Università degli studi Roma Tre

Tesi magistrale

On the computational content of proofs

Author:
Davide Barbarossa

Supervisor:
Lorenzo Tortora de Falco

III Sessione - II Appello A.A. 2017/18
21/03/2019

Dipartimento di Matematica e Fisica

Contents

Introduction

Keep computations to the lowest
level of the multiplication table.
D. Hilbert

One of the most essential features of mathematics is certainly to produce proofs.
In this thesis we shall deal with (some of) the mathematics that arises when considering proofs as mathematical objects themselves.
This quite large branch of mathematics (with its own peculiar algebraic and graph-theoretical taste, going from category theory to computational complexity theory, from set theory to programming languages,...) goes under the name of "proof theory".
We will obviously consider only a fragment of this discipline, trying to show in this particular fragment, a fundamental ingredient of all its the modern developments, with which it is often so much linked to be quite a whole, and which is theoretical computer science.
One could (and should, as always) ask: "why is this worth it?" The answers are many:
- it has several applications to computer science, which would be already enough view the importance of the subject nowadays (the first application being the conception itself of the very notion of "computer", by A. Turing back in the 30's);
- it creates new interesting mathematics and it relates some different areas with each other;
- it is extremely important from a foundational point of view. In fact it was firstly introduced by D. Hilbert as an - a posteriori wrong - antidote to the foundational crisis of the beginning of the XX^{th} century, and modern proof theory gives new ideas and provides a deep insight in the very essence of mathematics and of computer science.

Our attitude will be like the one of the physicist when he wants to describe the external world in a mathematical way; we will instead try to mathematically describe the final result of the process that is a proof. As the theoretical physicist's main reason to do so is - typically - a purely disinterested exercise of intellect, so it will be for us. And as physics finds lots of practical applications, so does our exercise of intellect: you find them mainly in computer science. However, we will not investigate the applications, and our approach remains disinterested, assuming more the point of view of the mathematician than that of the computer scientist[1]. While the mathematics arising in physics is - essentially - continuous, mathematics in proof theory (and in logic in general) is - essentially - discrete[2]: this is mainly due to the fact that a proof is, at least intuitively, always conceived as a *finite* object.
Beginning with a comparison between physics and logic is, the author thinks, interesting, since he finds some similarities between the figure of the physicist and that of the logician - clearly only from certain[3] points of view.
Out of that, it must be said that the discipline of proof theory is relatively young, in that it has

[1] It is in interesting and funny to mention how history made it happen that this kind of research, even the purely theoretical one which origins from a sort of philosophical attitude, is often done in computer science departments - take for instance France - more than in mathematics ones...

[2] One could pretty much identify an important part of theoretical computer science with discrete mathematics. This clarifies a little bit why we often find all this research and also many other important areas of discrete mathematics in computer science departments.

[3] There may be also technical profound similarities: with the advent of category theory and of the theory of "proof-nets", physicists started to understand that there could be some interesting common point. See for example "Physics, Topology, Logic and Computation: A Rosetta Stone", by John Baez and Michael Stay, 2009. It is the author belief that an interaction between this two communities would bring an exceptional fertility of ideas.

only a bit more than a century of life, and the Curry-Howard approach, the one that we follow, born only around the half of the last century.
Maybe because of its young age, maybe not, a typical feature of proof theory is the fact that (at least until nowadays) there is no canonical mathematical object to be considered: every representation of proofs behaves well for certain properties and bad for others. However, the "Curry-Howard correspondence" identifies the study of proofs with the study of programs, and indicates at least what are some good properties that - following this approach - one should hope for.
It is interesting from a methodological point of view to note that often the difficulties are, before being technical, conceptual. Indeed, it is a peculiar characteristic of logic in general, to live at the very intersection between mathematics, computer science and philosophy, which makes the concept particularly important, a central point being the right balance between this two ingredients[4] - technique and concept - more than in other branches of mathematics.

The goal of the present work is not to present one particular result, nor to be exhaustive in any extent. But rather, it is to present some (standard for the first half, modern for the second half) techniques and ideas.
The main chapters of this thesis are the second and the third one, which are consecrated respectively to the (basic) study of the fundamental Curry-Howard correspondence, and (an introduction) to the realizability theory of J.-L. Krivine. The main subject that will constantly escort us is the fascinating (and maybe astonishing?) duality proofs/programs, and the first and fourth chapters are there to give the ideas of the situation in logic respectively before and after this breakthrough, and are important to make it possible for the reader to understand the framework in which all our journey takes place. Another lecture of the journey will be possible at the end of the fourth chapter.
More specifically:
in chapter 1 we introduce the traditional notions of the first order logic, i.e. that of logical system (definitions 1.1, 1.2 and 1.3) given by a formal language in which study some particular words, a structure to interpret them, and formal proofs to prove them. We will discuss the typical fundamental theorems which describe the first order theory (theorems 1.4, 1.5 and 1.6) and discuss some conceptual issues, which will motivate us to change the level of our analysis of proofs, switching to chapter 2.
There, we will approach two - a priori - different questions: firstly we introduce the fundamental tool that is the prototypical functional programming language "λ-calculus" (which has a rich mathematical structure), its first main properties (section 2.1), and (in section 2.2) we show how programming in λ-calculus sounds like; secondly, we introduce (in section 2.3) the "natural deduction" for intuitionistic logic (no reasoning by contradiction) and, by proving its main properties (theorem 2.37 and 2.41) we justify the interest for such a system. Always in chapter 2, in section 2.4 we show how these two objects - intuitionistic logic and functional programming - are in fact just two faces of the same profound phenomenon, which is precisely the main object of interest from the "Curry-Howard perspective", the point of view assumed in this thesis. The last section of chapter 2, is dedicated to mentioning some important areas of research in this topics.
Finally arriving in chapter 3, we radically change the methods and technical issues, but still remaining under the perspective of the duality proofs/programs: it is there developed the brilliant "classical realizability theory" of the mathematician J.L. Krivine, a powerful technique (born in the years 2000) that allows to bring our fundamental duality even further in two different

[4] This is another common point with theoretical physics.

and great steps: extending it to a classic framework and extending it, other than for proofs, for axioms too. In section 3.1 we briefly show how to extend it to the classical case (so not any more restricting to the - too strict for a mathematician - framework of intuitionistic logic), recovering the reasoning by contradiction bringing in an interesting computational counterpart; then, in section 3.2 we will generalize all of that in an abstract framework (the so called "realizability algebras") and from section 3.2 show how it can be developed (in a second order framework) following some ideas coming from the set theoretic theory of "forcing" (which earned the Fields medal to its creator P. Cohen). Finally, we develop the technical specific tools in order to associate a (meaningful) program to an axiom, that of the existence of a non trivial ultrafiler on $\mathbb{N}$: it is an example of the power of Krivine's theory, which allows one to retrieve computational content from axioms. It is an example of how Krivine's classical realizability allows to think to "all" mathematics in terms of some (sophisticated) exercise of programming.
And finally, in the last brief chapter 4, we just give the idea of how, supporting the point of view of the mathematician J.-Y. Girard, all our work was in fact a series of steps of the descent in an interesting and completely innovative way of looking at logic (and mathematics). A descent that can be pushed even further, this time going over the duality proofs-axioms/programs itself.

Just a last administrative comment: the chapter three of this thesis has been written (then readjusted in order to suite in this work) in Marseille, France, last year, as (the original version of it) was the "mémoire" for the French graduation of the double diploma the author followed. With the author's supervisor in Marseille, L. Regnier, it has been chosen to work on the subject of the third chapter, following [Kriv08] in order to produce a clearer version of it.

