- in all other cases $\mathbf{t}$ is $((\mathbf{t}_0)\mathbf{s}_B\mathbf{0}_B)z_1\dots z_k\mathbf{t}_1\dots\mathbf{t}_m$ (remark that in this case $\mathbf{t}_0$ does not begin with a λ, thus $\mathbf{t}$ is normal).

In all three cases $\mathbf{t}$ does not begin with a λ.

By the translation above we can define, for all $1 \leq u \leq k$, $1 \leq p \leq 2$ a (closed) lambda term $M_{u,p}$ as follows

$$\begin{aligned} M_{u,1} &:= \lambda z_1.\dots.\lambda z_k.\lambda y_1.\dots.\lambda y_m.\mathbf{b}_{u,1} \\ M_{u,2} &:= \lambda x.\lambda z_1.\dots.\lambda z_k.\lambda y_1.\dots.\lambda y_m.\mathbf{b}_{u,2} \end{aligned} \tag{C.2.3}$$

We fix then, for $1 \leq i \leq k$,

$$M_u := \langle M_{u,1}, M_{u,2}\rangle = \lambda z.(z)M_{u,1}M_{u,2} \tag{C.2.4}$$

and finally

$$\mathbf{f}_u := \langle M_u, M_1,\dots,M_k\rangle = \lambda z.(z)M_u M_1\dots M_k \tag{C.2.5}$$

We prove now that the $\mathbf{f}_u$ are indeed a representation of the functions f_u:

Proposition C.2.1. *The terms $\mathbf{0}_B, \mathbf{s}_B$ along with the $\mathbf{f}_u$, for $1 \leq u \leq k$, satisfy the system $\mathcal{E}(f)$;*

Proof. For the first point let

$$f_u(\overline{s}(x),y_1,\dots,y_n) = b_{u,2} \tag{C.2.6}$$

be an equation in $\mathcal{E}(f)$; we treat only the case with $p=2$, the case with $p=1$ being a mere reformulation of the former.

$$\begin{aligned} (\mathbf{f}_u)(\overline{s}_B x)y_1\dots y_n &\rightsquigarrow (\overline{s}_B x)M_i M_1\dots M_k y_1\dots y_k \rightsquigarrow (M_u)U^2_2 x M_i M_1\dots M_k y_1\dots y_k \rightsquigarrow \\ &\rightsquigarrow (U^2_2)M_{u,1}M_{u,2}xM_u M_1\dots M_k y_1\dots y_k \rightsquigarrow (M_{u,2})xM_u M_1\dots M_k y_1\dots y_k \rightsquigarrow \\ &\rightsquigarrow \mathbf{b}_{u,2}[M_1/z_1,\dots,M_k/z_k] \end{aligned} \tag{C.2.7}$$

Now, if $b^*_{u,2}$ is the term obtained by substituting in $b_{u,2}$ all the occurrences of terms in Σ with their representations, one easily verifies that $b^*_{u,2}$ is β-equivalent to $\mathbf{b}_{u,2}[M_1/z_1, \ldots, M_k/z_k]$.

$\square$

Finally, in order to let our desired representation work over Church numerals, we have to define a *coding* $\sharp$ from Church to BGP numerals, and define, for $1 \leq u \leq k$,

$$\hat{\mathbf{f}}_u := \lambda z_0.\lambda z_1.\ldots.\lambda z_m.\mathbf{f}_u(\sharp z_0)z_1 \ldots z_m =_\beta \lambda z_0.\lambda z_1.\ldots.\lambda z_m.(\sharp z_0)M_u M_1 \ldots M_k z_1 \ldots z_m \quad \text{(C.2.8)}$$

The coding function is easily defined by iteration:

$$\sharp x = (x)\mathbf{s}_B\mathbf{0}_B \quad \text{(C.2.9)}$$

In definitive $\hat{\mathbf{f}}$ provides our desired representation (remark that $\hat{\mathbf{f}}$ is a normal λ-term).

Bibliography

[AL91] Andrea Asperti and Giuseppe Longo. *Categories, types and structures: an introduction to category theory for the working computer scientist.* The M.I.T. Press, 1991.