Preliminaries and notations

- As usual, we will use the standard symbol ":=" to mean that the thing on the left side is defined to be the thing on the right side, and the symbol "=:" to mean that the thing on the right side is defined to be the thing on the left side.
- Natural numbers $\mathbb{N}$ start from zero: $\mathbb{N} := \{0, 1, 2, 3, \dots\}$.
- As usual, $\mathscr{P}(X)$ denotes the power set of a set X.
- As usual, X^Y denotes the set of all the (total) functions from the set Y to the set X.
- We will essentially deal with words on alphabets. Let's recall the idea:
  let $\mathfrak{L}$ be a countable set. A *word* on $\mathfrak{L}$ is just a finite sequence of elements of $\mathfrak{L}$, plus the *empty word*, usually denoted as ϵ, which is the only one with length 0. A word $(w_1, \dots, w_n)$ is simply denoted with $w_1 \dots w_n$. The set of all the words on $\mathfrak{L}$ is thus $\mathfrak{L}^* := \bigcup_{n\in\mathbb{N}} \mathfrak{L}^n$ (with $\mathfrak{L}^0 := \{\epsilon\}$). We call $\mathfrak{L}$ an *alphabet* and its elements *characters*. The set $\mathfrak{L}^*$ is a monoid w.r.t. *concatenation* (i.e. the map $\cdot : \mathfrak{L}^* \times \mathfrak{L}^* \to \mathfrak{L}^*$, $w_1 \dots w_n \cdot w'_1 \dots w'_{n'} := w_1 \dots w_n w'_1 \dots w'_{n'}$), and has ϵ as neutral element. It is called the *free monoid on* $\mathfrak{L}$.
- We will constantly use *inductive constructions*. It is worth it to recall the idea and set a notation:
  let X be a set. Fix also a non-void subset $X_0 \subseteq X$ and a collection $\mathfrak{F}$ of functions f, each one with its arity k_f, from $\mathrm{Dom}(f) \subseteq X^{k_f}$ to X. We are interested in canonically consider the smallest (w.r.t. inclusion) subset $A \subseteq X$ s.t. $A \supseteq X_0$ and A is closed for $\mathfrak{F}$

(i.e. $f(x_1, \ldots, x_{k_f}) \in A$, $\forall x_i \in A$ s.t. $(x_1, \ldots, x_{k_f}) \in \mathrm{Dom}(f)$).
It is $A := \bigcap\{B \subseteq X \text{ s.t. } X_0 \subseteq B \text{ and } B \text{ closed for } \mathfrak{F}\}$.
Every element a of such an A is of the form $a \in X_0$ or $a = f(\vec{x})$, for some $x_1, \ldots, x_{k_f} \in X$, and every element of the form $f(\vec{x})$ is obtained beginning with some elements of X_0 by repeatedly applying only a *finite* number of times some $f \in \mathfrak{F}$ on them.
So we can *prove by induction* in A (also called "structural induction"): in order to prove that all the elements of A satisfy a certain property P, it is enough to prove that $\forall x_0 \in X_0$, x_0 satisfies P and that P is preserved by $\mathfrak{F}$, i.e. for all $f \in \mathfrak{F}$, if $x_1, \ldots, x_{k_f} \in X$ s.t. $\vec{x} \in \mathrm{Dom}(f)$ satisfy P then $f(\vec{x})$ satisfies P.
Our typical situation will be to have a $X_0 \subseteq X = \mathfrak{L}^*$ for some alphabet $\mathfrak{L}$, and a finite number m of maps f_i of arities k_i from $(\mathfrak{L}^*)^{k_i}$ to $\mathfrak{L}^*$. In this case in order to define the above set A we will use the standard evocative notation which consist in saying that:
"the elements a of A are inductively defined by $a ::= x \mid f_1(a, \ldots, a) \mid \ldots \mid f_m(a, \ldots, a)$, with $x \in X_0$".
In many cases we won't even precise the underling alphabet $\mathfrak{L}$, because inessential or clear from the context.

- Some of the most important objects that we will deal with (in particular λ-terms and proofs) are - or can be seen as - graphs. However, at our level we will never have the necessity to call in some rigorous graph theory and all we need to know is that a graph is just a set of nodes linked by edges. In particular, we'll always deal with connected and directed graphs, so that edges are arrows and for us the expressions "graph" and "connected directed graph" are synonyms. The exiting edges from a node are called children of the node and the entering ones are called parents, and a root is a node which has no parents.

1 The traditional Trinity: Semantics/Syntax/Meta

In this section we will only give some ideas without going into details, since we will overcome this point of view, and we will content ourselves more with intuitions rather than precise definitions.

We start assuming the position of a beginning of the XX^{th} century mathematician. Set theory has always been considered to be one of the most significant foundational grounds for "all" mathematics, and contradictions in set theory produced a literal crisis of the foundations. Of course, the crisis is interesting and one cares about it only from a philosophical and mathematically fine point of view: the rest (included the vast majority of set theory itself) is not affected by contradictions, the different branches develop and applications in physics are astonishing. It is however clearly a sort of defeat, for the philosopher as well as for the working mathematician, to found all that on a contradictory ground, and everyone should at least recognize the issue and the interest. This is not an historical nor a set theory manual, so we skip all historical considerations and don't even say how this history ended (indeed, has it?).
But we will see how it has been faced (has this approach worked?): briefly, the idea is to mathematically settle mathematics, in order to be extremely precise and avoid the use of natural languages with all their ambiguity.
It is here that some fundamental notions emerge: that of formal logical language, whose interesting words are called formulas, that of interpretation of a formula, and that of formal proof (which we will in the following simply call "proof", being the meaning always clear from the context). Those objects turn out to be interesting mathematical objects (more or less complicated). In the next chapters we will clearly be interested in the mathematical content of formal proofs, which turns out to be linked with something a priori completely unrelated.

Let's start from the very beginning, considering as an example a basic statement of arithmetic: $\forall n \in \mathbb{N}, \exists m \in \mathbb{N}$ s.t. $n < m$. We clearly see that we can isolate two different entities in this statement: there is a formula and a structure, and we are saying that such formula is true, or false, in that structure. In that case, "formula $= \forall n, \exists m \ s.t. \ n < m$", "structure $= \mathbb{N}$" and it is true. Now, we see that if we had referred the same formula to another structure rather than $\mathbb{N}$, the statement could have been false, for example if $N := \{1, 2, 3, 4\}$ then the statement $\forall n \in N, \exists m \in N$ s.t. $n < m$ is false. We can push that view a little further: let's say that by "$<$" we didn't mean the usual strict order on $\mathbb{N}$ but another relation; then the same statement could become false again.
Every (true or false) mathematical statement can be decomposed in the syntactic part, the formula, and the semantic part, the structure in which to interpret the formula. Now if we want to make a mathematical theory out of that, which is what A. Tarski did in the first half of the XX^{th} century, we need to be more rigorous and we end up giving the following two definitions:

Definition 1.1. *[Syntax] Let* $V_1 = \{x, y, z, \dots\}$ *be a countable set of symbols, whose elements are called* first order variables, $V_2 := \{X, Y, Z, \dots\}$ *a countable set of symbols, whose elements are called* second order variables, $C := \{c_1, c_2, c_3, \dots\}$ *a set of symbols, whose elements are called* constants, $F := \{f_1, f_2, f_3, \dots\}$ *a set of symbols, whose elements are called* function symbols *and finally* $\text{arity} : V_2 \cup F \to \mathbb{N}$ *a function (we write sometimes* X^n *to say that* $\text{arity}(X) = n$*).*
An expression[5] e *is a word defined inductively as:* $e ::= c \mid x \mid f(\overrightarrow{e})$ *(with* $c \in C$, $x \in V_1$, $f \in F$

[5] Usually the word "term" is employed. But since we will constantly talk about "λ-terms" in the future, calling them simply "terms", we prefer to use now this a little unusual terminology.

and of course in $f(\overrightarrow{e})$ there are exactly $\mathrm{arity}(f)$ expressions between the parentheses).
A formula *A is a word defined inductively as: $A ::= X(\overrightarrow{e}) \mid A \wedge A \mid A \vee A \mid \forall x A \mid \exists x A \mid \neg A$ (with $X \in V_2$ and $\overrightarrow{e}$ is a sequence of $\mathrm{arity}(X)$ expressions).*
We say that a formula is closed *when every one of its first order variables is quantified.*
One uses the so called logical connectives*, which are the symbols $\neg, \wedge, \vee, \forall, \exists$, instead of the words "not[6] ", "and", "or", "for all[7] ", "it exists", just to emphasize the fact that we are in a formal contest and to make things clearer.*

It's clear that every mathematical statement gives us a formula (in the sense of the above definition). For example, a statement of the form "A $\Rightarrow$ B" would give a formula of the form $A \to B := \neg A \vee B$, and our theorem "for every natural number it exists a bigger natural number" gives a formula $\forall x \exists y X(x, y)$.
Now let's formalize the idea of interpretation in a structure:

Definition 1.2. *[Semantics] A first order structure is a couple (M, φ), where:*
M is a non-empty *set,*
φ is a function (we don't specify its domain and codomain because it's tedious, but it's clear from the definition), called "interpretation", s.t. for every c constant, f function symbol, X^k second order variable, $\varphi(c) \in M$, $\varphi(f) : M^h \to M$ (with $h = \mathrm{arity}(f)$), $\varphi(X) \subseteq M^k$ (if $k = 0$ we set $\varphi(X) \in \{0, 1\}$) and $\varphi(\neg X) = M^k \setminus \varphi(X)$ (if $k = 0$ we require $\varphi(\neg X) \neq \varphi(X)$).
We extend φ on the expressions setting: $\varphi(f(\overrightarrow{e})) := \varphi(f)(\overrightarrow{\varphi(e)})$ (here we are not precise, but the idea is clear).
We now define the relation $\mathfrak{M} \models A$ of satisfiability of a formula A in a structure $\mathfrak{M} = (M, \varphi)$ by induction on A:
$\mathfrak{M} \models X(\overrightarrow{e})$ iff $\overrightarrow{\varphi(e)} \in \varphi(X)$ (if $\mathrm{arity}(X) = 0$ we require $\varphi(X) = 1$)
$\mathfrak{M} \models \neg A$ iff $\mathfrak{M} \not\models A$
$\mathfrak{M} \models A \wedge B$ iff $\mathfrak{M} \models A$ and $\mathfrak{M} \models B$ (and similarly for $\vee$ we use "or")
$\mathfrak{M} \models \forall x A$ iff for all $a \in M$, $\mathfrak{M} \models A[a/x]$ (and similarly for $\exists$ we use "it exists").
With the writing $A[a/x]$ we mean the formula A in which we interpret all its free (i.e. not quantified) occurrences of x by the element a.
We say that the truth value of A in $\mathfrak{M}$ is 1 when $\mathfrak{M} \models A$, 0 otherwise.
A theory T is a set of closed formulas. We say that a formula A is a logical consequence of a theory T, and we write $T \models A$, iff for all structures $\mathfrak{M}$ one has: $\mathfrak{M} \models T \Rightarrow \mathfrak{M} \models A$ (where a structure satisfies a theory when it satisfies all its formulas). We also call a "model of T" any structure that satisfies T.

What we have done in these definitions has been to write a formal language while keeping in mind the intended meaning behind it, and definition 1.2 is just writing down that meaning. The language refers to the usual meaning of "not", "and", "for all", etc, and we are just writing mathematics in a formal and non ambiguous way, instead of the natural language usually used, paying particular attention to what and how we can write things and where and when they are true.

[6] It turns out that it would be better to consider the negation not as a logical connective but directly as a relation on second order variables, but this is inessential for our quick journey in this subject. Indeed, one could also define negation also in other ways, as we shall see later on.

[7] Note that we have allowed this quantification only on first order variables, and not also on second order ones. We will quantify on second order variables in the third chapter of the thesis.

The attention to the logical form of the language one uses, allows to generate a very interesting and developed theory, which is called *model theory*. It studies the notion of models of theories, i.e. it studies the general notion of being a true (or false) formula in a certain mathematical structure.
Just to give an example, a field would be seen as a model of the theory obtained by writing in a formal first order language the usual field's axioms, and to study the logical consequences of this theory amounts to study the properties which hold in any field. Model theory is indeed very linked to algebra and it finds there its more remarkable applications. Just to give a basic simple example, one can easily prove that if a *first order* property[8] P holds in all fields with characteristic 0, then $\exists k \in \mathbb{N}$ s.t. P holds in any field $\mathbb{F}$ s.t. $\text{Char}(\mathbb{F}) \geqslant k$.
See [AbrTdF12] for a brief introduction to model theory.
That said, it's clear that from a *foundational* point of view this approach is quite empty.
In fact, we are simply saying that a formula A holds when its translation in natural language does, i.e. we are referring, with a pleonasm, syntax to its "meta", which is semantics and which is exactly the same as syntax but in the meta-world of structures.
But there's also another capital lack in what we have done: we have never even mentioned the word "proof"! To say it with the words of the french mathematician J.-Y. Girard:

> *Realism says that truth makes sense independently of the way we access it. [...] However, common sense should tell us that nobody has ever seen the class of all integers, not to speak of this ≪ Book of Truth and Falsehood ≫ supposed kept by Tarski. All these abstractions, truth, standard integers, are handled through proofs [...] The compulsory access to ≪ foundations ≫ is through a bleak Trinity Semantics/Syntax/Meta: this approach owns the copyright: it you don't accept this, you are not interested in foundations, period. [...] Clearly, if this is the only approach to foundations, one should do something else... by the way is what the main stream of mathematics realised long ago.*
> J.-Y. Girard,
> "From foundations to ludics", Bulletin of Symbolic Logic, 2003

If we analyse (also in an etymological sense: if we decompose) what we do in a proof, we see that one does a lot of small steps in which one deduces something from what already has (or by an axiom of the form "A or not-A") and the deductions are made according to some very basic rules. One is naturally led to give the following definition, which is well known in proofs-theory (and we'll soon realize that it's not the approach we're looking for).

Definition 1.3. *[Sequent calculus system LK for classical logic] A multiset μ on a set X is a function $\mu : X \to \mathbb{N}$. A multiset μ is said to be finite iff the set $\{x \in X \text{ s.t. } \mu(x) \neq 0\}$, called its support, is finite. It is consuetude to indicate a finite multiset μ with support $\{a_1, \dots, a_n\}$ by the writing $[a_1, \dots, a_1, \dots, a_n, \dots a_n]$ (where a_i occours $\mu(i)$ times).*
We call (first order) sequent *a finite multiset on the set of (first order) formulas. It is consuetude to indicate with the writing $\vdash \bot$ the empty sequent (i.e. the multiset constantly equal to 0) and a sequent $[A_1, \dots, A_1, \dots, A_n, \dots A_n]$ with the writing $\vdash \Gamma$, where Γ is any permutation of the list $A_1, \dots, A_1, \dots, A_n, \dots A_n$.*
We define the proofs of sequents and say that a proof of a formula A is a proof of the sequent $\vdash A$.

[8] A first order property is a property that can be expressed as a first order formula, i.e. a formula in some formal language as defined in 1.1

A proof *of a sequent* $\vdash \Gamma$ *is a directed tree*[9] *with nodes in the set of symbols* $\{\mathrm{ax}, \mathrm{cut}, \mathrm{W}, \mathrm{C}, \otimes, \parr, \oplus_1, \oplus_2, \&, \forall, \exists\}$ *and edges that satisfy the following conditions:*
- the ax*-nodes have exactly 0 premises and 1 conclusion*
- the cut*-nodes, the* $\otimes$*-nodes and the* $\&$*-nodes have exactly 2 premises and 1 conclusion*
- the W*-nodes (which is the abbreviation for* weakening*), the* C*-nodes (which is the abbreviation for* contraction*), the* $\parr$*-nodes, the* $\oplus_i$*-nodes, the* $\forall$*-nodes and the* $\exists$*-nodes have exactly 1 premise and 1 conclusion*
- there is exactly 1 node having exactly 1 premise and 0 conclusions (so it is the root of the tree)
- every edge is labeled with a sequent according to the following "deduction rules" associated to each type of node, which read:
the sequents that are above the horizontal line are the labels of the premises of the node, and the sequent under the line is the label of the conclusion of the node. The deduction rules are the followings:

$$\frac{}{\vdash A, \neg A}\mathrm{ax} \qquad \frac{\vdash \Gamma, A \qquad \vdash \neg A, \Delta}{\vdash \Gamma, \Delta}\mathrm{cut}$$

$$\frac{\vdash \Gamma}{\vdash \Gamma, A}\mathrm{W} \qquad \frac{\vdash \Gamma, A, A}{\vdash \Gamma, A}\mathrm{C}$$