[Als86] William P. Alston. Epistemic circularity. *Philosophy and Phenomenological Research*, 47(1):1–30, 1986.

[BAGM92] Henk Barendregt, S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum. Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992.

[Bar85] Henk Barendregt. *Lambda calculus, its syntax and semantics.* North-Holland, 1985.

[BBJ07] George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and logic.* Cambridge University Press, 2007.

[Ber88] Stefano Berardi. Towards a mathematical analysis of the Coquand-Huet calculus of constructions and the other systems in Barendregt's cube. Technical report, Department of Computer Science, CMU and Dipartimento di Matematica, Università di Torino, 1988.

[BF97] Cesare Burali-Forti. Una questione sui numeri transfiniti. In *Rendiconti del Circolo Matematico di Palermo*, volume 11, pages 154–164, 1897.

[BFSS90] E.S. Bainbridge, Peter J. Freyd, Andre Scedrov, and Philip J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70:35–64, 1990.

[BGP94] Corrado Bohm, Stefano Guerrini, and Adolfo Piperno. λ-definition of funcion(al)s by normal forms. In *Programming Languages and Systems - ESOP '94*, number 188 in Lecture Notes in Computer Science, pages 135–149, 1994.

[Bog96] Paul Artin Boghossian. Analyticity reconsidered. *Noûs*, 30(3):360–391, 1996.

[Bog03] Paul Artin Boghossian. Epistemic analyticity: a defense. *Grazer Philosophische Studien*, 66(1):15–35, 2003.

[Boo75] George Boolos. On second order logic. *The journal of philosophy*, 72(16):509–528, 1975.

[Bor00] Jorge Luis Borges. The library of Babel. In *The total library: Non-fiction 1922-1986*, pages 214–216. Allen Lane The Penguin Press, 2000.

[Car37] Rudolf Carnap. *The Logical Syntax of Language.* K. Paul Trench, 1937.

[Car83] Rudolf Carnap. The logicist foundations of mathematics (1931). In Paul Benacerraf and Hilary Putnam, editors, *Philosophy of Matemathics: selected readings*, pages 41–52. Cambridge University Press, 1983.

[CC91] Felice Cardone and Mario Coppo. Type inference with recursive types: syntax and semantics. *Information and Computation*, 92:48–80, 1991.

[CF58] Haskell B. Curry and Robert Feys. *Combinatory logic. Vol. I.* North-Holland, 1958.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, pages 56–68, 1940.

[Cof91] Alberto Coffa. *The semantic tradition from Kant to Carnap.* Cambridge University Press, 1991.

[Coq86] Thierry Coquand. An analysis of Girard's paradox. In *LICS*, 1986.

[Coq90] Thierry Coquand. Metamathematical investigations of a Calculus of Constructions. In P. Odifreddi, editor, *Logic in Computer Science.* Academic Press, 1990.

[Coq94] Thierry Coquand. A new paradox in type theory. In *9th International Conference on Logic, Methodology and Philosophy of Science*, pages 555–570, 1994.

[Coq05] Thierry Coquand. Completeness theorems and λ-calculus. In *Typed Lambda Calculi and Applications*, volume 3461 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin Heidelberg, 2005.

[Cro04] Tristan Crolard. A type theory which is complete for Kreisel's modified realizability. In *Third workshop on automated verification of critical systems (AVoCS'2003).* Springer-Verlag, 2004.

[DB70] Nicolaas Govert De Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In *Symposium on Automatic Demonstration (Versailles 1968)*, volume 125 of *Lecture Notes in Mathematics*, pages 29–61. Springer, 1970.

[Ded96] Richard Dedekind. Was sind und was sollen die Zahlen? (english translation). In William Erwald, editor, *From Kant to Hilbert: a source book in the Foundations of Mathematics, vol. 2*, pages 787–833. Clarendon Press, 1996.

[Dos] Kosta Dosen. Inferential semantics. To appear in: H. Wansing ed., Dag Prawitz on Proofs and Meaning, Springer, Berlin.