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B}\otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B}\parr$$

$$\frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_1 \vee A_2}\oplus_i \qquad \frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B}\&$$

$$\frac{\vdash \Gamma, A\{x\}}{\vdash \Gamma, \forall x A}\forall \text{ (with } x \text{ not free in } \Gamma) \qquad \frac{\vdash \Gamma, A[e/x]}{\vdash \Gamma, \exists x A}\exists \text{ (with } e \text{ expression)}$$

Given a theory T*, we say that a formula* A *is provable from* T*, and we write* $T \vdash A$*, iff there exist a proof of* A *in which some ax-nodes are eventually not of the form* $\vdash A, \neg A$ *but of the form* $\vdash B$*, for some* $B \in T$*.*

The definition is a little uncomfortable to write but it's clear what we have in mind: a proof is a tree obtained by linking together the deduction rules in a "correct way". In the traditional approach a deduction rule should be read as: if in the sequents above the line there is at least one true formula, then there is at least one also in the sequent under the line.
For example a proof of the *drunk man*[10] formula $\exists x(D\{x\} \to \forall y D\{y\})$ would look like this:

[9] A tree is a connected acyclic graph. A directed tree is a connected acyclic directed graph. We can hence speak of the conclusions (exiting edges) and the premises (entering edges) of a node. And more, we'll need to keep trace of right and left parents, so we also require an order "left/right" on the premises on a node.

[10] Indeed, take as structure the set of people in a non empty bar and interpret $D\{x\}$ as "x is drunk". Now you know that in a non empty bar there always is a person that is such that if he is drunk then everybody is drunk too...

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg D\{y\}, \forall y D\{y\}), \neg D\{t\}, D\{y\}}\text{ax}}{\vdash \neg D\{y\} \vee \forall y D\{y\}), \neg D\{t\}, D\{y\}}\parr}{\vdash \exists x(\neg D\{x\} \vee \forall y D\{y\}), \neg D\{t\}, D\{y\}}\exists}{\vdash \exists x(\neg D\{x\} \vee \forall y D\{y\}), \neg D\{t\}, \forall y D\{y\}}\forall}{\vdash \exists x(\neg D\{x\} \vee \forall y D\{y\}), \neg D\{t\} \vee \forall y D\{y\}}\parr}{\vdash \exists x(\neg D\{x\} \vee \forall y D\{y\}), \exists x(\neg D\{x\} \vee \forall y D\{y\})}\exists}{\vdash \exists x(\neg D\{x\} \vee \forall y D\{y\})}\text{C}$$

Why taking exactly those rules? The traditional justification refers to semantic, verifying that the rules all preserve truth[11], in the sense that we already mentioned. Intuitively, that is obvious: now, since our definition 1.2 of semantics is just a copy of the formal language in the meta-language, that should work fine. And in fact we have the following theorem:

Theorem 1.4. *[Soundness] Let A be a formula and T a theory.*
Then: $T \vdash A \Rightarrow T \models A$

Proof. The proof follows the intuition, with only a technical adjustment, and it's quite simple. In any case, it is inessential in this thesis, so see for example [AbrTdf12]. □

Now that we are convinced that those rules allow us to really prove something (in the sense to establish that a certain formula is true), we can ask ourselves another questions: are they enough to prove everything that is true, or maybe we need some other rules?
The answer is that they are enough and it is given by the following theorem, which states the inverse implication of the previous one:

Theorem 1.5. *[Completness, Gödel (30's)] Let A be a formula and T a theory.*
Then: $T \models A \Rightarrow T \vdash A$

Proof. The proof is not trivial at all. There are several proofs, and in the case of countable languages one can give a very interesting proof (which has a modern view, and it is not the proof given by Gödel). But again, it would led us elsewhere to report and discuss it in this thesis, so for example see again [AbrTdF12]. □

There is another natural question that we can ask ourselves: we've just seen that the rules are sufficient to prove all true formulas; are they also all necessary to do that, i.e. is there some rule that we can eliminate?
The answer is positive, there are several rules that we can eliminate:
for example the reader can amuse himself to find a proof of the conclusion of a $\parr$-rule and of a

[11] The constraint $M \neq \varnothing$ in the definition of first order structure is purely ad hoc. If we drop that constraint, the definition still makes sense, and the only languages which admit the first-order structure with $M = \varnothing$ are "propositional" languages, i.e. with only 0-ary second order variables (i.e. "propositions"). In such a structure the truth tables of the non-quantified formulas are given by the usual truth tables and the truth tables for quantified formulas are: $\forall xA$ is true for every A, and $\exists xA$ is false for every A. Let's notice that the void structure is the only one in which the formula $\forall xA \to \exists xA$ is false, for any A (if $M \neq \varnothing$ then the formula is always true). Now, it is easy done to verify that $\vdash \forall xA \to \exists xA$. So we prove a formula which can be false: something not really nice. And so we would have to add the clause "$M \neq \varnothing$" to the hypothesis of the following theorem. Even if it can be the sign that this setting is in a sense unsatisfying, we will not worry about it and keep this constraint as it is habit to do.

$\otimes$-rule from their premises using only the rules C, W, $\oplus$ and &,
and also to prove the conclusion of a &-rule and of a $\oplus$-rule from their premises using only the rules C, W, $\parr$ and $\otimes$.
These easy exercises are far from being "just some fun facts". Indeed they accentuate the role of the C and W-rules, which are called *structural rules*, and whose role is essential in both logic and computer science, as we will mention later on.
There's also another rule that can be eliminated, and the proof of this result is a cornerstone in proof theory, which actually started the modern vision of the discipline:

Theorem 1.6. *[Cut-eliminability, Gentzen (30's)] If the language is countable we have: if a sequent is provable, then it is provable also by a cut-free proof.*

Proof. The original proof is not trivial at all. It goes by itself that we will not report it here, for the usual reasons. Anyway we will prove it in a different framework in the next section. But it must be said that this theorem, the *eliminability* of the cut-rule, is a trivial corollary of the same powerful method that produces a proof of the completeness theorem in the case of countable languages which we mentioned before.But what is interesting is the original proof given by G. Gentzen, that in fact proves a much more stronger result: it explicitly gives a procedure to eliminate the cut-rules in a proof, called the *cut-elimination*, transforming a proof in a cut free proof.
At a first reading one will probably not get the depth of such a result (we don't have yet the tools to understand its depth), but we will have all the time to go through it in the following of the thesis. □

While cut-elimination was in the 60's discovered to be in fact essentially the core of (functional) programming (as we shall see in the following chapter), and while in the 80's J.-Y. Girard elevated it to be "the" essential feature of all logic and discovered that this elevation has enormous consequences, the original motivation of Gentzen in the 30's was to produce the so-called "consistency proofs", in the spirit of the Hilbertian program. The following theorem clarifies it:

Corollary 1.7 (Consistency of the predicate calculus)**.** *Our deductive system (i.e. the set of our deduction rules) is non-contradictory*[12]*, i.e.* $\nvdash \bot$.

Proof. If there is a proof of $\vdash \bot$ then by cut-eliminability there is also a cut-free proof of $\vdash \bot$. Now let's consider the last rule used in that proof, that hence is not a cut. Since in all other rules but cut the conclusion has at least one formula, we have a contradiction. □

We have now a sound, complete and non-contradictory deductive system: the situation doesn't seem so bad! And in fact it's not... if we stick to the mere question of "provability", i.e. to say if for a given formula it does or it does not exist a proof of it. In fact, by soundness and completeness, to be provable equals to be true in all structures (i.e. to only have models). In a given structure, a closed formula is interpreted by a bit of information (1 or 0, it is either true or false), and a proof is just a "bureaucratic artefact" to discover (when possible) if it is 0 or 1, and we can't aboard the question of the *structure* of proofs. In other words, we are not considering

[12] To be contradictory means to prove both the formulas A and $\neg A$ (for some A), or equivalently to prove $A \wedge \neg A$, or equivalently to prove A for any formula A, or equivalently to prove $\bot$. It is an easy exercise to prove this equivalences.

the question "what is a proof" but only "what is provable", i.e. "what is true in every model", and again we fall in Tarskian semantic which is so good but not at all for our kind of pursuit.
How can we achieve our goal?
A first hint is given by an easy corollary of cut-eliminabilty, the so called *subformula property*:

Corollary 1.8. $\vdash A \Rightarrow \exists\, \pi$ *proof of* A *s.t. every formula that occurs in* π *is a subformula*[13] *of* A.

Proof. By simply looking at the rules of LK one sees that, except for the cut-rule, the sequents in the hypothesis of every one of them always contain only subformulas of the conclusion of the rule. (In the cut-rule we loose information: the formula that we cut is forever lost); now since via cut-eliminability every provable formula is cut-free provable, we are done. □

This property has some important consequences in the "automatic search of proofs": formulas and proofs are finite objects, so at least in a theoretical way one can then start generate all possible finite trees using only the finitely many subformulas of a formula A, and if A is provable then one will surely find a proof of it. Technically speaking, this is expressed saying that the notion of "provability" is "recursively enumerable[14]".
But more specifically, it says that everything we need in order to produce a proof of A, is "already there in the syntax of A and of the deduction rules", i.e. we are stating a sort of *internal completeness*, which has nothing to do with the external semantical one. It is a first manifestation of a radical change of perspective in the very approach to logic.
This is a central point in all the technical and philosophical work of J.-Y. Girard, from who we borrow again some (french) words (that we're traducing in English):

> *Rather than saying that logical rules preserve sacred principles (the truth) and are hence seconds, we shall, on the contrary, start from logical rules, without the seek of justifying them, and analyse their functioning.*
> J.-Y. Girard,
> "De la syllogistique à l'iconoclasme", manuscript 2007

and which can be resumed in his slogan: *from the rules of logic to the logic of rules.*
However, it is not exactly the aim of this thesis to go deep into this ambitious program, at the very intersection between mathematics, theoretical computer science and philosophy, producing an incredible amount of technical and conceptual innovations, and which we will only quickly mention later on. In this thesis we will rest on a little upper level, which is by the way the necessary step to do in order to have a deeper vision.
In particular, after the short discussion of the previous results, we begin to see that there is some lack in this traditional approach, from both a technical and a conceptual point of view. The development of the next chapters confirms this intuition, and this motivates the switch, in the next chapter, to a different perspective: that of the theoretical computer scientist.

As a final remark, it is perhaps worth to quickly specify that we didn't include even a word on the famous Gödel's incompleteness theorems: this is not because they are not related to the

[13] A subformula of a formula A is a subword of A which is also a formula. This set $\mathrm{subf}(A)$ of all the subformulas of A can be given an inductive characterization: $\mathrm{subf}(X(\vec{t})) = \{X(\vec{t})\}$, $\mathrm{subf}(A) = \mathrm{subf}(B) \cup \mathrm{subf}(C) \cup \{A\}$ if $A = B \wedge C$ or $A = B \vee C$, $\mathrm{subf}(\neg A) = \mathrm{subf}(A)$, $\mathrm{subf}(A) = (\bigcup_{e\ \mathrm{expr}} \mathrm{subf}(B[e/x])) \cup \{A\}$ if $A = \forall x B$ or $A = \exists x B$.

[14] Notion for which you can look in any recursion theory reference, for example [AbrTdf18]

content of the thesis, but it is more due to the fact that the following of this work is - strictly technically speaking only - independent from them. And since even only mentioning them would require to be extremely careful and bring in an all new series of notions and considerations, we opted for directly omitting them. Note that, however, a finer - both technical and conceptual - analysis than ours would require to bring them in. For example, we can read them as an indication that the XX^{th} century view has to be overcome.

2 The Curry-Howard correspondence

> *A theorem is not a decoration that one puts on a shelf, it is a tool, which can be used as a lemma to produce other theorems : the use of A through logical consequence is the actual meaning of A.*
> J.-Y. Girard,
> "Truth, modality and intersubjectivity", manuscript 2007

Let's abandon for a moment the question of proofs (in fact we are not abandoning it at all, but unconsciously partially answering it) and let's consider another interesting one: "what is an algorithm?", in the sense of "what is a computable function?"
We know the answer from the years 30's. By "knowing the answer" we mean that we have "a" model[15] that exactly captures the intuitive idea of "computable function", and of course the best we can hope is to empirically see that that model works, since the concept we are trying to modelize is intuitive[16]. In this case the fact that allows us to accept that the question is answered, is the fact that we found many models which clearly produce intuitively computable functions and that, even being radically different, are at the end equivalent. And this is instead a mathematical result (theorem 2.1) that one can prove.
The most important models are the following three (all introduced more or less in the 30's):
Turing-machines: introduced by Turing, are the abstract prototype of computers, in particular of *imperative programming*. We will not define them because we won't need them, but you can see any reference in computability theory, for example [Deh93]. A function is said to be Turing-computable iff it exist a Turing-Machine that computes it.
Recursion theory: developed mainly by Gödel and Kleene. We will not define them because we won't need them, but you can see for example [AbrTdF18] or any reference in computability theory, for example [Deh93]. One defines a set R of functions which are called recursive.
λ-calculus: introduced by A. Church. It's not an exaggeration to say that it is the leading protagonist of our thesis and an important actor in many domain of proof theory and theoretical computer science. We are about to define it and never leave it! It is a hugely rich theory by its own, for which a standard reference is [Bar84], or [Kriv93] for the kind of topics that we will treat the most. The notion of a representable function in λ-calculus is more delicate and we will mention it later.

Theorem 2.1. *Let $f : D \subseteq \mathbb{N} \to \mathbb{N}$. We have:*
f is Turing-computable $\Leftrightarrow$ f is recursive $\Leftrightarrow$ f is λ-representable.

Proof. It would led us completely elsewhere to prove it, since it would require to develop all the three theories. See any reference in computability theory and λ-calculus. □

This theorem is the technical result that supports the following:

[15] Of course here the word "model" has the usual meaning and has nothing to do with the models of which we spoke in the previous chapter.

[16] It's exactly the same situation as if we wanted to capture mathematically the intuitive notion of "drawing a line continuously". Also in this case we know the answer, in the sense that we have found a mathematical notion, that of continuity for real functions, that seems to work pretty well. Then, conceptually speaking, it is an *assumption* to say that the intuitive idea of continuity of a line is exactly that of the continuity of a real function, assumption that of course we can strongly accept thanks to evidence.

Church's Thesis:
the intuitive concept of computable function (from $D \subseteq \mathbb{N}$ to $\mathbb{N}$) is *exactly* the one of Turing-computable function (or equivalently, by the previous theorem, that of recursive function, or that of λ-representable functions).

One has probably noticed that we have always spoken about functions $f : D \subseteq \mathbb{N} \to \mathbb{N}$ and not directly from $\mathbb{N}$ to $\mathbb{N}$. Those functions are called *partial* functions, and computability theory discovers that in order to speak about the concept of computation it is peremptory to deal with partiality and not totality. This is essentially due to the necessity of non-terminating programs.

Now can finally go through λ-calculus.