[Dum91a] Michael Dummett. *Frege: philosophy of mathematics.* Duckworth, 1991.

[Dum91b] Michael Dummett. *The logical basis of metaphysics.* Columbia University Press, 1991.

[Dum06] Michael Dummett. The vicious circle principle. In *Cambridge and Vienna: Frank P. Ramsey and the Vienna Circle*, volume 12 of *Vienna Circle Institute Yearbook*, pages 29–33. Springer, 2006.

[Fef62] Solomon Feferman. Transfinite recursive progressions of axiomatic theories. *Journal of Symbolic Logic*, 27(3):259–316, 1962.

[Fre50] Gottlob Frege. *The foundations of arithmetic.* Basil Blackwell, 1950.

[Fre80] Gottlob Frege. *Philosophical and mathematical correspondence.* Basil Blackwell, 1980.

[Fre13] Gottlob Frege. *The Basic Laws of Arithmetic (1893).* Oxford University Press, 2013.

[Fri78] Harvey Friedman. Classically and intuitionistically provably recursive functions. *Higher Set Theory*, 669:21–28, 1978.

[G4̈4] Kurt Gödel. Russell's mathematical logic. In P. A. Schilpp, editor, *The philosophy of Bertrand Russell*, pages 123–153. Northwestern University, 1944.

[G5̈8] Kurt Gödel. über eine bisher noch nicht benütze Erweiterung des finiten Standpunktes. *Dialectica*, pages 280–287, 1958.

[Gal90] Jean Gallier. On Girard's "candidats de réductibilité". *Logic and Computer Science*, 1990.

[Gal95] Jean Gallier. Proving properties of typed λ-terms using realizability, covers and sheaves. *Theoretical Computer Science*, 142, 1995.

[Gen64] Gerhard Gentzen. Investigations into logical deduction (1934). *American Philosophical Quarterly*, 1(4):288–306, 1964.

[Gen69] Gerhard Gentzen. Provability and nonprovability of restricted transfinite induction in elementary number theory. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen.* North-Holland, 1969.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmetique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

[Gir76] Jean-Yves Girard. Three-valued logic and cut-elimination: the actual meaning of Takeuti's conjecture. *Dissertationes mathematicae*, 1976.

[Gir85] Jean-Yves Girard. Introduction to Π^1_2-logic. *Synthese*, 62(2):191–216, 1985.

[Gir86] Jean-Yves Girard. The System F of variable types, fifteen years later. *Theoretical Computer Science*, 45(2):159–192, 1986.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[Gir89a] Jean-Yves Girard. Geometry of interaction I: interpretation of system f. In Ferro, Bonotto, Valentini, and Zanardo, editors, *Logic colloquium*, 1989.

[Gir89b] Jean-Yves Girard. Proof theory and logical complexity, vol. 2. (unpublished), 1989.

[Gir89c] Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92, 1989.

[Gir90a] Jean-Yves Girard. Geometry of interaction II: deadlock-free algorithms. In *International Conference on Computational Logic, Tallinn*, 1990.

[Gir90b] Jean-Yves Girard. *Proof theory and logical complexity, Vol. 1.* Studies in proof theory. Elsevier Science, 1990.

[Gir91] Jean-Yves Girard. A new constructive logic: classical logic. Technical report, INRIA, 1991.

[Gir95] Jean-Yves Girard. Geometry of interaction III: accomodating the additives. In *Advances in Linear Logic, London Mathematical Society, Lecture Note Series*. Cambridge University Press, 1995.

[Gir00] Jean-Yves Girard. Du pourquoi au comment: la théorie de la démonstration de 1950 á nos jours. In *Developments of mathematics 1950-2000*, pages 515–545. Birkhäuser, Basel, 2000.

[Gir06] Jean-Yves Girard. Geometry of interaction IV: the feedback equation. In *Logic Colloquium 2003, Association of Symbolic Logic*, 2006.

[Gir10] Jean-Yves Girard. Geometry of interaction V: logic in the hyperfinite factor. *Theoretical Computer Science*, 2010.

[Gir11] Jean-Yves Girard. *The blind spot*. European Mathematical Society, 2011.

[Gir13] Jean-Yves Girard. Geometry of interaction VI: a blueprint for transcendental syntax. Under consideration for publication in Mathematical Structures in Computer Science, 2013.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[Gol81] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[GRDR88] Paola Giannini and Simona Ronchi Della Rocca. Characterization of typings in polymorphic type discipline. In *Proceedings of the 3-th Annual IEEE Symposium on Logic in Computer Science*, pages 61–70, Edinburgh, 1988.

[GRDR91] Paola Giannini and Simona Ronchi Della Rocca. *Type inference in polymorphic type discipline*. Theoretical Aspects of Computer Software, International Conference TACS '91, Sendai, Japan, 1991.

[GS89] Jean Gallier and Wayne Snyder. Higher-order unification revisited: complete sets of transformations. *Journal of Symbolic Computation*, 8(1-2):101–140, 1989.

[GSS92] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Normal forms and cut-free proofs as natural transformations. In Y. Moschovakis, editor, *Logic from Computer Science*, volume 21, pages 217–241. Springer-Verlag, 1992.

[GV09] Herman Geuvers and Joep Verkoelen. On fixed point and looping combinators in type theory. `http://www.cs.ru.nl/~herman/PUBS/TLCApaper.pdf`, 2009.

[Hen88] Fritz Henglein. Type inference and semi-unification. In *Proceedings of the 1988 ACM Conference on Lisp and Functional Programming*, pages 184–197. ACM press, 1988.

[Hen89] Fritz Henglein. *Polymorphic type inference and semi-unification*. PhD thesis, The State University of New Jersey, 1989.

[Her67] Jacques Herbrand. Investigations in proof theory. In Jean Van Heijenoort, editor, *From Frege to Gödel: a source book in mathematical logic, 1879-1931*. Harvard University Press, 1967.

[Her71] Jacques Herbrand. On the consistency of arithmetics (1931). In Warren D. Goldfarb, editor, *Jacques Herbrand, Logical Writings*. Harvard University Press, 1971.

[Hey56] Arend Heyting. *Intuitionism, an introduction.* North-Holland, 1956.

[Hil96a] David Hilbert. The logical foundations of mathematics (1923). In William Erwald, editor, *From Kant to Hilbert: a source book in the Foundations of Mathematics.* Clarendon Press, 1996.

[Hil96b] David Hilbert. Mathematical problems (1900). In William Ewald, editor, *From Kant to Hilbert: a source book in the Foundations of Mathematics*, volume II. Clarendon Press, 1996.

[Hin69] Roger J. Hindley. The principal type scheme of an object in combinatory logic. In *Transactions of the American Mathematical Society*, volume 146, pages 29–60, 1969.

[Hin83] Roger J. Hindley. The completeness theorem for typing λ-terms. *Theoretical Computer Science*, 22(1-2):1–17, 1983.

[How80] William A. Howard. The formula-as-types notion of construction (1969). In *To H.B. Curry. Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980.

[Hur95] Antonius J. C. Hurkens. A simplification of Girard's paradox. In *TLCA 95, Second International Conference on Typed Lambda Calculi and Applications*, pages 266–278, 1995.

[Hyl82] Martin Hyland. The effective topos. In Troelstra and Van Dalen, editors, *The L.E.J. Brouwer Centenary Symposium*, volume 216, 1982.

[Joi07] Jean-Baptiste Joinet. Logique et interaction. Thèse d'habilitation à diriger des recherches, Université Denis Diderot (Paris 7), 2007.

[Joi09] Jean-Baptiste Joinet. Ouvrir la logique au mond. In *Ouvrir la logique au mond: Philosophie et Mathématique de l'interaction.* Hermann et CCI-Cerisy, 2009.

[Joi11] Jean-Baptiste Joinet. Logique et métaphysique. In *O que è metafisica?* Editora da Universidade Federal do Rio Grande do Norte, 2011.

[KL68] Georg Kreisel and Azriel Levy. Reflection principles and their use for establishing the complexity of axiomatic systems. *Mathematical Logic Quarterly*, 14(7-12):97–142, 1968.