2.1 Pure λ-calculus

When Church firstly defined it he actually had in mind to form a "general theory of functions", i.e. a theory in which everything is a function[17].
Let's consider the two following functions $f_1, f_2 : \mathbb{N}^4 \to \mathbb{N}$ defined setting, $\forall x, y, z, n \in \mathbb{N}$,

$$f_1(x, y, z, n) := 0 \text{ and } f_2(x, y, z, n) := \begin{cases} 0 & \text{if } xyzn = 0 \text{ or } n = 1 \text{ or } n = 2 \text{ or } x^n + y^n \neq z^n \\ 1 & \text{otherwise} \end{cases}$$

Thanks to A. Wiles we know that $f_1 = f_2$. However, it's clear that the way we defined them is radically different, independently from the truth of Fermat's last theorem.
In all that follows we are interested in the so-called *intentional* point of view, i.e. viewing a function as an algorithm, not in the *extensional* one, which identifies functions with their graph, and that is the mainstream point of view in mathematics (in fact it is set theory).
Now, there are two things that one does with a function: applying it to an argument (called application) and declaring its argument (called abstraction). And then of course there are the memory addresses (called variables) where one stores the information to compute with. This leads to the following definition:

Definition 2.2 (λ-calculus)**.** *Let's fix a countable set of symbols whose elements are called variables. A λ-term t is a word defined inductively as:*

$$t ::= x \mid \lambda x t \mid (t)t$$

where x is a variable and "λ" is just a character. We indicate the set of λ-terms with Λ.
A term that starts with λ is called an abstraction; a term that starts with a parenthesis is called an application (and of course we will say that $(t)u$ is the application of t to the argument u); a term that is an application of an abstraction is called a redex. So redexes are of the form $(\lambda x t)u$.
We define the set $\mathrm{FV}(t)$ of the free variables of a term t by induction:

- $\mathrm{FV}(t) := \{x\}$

[17] Like in set theory where everything is, instead, a set. Indeed, as in set theory one can encode "all" mathematics, in λ-calculus one can encode a great part of mathematics. For instance, we will encode some arithmetic:

- $\mathrm{FV}(\lambda x t) := \mathrm{FV}(t) \setminus \{x\}$
- $\mathrm{FV}((t)u) := \mathrm{FV}(t) \cup \mathrm{FV}(u)$

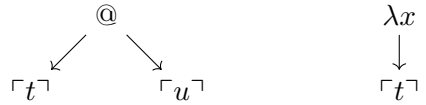
We say that a variable x is bound in t iff $x \notin \mathrm{FV}(t)$, and a term is closed if all its variables are bound. We will sometimes express the fact that the free variables of t are in the set $\{x_1, \dots, x_k\}$ by writing $t = t\{x_1, \dots, x_k\}$.
One can easily define an equivalence relation on Λ, called α-equivalence, which identifies the terms which differ only by the name of the bound variables. We will, as it is custom, consider terms up to renaming of bound variables[18]. That is to say, we work in the quotient set Λ/α. But it is a completely intuitive equivalence, so we will keep referring to Λ as the set of terms etc.

Notation 2.3 (Krivine's notation)**.** *We will sometimes write tu instead of $(t)u$, when this is not confusing. For example, we will often write xx instead of $(x)x$.*
We will also indicate with the writing $tu_1 \dots u_k$, or equivalently $(t)u_1 \dots u_k$, the term $((\dots((t)u_1)\dots)u_{k-1})u_k$.
For the seek of clarity, let's give an example:
the expression $(t_1)\dots(t_{k-1})t_k$ is the term obtained by applying t_{k-1} to t_k, obtaining $(t_{k-1})t_k$, then applying t_{k-2} on it, obtaining $(t_{k-2})(t_{k-1})t_k$, etc, till applying t_1 on the resulting term $(t_2)\dots(t_{k-1})t_k$. Conversely, the expressions $t_1 \dots t_k$ and $(t_1)t_2 \dots t_k$ are just syntactic sugar for the term $((((t_1)t_2)\dots)t_{k-1})t_k$, i.e. the term obtained applying t_1 on t_2, then applying it on t_3, etc, till applying resulting term on t_k.

Remark 2.4. *It would be more elegant and meaningful to think of a λ-term t as the graph $\ulcorner t \urcorner$ according to the following:*
- the nodes are in the set of symbols $\{@, x, \lambda x, y, \lambda y, \dots\}_{x,y,\dots \text{ variables}}$ and edges respecting the following constraints:

- *a λx-node has at most one parent and exactly one child;*
- *a $@$-node has at most one parent and exactly two children;*
- *there is exactly one root for in each graph.*

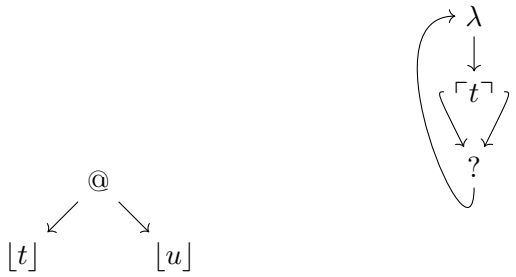
- the translation is inductively defined as:
$\ulcorner x \urcorner := x$ (i.e. the graph with only the node x and no edges), and $\ulcorner tu \urcorner$ and $\ulcorner \lambda x t \urcorner$ are respectively:



where in both cases the new edge (or edges) introduced, linking the new node λx (or $@$) to $\ulcorner t \urcorner$ (or $\ulcorner t \urcorner$ and $\ulcorner u \urcorner$), is (or are) entering in the root of $\ulcorner t \urcorner$ (or of $\ulcorner t \urcorner$ and of $\ulcorner u \urcorner$).

[18] The α-equivalence is nothing but the operation that one concurrently does when changing the name of the differential and the parameter of integration in an integral, when the name one wanted to use was already taken. It allows us to define the crucial notion of substitution, and it allows one to have a much clearer syntax: for example if we want to write the term that applies λxx to its argument, we can write $\lambda y(\lambda xx)y$ instead of the incomprehensible $\lambda x(\lambda xx)x$.

Actually, in order to handle α-equivalence the best thing to do is to add a new node[19]*$?$ to the set of nodes, replace the nodes of the form λx with a single node λ, and collect all the occurrences of bound variables x in a new $?$-node returning a single edge, which is then linked with the respective λ-node. More precisely, first we α-convert t in order to make all the bound variables different from each other, and then we translate the above graphs $\ulcorner t \urcorner$ as the following graphs $\lfloor t \rfloor$ (with all the usual constraints):*
$\lfloor x \rfloor := \ulcorner x \urcorner$ and $\lfloor tu \rfloor$ and $\lfloor \lambda x t \rfloor$ are respectively:



where in the second case the edges from $\ulcorner t \urcorner$ to $?$ exit from the x-nodes in t, for every occurrence of x in t which is bound in $\lambda x t$ (in the drawing we took the case of two occurrences).
However, for what we are concerned with in this thesis, this remark brings nothing and we can return thinking to λ-terms as words on an alphabet, modulo α-equivalence.

Example 2.5.
- *The identity function, which takes the argument and returns it back (here we don't care about domain or codomain issues) should be clearly implemented by the term $I := \lambda x x$ ($= \lambda y y = ...$, by α-equivalence)*
- *The constant function that whatever the input is, returns a fixed value x should be implemented by the term $\lambda y x$ (with $y \neq x$)*
- *The projections, i.e. the functions which takes two arguments and returns one of them, should be clearly implemented by the terms $T := \lambda x \lambda y x$ and $F := \lambda x \lambda y y$*
- *The function which takes an argument and applies it to itself (everything is a function: we can do it!) should be implemented by the term $\Delta := \lambda x (x) x$ ($= \lambda x x x$ with our notation)*
- $\Omega := \Delta\Delta = (\lambda x x x)\lambda x x x$ *(what would that function do?... We are about to see it.)*

We have our functions, but we didn't say how to compute them. Up to now Λ is just a static set of string of symbols. But the most important feature of a function, or a program, is to compute, to be executed.
Let's see how we evaluate for example the function $f(x) := x^3 + 1$ on $x = 2$:
we write $f(2)$, then $2^3 + 1$, then $8 + 1$, then 9.
In λ-calculus we would like to say that f is $\lambda x((+)(\text{cube})x)1$, and in order to evaluate $f(2)$ write:
$(\lambda x((+)(\text{cube})x)1)2$ then search the *free occurrences* of x in $((+)(\text{cube})x)1$ and *substitute* all of

[19] This notation comes from linear logic and in this framework "?" is an n-ary contraction. In fact this way of representing terms is a first step toward the crucial notion of *proof-net*, but we won't speak about it.

them with 2, obtaining ((+)(cube)2)1, and then do the same thing with + and cube, obtaining first (cube)2 + 1, then 8 + 1, and finally 9.
This lead to the following definition:

Definition 2.6 (β-reduction)**.** *Let x be a variable and t, u terms.*
We inductively define the term $t[u/x]$ as follows:

- $y[u/x] := \begin{cases} y & \text{if } y \neq x \\ u & \text{if } y = x \end{cases}$
- $\underbrace{\lambda y t'}[u/x] := \begin{cases} \lambda y \underbrace{t'[u/x]} & \text{if } y \neq x \\ \lambda y t' & \text{if } y = x \end{cases}$
- $\underbrace{t'u'}[u/x] := (t'[u/x])u'[u/x]$

It is the term obtained from t when firstly renaming all the bounded occurrences in t of all the variables in u such that in t there remain no more bounded variables that occur in u, and then substituting simultaneously all the free occurrences of x in t (with renamed variables) by u.
The β-reduction $\to_\beta \subseteq \Lambda^2$ is the binary relation defined by reflexive and transitive closure of the following relation $\to_\beta^1 \subseteq \Lambda^2$:

- $(\lambda x t)u \to_\beta^1 t[u/x]$
- $\lambda x t \to_\beta^1 \lambda x t'$, $\forall t' \in \Lambda$ *s.t.* $t \to_\beta^1 t'$
- $tu \to_\beta^1 t'u$, $\forall t' \in \Lambda$ *s.t.* $t \to_\beta^1 t'$, *and* $tu \to_\beta^1 tu'$, $\forall u' \in \Lambda$ *s.t.* $u \to_\beta^1 u'$.

We say that a term t is normal iff it can't be β^1-reduced, i.e. iff $\nexists t' \in \Lambda$ s.t. $t \to_\beta^1 t'$, i.e. iff t does not contain any redex.
We say that a term t is β-normalisable iff $\exists t' \in \Lambda$ s.t. $t \to_\beta t'$ and t' is normal. In this case t is called a normal form of t.

The intuition behind those definitions is clearly that β-reduction is the evaluation of the function, the execution[20] of the program, and a normal term is a result of some computation. Let's see if the definition sticks to the intuition we had in example 2.5:

- $It = (\lambda x x)t \to_\beta t$ so it really computes the identity. Let's just remark that obviously β-reduction is not symmetric: for example $Ix \to_\beta x$ but $x \not\to_\beta Ix$

[20]The kind of execution captured in β-reduction is what programmers call "call-by-name" execution: when evaluating a function $\lambda x t$ on the argument u we pass to the body t of the function all the argument u without looking at it (i.e. without reducing it, even if possible). Another typical kind of reduction is the "call-by-value" where, in the previous situation, we first evaluate u and then pass to t the result of the evaluation. Those two reductions (and other possible) have its own pros and cons in term of convenience (resources in space, time, etc), but from a theoretical point of view taking call-by-name is typically enough.

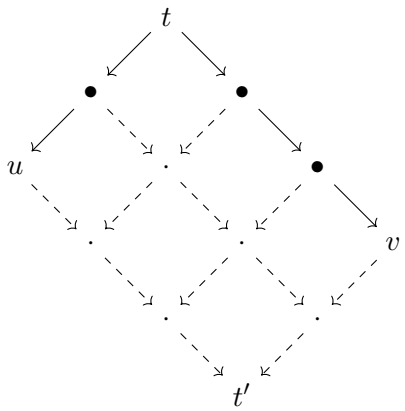
- $(\lambda yx)t \to_\beta x$ (for $y \neq x$) for all t (this is a program that erases the input)
- $Ttu = (\lambda x \lambda yx)tu \to_\beta (\lambda yt)u \to_\beta t$, and $Ftu = (\lambda x \lambda yy)tu \to_\beta (\lambda yy)u \to_\beta u$ so they compute the projections
- $\Delta t = (\lambda xxx)t \to_\beta tt$ (this is a program that duplicates the input)
- $\Omega = \Delta\Delta \to^1_\beta \Delta\Delta = \Omega$, and this is the only possible reduction. Thus Ω is not normalizable. It is the prototype of non-terminating programs.

We have just seen that not every term is normalizable.
There is another natural question that arises: if a term is normalizable, does it have a unique normal form?
Let's remark that, following the intuition of a normal form being the result of a computation, the answer should be positive.
The answer is indeed positive, and it's a trivial corollary of the following Church-Rosser theorem.

Definition 2.7. *Let $\to \subseteq R^2$ a binary relation on a set R.*
We say that $\to$ enjoys the diamond property iff $\forall t, u, v \in R$ s.t. $t \to u$ and $t \to v$, $\exists t' \in R$ s.t. $u \to t'$ and $v \to t'$.
We say that $\to$ is confluent (or that enjoys Church-Rosser property) iff the reflexive transitive closure[21] of $\to$ is locally confluent.

Proposition 2.8. *If $\to$ is locally confluent then $\to$ is confluent.*

Proof. Read the following diagram as: big points, t, u, v and the full lines are given, small points, t' and dashed lines are obtained by local confluence.



□

We will clearly look at the case $R = \Lambda$ and $\to = \to^1_\beta$ (note that the reflexive transitive closure of $\to^1_\beta$ is $\to_\beta$).

Remark 2.9. $\to^1_\beta$ *is* not *locally confluent:*
take $t = (\lambda dI)\Omega$; it reduces as $t \to^1_\beta I$ (reduce the redex t) and also $t \to^1_\beta t$ (reduce the redex Ω); obviously $t \neq I$ and I being normal, we don't have local confluence.

[21] The reflexive transitive closure of a relation $\to$ is the smallest relation $\to^*$ s.t. $\to^* \supseteq \to$ and that is reflexive and transitive.

But nevertheless, it is confluent:

Theorem 2.10 (Church-Rosser)**.** $\rightarrow_\beta^1$ *is confluent.*

Proof. Not trivial at all. We would need to introduce some new smart ideas to prove it, so see any reference of λ-calculus, for example [Kriv93]. □

Corollary 2.11. *If t is normalizable then it has a unique normal form.*

Proof. Suppose $t \rightarrow_\beta u, u'$ with u, u' normal. By confluence $\exists t'$ s.t. $u, u' \rightarrow_\beta t'$. But since u, u' are normal it must be (by definition of β-reduction) $u = t' = u'$. □

We have seen β-reduction, but there are several other really important notions of reduction of a term, which correspond to different strategies of executing a program. We will only quickly mention them:

Proposition 2.12. *All terms have a unique canonic writing:*
$\forall t \in \Lambda$, $\exists\,!\ k, n \in \mathbb{N}$, $\exists\,!\ x_1, \ldots, x_k$ *variables,* $\exists\,!\ R$ *variable* or *redex,* $\exists\,!\ t_1, \ldots, t_n \in \Lambda$
s.t. $t = \lambda x_1 \ldots \lambda x_k (R) t_1 \ldots t_n$.

Proof. Two easy inductions of t:
"existences":
- case $t = x$ and case $t = \lambda x t'$: trivial.
- case $t = uv$: then $u = \lambda z_1 \ldots \lambda z_l (R_1) t'_1 \ldots t'_m$. If $l = 0$, just take $k = 0$, $R = R_1$, $n = m + 1$ and $t_i = t'_i$, $i = 1, \ldots, m$, $t_n = v$; if $l \geq 1$ then uv is a redex and thus take $k = n = 0$ and $R = t$.