[Kle45] Stephen Cole Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10(4):109–124, 1945.

[Kle52] Stephen Cole Kleene. *Introduction to meta-mathematics.* North-Holland, 1952.

[Kle59] Stephen Cole Kleene. Countable functionals. In Arend Heyting, editor, *Constructivity in mathematics*, pages 81–100. North-Holland, 1959.

[Kle73] Stephen Cole Kleene. Realizability, a retrospective survey. In *Cambridge Summer School in Mathematical Logic (1971)*, volume 337 of *Lecture Notes in Mathematics*, pages 95–112. Springer, 1973.

[Kre59] Georg Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In Arend Heyting, editor, *Constructivity in mathematics*, pages 101–128. North-Holland, 1959.

[Kre60] Georg Kreisel. Foundations of intuitionistic logic. In *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, pages 198–210. Stanford University Press, 1960.

[Kre65] Georg Kreisel. Mathematical logic. In *Lectures on modern mathematics, Vol. III*, pages 95–195. Wiley, 1965.

[Kre70] Georg Kreisel. Church's thesis: a kind of reducibility axiom for constructive mathematics. In *Intuitionism and Proof Theory: proceedings of the summer conference at Buffalo, N.Y.* North-Holland, 1970.

[Kri] Jean-Louis Krivine. On the structure of classical realizability models of ZF. arXiv:1408.1868 [cs.LO].

[Kri93] Jean-Louis Krivine. *Lambda calculus, types and models.* Ellis Horwood, 1993.

[Kri09] Jean-Louis Krivine. Realizability in classical logic. In *Interactive models of computation and program behaviour. Panoramas et synthèses.* Société Mathématique de France, 2009.

[Kri11] Jean-Louis Krivine. Realizability algebras: a program to well order R. *Logical Methods in Computer Science*, 3(2):1–47, 2011.

[Kri12] Jean-Louis Krivine. Du programme de hilbert aux programmes informatiques. Leçons de mathématiques d'aujourd'hui, Bordeaux, 2012.

[KT74] Georg Kreisel and Gaisi Takeuti. Formally self-referential propositions for cut free analysis and related systems. *Dissertationes mathematicae*, 118, 1974.

[KTU93] Assaf J. Kfoury, Jerzy Tiuryn, and Pawel Urzyczyn. The undecidability of the semi-unification problem. *Information and Computation*, 102(1):83–101, 1993.

[LC89] Philippe Le Chenadec. On the logic of unification. *Journal of Symbolic Computation*, 8(1-2):141–199, 1989.

[Lei83] Daniel Leivant. Reasoning about functional programs and complexity classes associated with type disciplines. In *24th Annual Symposium on the Foundations of Computer Science*, pages 460–469, 1983.

[Lei90] Daniel Leivant. Contracting proofs to programs. In P. Odifreddi, editor, *Logic and Computer Science*, volume 11, pages 279–327. Academic Press, 1990.

[Lei94] Daniel Leivant. Higher order logic. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of logic in artificial intelligence and logic programming*, volume 2, pages 229–231. Oxford University Press, 1994.

[Lei01] Daniel Leivant. Peano's lambda calculus: the functional abstraction implicit in arithmetic. In *Logic, meaning and computation, Essays in Memory of Alonzo Church*, volume 305 of *Synthese Library, Studies in Epistemology, Logic, Methodology and Philosophy of Science*, pages 313–329. Springer Netherlands, 2001.

[LF97] Giuseppe Longo and Thomas Fruchart. Carnap's remarks on impredicative definitions and the genericity theorem. In *Logic, Methodology and Philosophy of Science: Logic in Florence*. Kluwer, 1997.

[LMS93] Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. The genericity theorem and the notion of parametricity in the polymorphic λ-calculus. *Theoretical Computer Science*, 121:323–349, 1993.

[LS86] Joachim Lambek and Philip J. Scott. *Introduction to higher order categorical logic.* Cambridge University Press, 1986.

[Mal90] Sophie Malecki. Generic terms having no polymorphic types. In *Automata, Languages and Programming, 17th International Colloquium Warwick University, England*. Springer Berlin Heidelberg, 1990.

[Mal92] Sophie Malecki. *Quelques résultats sur la typabilité dans le système F.* PhD thesis, Université Paris 7, 1992.

[Mal97] Sophie Malecki. Proofs in system F_ω can be done in system F^1_ω. In *Computer Science Logic*, volume 1258 of *Lecture Notes in Computer Science*, pages 297–315. Springer, 1997.

[MD82] R. Milner and L. Damas. The principal type schemes for functional programs. In *Symposium on Principles of Programming Languages, ACM*, 1982.

[Men87] Paul Francis Mendler. *Inductive definitions in Type Theory.* PhD thesis, Cornell University, 1987.

[Mil78] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Science*, 17:348–374, 1978.

[Min78] G. E. Minc. Finite investigations of infinite derivations. *Journal of Soviet Mathematics*, 10:548–596, 1978.

[Mit86] John C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions. In *Proceedings of the 1986 ACM Symposium on Lisp and Functional Programming*, pages 308–319, 1986.

[ML70a] Per Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J.E. Fenstad, editor, *Proceedings of the 2nd Scandinavian Logic Symposium (Oslo)*, 1970.

[ML70b] Per Martin-Löf. A theory of types. Unpublished manuscript, 1970.

[ML75] Per Martin-Löf. An intuitionistic theory of types: predicative part. In *Proceedings of the Logic Colloquium 1973, Bristol*, volume 80 of *Studies in logic and the foundations of mathematics*, pages 73–118. North-Holland, 1975.

[ML84] Per Martin-Löf. *Intuitionistic type theory.* Bibliopolis, 1984.

[ML86] Per Martin-Löf. Unifying scott's theory of domains for denotational semantics and intuitionistic type theory (abstract). In CLUEB, editor, *Atti del congresso di logica e filosofia della scienza, San Gimignano 7-11 settembre 1983*, 1986.

[ML87] Per Martin-Löf. Truth of a proposition, evidence of a judgement, validity of a proof. *Synthese*, 73:407–420, 1987.

[ML98] Per Martin-Löf. Truth and knowability: on the principles c and k of Michael Dummett. In H.G. Dales, editor, *Truth in mathematics*. Clarendon Press, 1998.

[NPS14] Alberto Naibo, Mattia Petrolo, and Thomas Seiller. On the computational meaning of axioms. `halshs.archives-ouvertes.fr/hal-00930222/`, 2014.

[Pal90] Erik Palmgren. Domain interpretations of martin-löf's partial type theory. *Annals of Pure and Applied Logic*, 48:135–196, 1990.

[Par93] Michel Parigot. Classical proofs as programs. In *Kurt Gödel Colloquium*, volume 713 of *Lecture Notes in Computer Science*, pages 263–276. Springer-Verlag, 1993.

[Pea93] Cristopher Peacocke. Proof and Truth. In J. Haldane and C. Wright, editors, *Reality, Representation and Projection*. Oxford University Press, 1993.

[Pfe88] Frank Pfenning. Partial polymorphic type inference and higher-order unification. In *Proceedings of the ACM Conference on Lisp and Functional Programming*, pages 153–163. ACM press, 1988.

[Poh89] Wolfram Pohlers. *Proof theory: the first step into impredicativity*, volume 1407 of *Lecture Notes in Mathematics*. Springer, 1989.

[Poi06] Henri Poincaré. Les mathématiques et la logique. *Revue de Métaphysique et de Morale*, 14(3):294–317, 1906.

[Pra65] Dag Prawitz. *Natural deduction, a proof-theoretical study*. Almqvist & Wiskell, 1965.

[Pra68] Dag Prawitz. Hauptsatz for higher order logic. *The Journal of Symbolic Logic*, 33(3):452–457, 1968.

[Pra71a] Dag Prawitz. Ideas and results in proof theory. In J.E. Fenstad, editor, *Proceedings of the 2nd Scandinavian Logic Symposium (Oslo), Studies in logic and foundations of mathematics*, volume 63. North-Holland, 1971.

[Pra71b] Dag Prawitz. Towards a foundation of a general proof theory. *Logic, Methodology and Philosophy of Science*, VI, 1971.

[Pra74] Dag Prawitz. On the idea of a general proof theory. *Synthese*, 27:63–77, 1974.

[Pra02] Dag Prawitz. Problems for the generalization of a verificationist theory of meaning. *Topoi*, 21(2-1):87–92, 2002.

[Pra12] Dag Prawitz. Truth as an epistemic notion. *Topoi*, 31(1):9–16, 2012.

[Pri67] Arthur Prior. The runabout inference ticket. In Peter Strawson, editor, *Philosophical logic*, pages 38–39. Oxford University Press, 1967.

[PW78] Michael S. Paterson and Mark N. Wegman. Linear unification. *Journal of Computer and System Science*, 16:158–167, 1978.

[Qui53] Willard Van Orman Quine. Two dogmas of empiricism (1951). In *From a logical point of view*. Harvard University Press, 1953.

[Qui76] Willard Van Orman Quine. Truth by convention (1936). In *The ways of paradox*. Harvard University Press, 1976.

[Qui80] Willard Van Orman Quine. Logic and the reification of universals. In *From a logical point of view*, pages 102–129. Harvard University Press, 1980.

[Qui86] Willard Van Orman Quine. *Philosophy of logic*. Harvard University Press, 1986.

[Ram31] Frank Plumpton Ramsey. *Foundations - Essays in Philosophy, logic, mathematics and economics*. Humanities Press, 1931.

[Rey74] John C. Reynolds. Towards a theory of type structure. In *Programming Symposium*, pages 408–423. Springer-Verlag, 1974.

[Rey83] John C. Reynolds. Types, abstraction and parametric polymorphism. In R.E.A. Mason, editor, *Information Processing '83*, pages 513–523. North-Holland, 1983.

[Rey84] John C. Reynolds. Polymorphism is not set-theoretic. In *Semantics of data types, International Symposium Sophia-Antipolis, France, June 1984*, volume 173 of *Lecture Notes in Computer Science*, pages 145–156. Springer Berlin Heidelberg, 1984.

[Rob65] Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Rus06a] Bertrand Russell. Les paradoxes de la logique. *Revue de Métaphysique et de Morale*, 14(5):294–317, 1906.

[Rus06b] Bertrand Russell. On some difficulties in the theory of transfinite numbers and order types. In *Proceedings of the London Mathematical Society*, volume 4, pages 29–53, 1906.

[Rus08] Bertrand Russell. Mathematical logic as based on the theory of types. *Americal Journal of Mathematics*, 30(3), 1908.

[Sch56] Kurt Schütte. Ein system des verknüpfeden Schliessens. *Archiv für mathematische Logic und Grundlagenforschung*, 2(2-4):55–67, 1956.

[Sch60] Kurt Schütte. Syntactical and semantical properties of simple type theory. *The Journal of Symbolic Logic*, 25(4):305–326, 1960.

[Sco68] Dana Scott. Constructive validity. In *Symposium on Automatic Demonstration*, Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1968.

[Sea69] John Searle. *Speach acts: an essay in the philosophy of language*. Cambridge University Press, 1969.

[SH91] Peter Schroeder-Heister. Uniform proof-theoretic semantics for the logical constants, abstract. *Journal of Symbolic Logic*, 56:1142, 1991.

[SH06] Peter Schroeder-Heister. Validity concepts in proof-theoretic semantics. *Synthese*, 148:525–571, 2006.

[SH12] Peter Schroeder-Heister. Proof-Theoretic Semantics. Entry for the Stanford Encyclopedia of Philosophy, `http://plato.stanford.edu/entries/proof-theoretic-semantics/`, 2012.

[Sha00] Stewart Shapiro. *Foundations without foundationalism: a case for second order logic.* Oxford Logic Guides. Oxford University Press, 2000.

[ST00] Helmut Schwichtenberg and Anne Sjerp Troelstra. *Basic proof theory.* Cambridge University Press, 2000.

[Sta73] John Staples. Combinator realizability of constructive finite type analysis. In *Cambridge Summer School in Mathematical Logic (1971)*, pages 253–273. Springer, 1973.

[Str67] Christopher Strachey. Fundamental concepts in programming languages. *Higher Order and Symbolic Computation*, 13:11–49, 1967.

[SU06] Morten Heine Sorensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*, volume 149 of *Studies in logic and the foundations of mathematics.* Elsevier Science, 2006.

[Sun83] Göran Sundholm. Construction, proofs and the meaning of the logical constants. *Journal of Philosophical Logic*, 12(2):151–172, 1983.

[Sun98] Göran Sundholm. Proofs as acts and proofs as objects: some questions for Dag Prawitz. *Theoria*, 64(2-3):187–216, 1998.

[Sun99] Göran Sundholm. Intuitionism and logical tolerance. In *Alfred Tarski and the Vienna Circle*, volume 6 of *Vienna Circle Institute Yearbook*, pages 135–148. Springer, 1999.

[Tai67] William W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

[Tai68] William W. Tait. A nonconstructive proof of Gentzen's Hauptsatz for second order predicate logic. *Journal of Symbolic Logic*, 33(2):289–290, 1968.

[Tai75] William W. Tait. A realizability interpretation of the theory of species. In *Proceedings of the Logic Colloquium*, volume 435 of *Lecture Notes in Mathematics*, pages 240–251, 1975.

[Tak57] Gaisi Takeuti. On a generalized logical calculus. *Journal of Symbolic Logic*, 22(4):351–352, 1957.

[Tak67] Moto-o Takahashi. A proof of cut-elimination theorem in simple type-theory. *Journal of the Mathematical Society of Japan*, 19(4):399–410, 1967.

[Tar83] Alfred Tarski. On the concept of logical consequence. In *Logic, Semantics, Metamathematics*, (translation of "Uber den Begriff der logischen Folgerung", in Actes du Congrès International de Philosophie Scientifique, fasc. 7, Paris, Hermann et Cie, 1936). Hackett, 1983.

[Tro63] Anne Sjerp Troelstra. *Metamathematical investigations of intuitionistic arithmetic and analysis.* Springer, 1963.

[Tro69] Anne Sjerp Troelstra. *Principles of intuitionism.* Lecture Notes in Mathematics. Springer-Verlag, 1969.

[TVD88] Anne Sjerp Troelstra and Dirk Van Dalen. *Constructivism in mathematics, vol. 2*, volume 123 of *Studies in logic and the foundations of mathematics.* North-Holland, 1988.

[Urz97] Pawel Urzyczyn. Type reconstruction in F^{ω}. *Mathematical Structures in Computer Science*, 7(4):329–358, 1997.

[Vaa01] Jouko Vaananen. Second-order logic and foundations of mathematics. *Bulletin of Symbolic Logic*, 7(4):504–520, 2001.

[VO02] Jaap Van Oosten. Realizability: a historical essay. *Mathematical Structures in Computer Science*, 12:239–263, 2002.

[VO08] Jaap Van Oosten. *Realizability: an introduction to its categorical side*, volume 152 of *Studies in logic and the foundations of mathematics*. Elsevier, 2008.

[Wad89] Philop Wadler. Theorems for free! In *Proceedings of the fourth international conference on functinoal programming languages and computer architecture - FPCA '89*, 1989.

[Wel98] J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98:111–156, 1998.

[Wit78] Ludwig Wittgenstein. *Remarks on the Foundations of Mathematics (1956)*. Blackwell, 1978.

[Wit89] Ludwig Wittgenstein. *Lectures on the Foundations of Mathematics (1939)*. Cambridge University Press, 1989.

[Wit01] Ludwig Wittgenstein. *Tractatus logico-philosophicus (1921)*. Routledge, 2001.

[Wit09] Ludwig Wittgenstein. *Philosophical Investigations (1953)*. Wiley-Blackwell, 2009. German text, with an English translation by G.E.M. Anscombe, P.M.S. Hacker and Joachim Schulte.