"uniqueness":
- case $t = x$: trivial.
- case $t = \lambda x t'$: let $\lambda x_1 \ldots \lambda x_k (R) u_1 \ldots u_n = t = \lambda y_1 \ldots \lambda y_{k'} (R') u'_1 \ldots u'_{n'}$. So $x_1 = x = y_1$ and so $\lambda x_2 \ldots \lambda x_k (R) u_1 \ldots u_n = t' = \lambda y_2 \ldots \lambda y_{k'} (R') u'_1 \ldots u'_{n'}$. By inductive hypothesis then $k = k'$, $x_i = y_i$, $i = 2, \ldots, k$, $R = R'$, $n = n'$, $u_i = u'_i$, $i = 1, \ldots, n$, and so we have also the uniqueness for the two writings of t.
- case $t = uu'$: the two writings of t must be of the form $(R) v_1 \ldots v_n = t = (R') v'_1 \ldots v'_{n'}$. So $(R) v_1 \ldots v_{n-1} = u = (R') v'_1 \ldots v'_{n'-1}$ and $v_n = u' = v'_{n'}$. By inductive hypothesis on u we have $R = R'$, $n - 1 = n' - 1$ and $v_i = v'_i$, $i = 1, \ldots, n-1$, so that we obtain also the uniqueness for the two writings of t. □

Corollary 2.13 (Characterization of β-normal forms)**.** *Let $t \in \Lambda$. Then:*
t is normal iff $t = \lambda x_1 \ldots \lambda x_k (x) t_1 \ldots t_n$, with t_i normal (and x can *be equal to an x_i).*

Proof. Trivial: let $t = \lambda x_1 \ldots \lambda x_k (R) t_1 \ldots t_n$ its canonic writing. Since t is normal R cannot be a redex, so it is a variable. And if some t_i is not normal then also t is not. □

Definition 2.14. *Let $t = \lambda x_1 \ldots \lambda x_k (R) t_1 \ldots t_n$ be the canonic writing of a term $t \in \Lambda$.*
In the case R redex we say that R is the head redex *of t;*
in the case R variable, we say that R is the head variable *of t, and in this case t is a said to be* head

normal.
We define the head reduction $\to_h$ *to be the relation which consist in* β*-reducing at each step the head redex, if the term is not head normal, and in doing nothing if we reach a head normal term.*
If $t \to_h t'$ *and* t' *is head normal, we say that* t' *is the head normal form of* t *(of course if it exists it is unique).*

Remark 2.15. *Of course* $\to_h \subseteq \to_\beta$.
And also, a head normal form is not, in general, a normal form. In fact it's a simple exercise to verify that:
$Y := (\lambda x \lambda f(f)((x)x)f)\lambda x \lambda f(f)((x)x)f \to_h \lambda f(f)(Y)f$ *which is head normal (with the notation of the previous definition, take* $k = 1 = n$, $x_1 = f$, $R = f$ *(variable) and* $t_1 = Yt$*). However this is not a normal term, since it contains the redex* Y.

Definition 2.16. *The left-reduction* $\to_l$ *is defined to be the relation which consist in reducing at each step the leftmost redex, if there is one, in doing nothing otherwise.*
Equivalently we can give a recursive definition: in order to left-reduce t *you:*
- head-reduce t *till you find a head normal form; call it* $\lambda x_1 \dots \lambda x_k (y) t_1 \dots t_n$
- left-reduce t_1
...
- left-reduce t_k.

Let's remark that $\to_h$ and $\to_l$ are deterministic strategies, i.e. at each step there is exactly one possible reduction to do (that could be do nothing). One will never have to choose which redex to reduce at a given step. So it's meaningful to say that for example left-reduction ends on a term.

The interest in left-reduction is given by the following theorem:

Theorem 2.17. *Let* t *be a term. We have:* t *is* β*-normalizable* $\Leftrightarrow$ *left-reduction on* t *ends.*
And since of course $\to_l$ *reduces all redexes, if it ends then the term that one finds is the* β*-normal form of* t.

Proof. We would need many more notions and results. See for example [Kriv93]. □

Remark 2.18. *The previous theorem says that if we* already know *that a term is normalizable, then the algorithm of left-reduction normalizes it and finds its normal form.*
So the problem of finding the normal form of a given normalizable term is decidable (i.e. algorithmically resolvable in a finite time).
However that doesn't mean that we have an algorithm that, given a term t*, tells us if* t *is normalizable or not.*
In fact, if a t *is not normalizable, running left-reduction on it will never end, and in a finite time we will never know if it's going to stop later or never. So the hypothesis of already knowing that* t *is normalizable is necessary.*
More precisely, since λ*-terms are equivalent to Turing-machines, having an algorithm that always decides if a term is normalizable or not is equivalent to decide the halting problem, proved to be indecidable by Turing in 1936.*

Another important notion that regards reduction is the following:

Definition 2.19. *A term t is said to be* strongly normalizable *iff all the possible β-reductions of t ends.*

For example $((\lambda y \lambda z y)x)(\lambda x x x)\lambda x x x = Tx\Omega \to_\beta x$ which is its normal form, if we operate 2 steps of head-reduction, but also $Tx\Omega \to_\beta Tx\Omega$ if we start by reducing the redex Ω. So if at each step we choose this reduction, we will never stop reducing.
Thus $Tx\Omega$ is normalizable but not strongly normalizable.

The last important notion that we mention is the following:

Definition 2.20. *The β-equivalence $\simeq_\beta \subseteq \Lambda^2$ is the equivalence relation that is the symmetric closure of $\to_\beta$.*

The intuition is that β-equivalence identifies the terms which have the same computational behaviour, as the following proposition clarifies.

Proposition 2.21. $t \simeq_\beta t' \Leftrightarrow \exists u \in \Lambda$ *s.t.* $t, t' \to_\beta u$.

Proof. $\Rightarrow$: obvious.

$\Leftarrow$: Clearly $t \simeq_\beta t' \Leftrightarrow \exists\ t_0 = t, t_1, \dots, t_{n-1}, t_n = t'$ s.t. $t_{2i} \to_\beta t_{2i+1} \leftarrow_\beta t_{2i+2}$, for $i = 0, \dots, \frac{n-2}{2}$ (and n is even). But then by confluence (applied on the terms with even index) $\exists u_1, \dots, u_{\frac{n-2}{2}} \in \Lambda$ s.t. $t_{i+1} \to_\beta u_{i+1} \leftarrow_\beta t_{i+3}$. We can then repeat[22] this argument, finally ending up with the existence of u.

□

Let's return on the term $Y = (\lambda x \lambda f(f)((x)x)f)\lambda x \lambda f(f)((x)x)f$ and recall that λ-terms are functions.

Definition 2.22. (Fixed points).
A term h is said to be a fixed point *of a term t iff $(t)h \simeq_\beta h$.*
A closed term F is said to be a fixed point combinator *iff $(t)(F)t \simeq_\beta (F)t$ for all terms t.*
Note that if F is a fixed point combinator, then Ft is a fixed point of t, for any term t.

Do fixed points exist? And if yes, does it exist a fixed point combinator, with which to generate a fixed point for any term? Recall that "usual" functions (not λ-terms) don't always admit fixed points.

Proposition 2.23. *The term Y is a fixed point combinator.*
So in particular, every λ-term t admits a fixed point, namely $(Y)t$.

[22] An induction would be a precise way of doing it. But a drawing would be a lot clearer.

Proof. For any term t we have[23]: $(Y)t \to_\beta (\lambda f(f)(Y)f)t \to_\beta (t)(Y)t$. Hence $(t)(Y)t \simeq_\beta (Y)t$. □

Let's just remark that however in general, $(t)(Y)t \not\twoheadrightarrow_\beta (Y)t$.

Remark 2.24. *Being a fixed point combinator is a remarkable property, and Y, which is called Church's fixed point combinator, is not the sole to have this behaviour:*
another fixed point combinator is (as one can easily verify) the Curry fixed point combinator:
$Y' := \lambda h(\lambda x(h)(x)x)\lambda x(h)(x)x$.
Let's just remark that while $(Y)t \to_\beta (t)(Y)t$, for Y' we have $(Y')t \not\twoheadrightarrow_\beta (t)(Y')t$,
but (thanks to the previous proposition) $(t)(Y')t \simeq_\beta (Y')t$ anyway since (as one can verify)
$(Y')t \to_\beta (t)(t)(\lambda x(t)(x)x)\lambda x(t)(x)x \leftarrow_\beta (t)(Y')t$.
A last example is (as one can easily verify) the Turing fixed point combinator:
$\Theta := (\lambda x\lambda y(y)(x)(x)y)\lambda x\lambda y(y)(x)(x)y$.

2.2 Some programming

It should be clear at this point that λ-calculus is nothing but a (low level functional[24]) programming language.
In this section we shall implement some basic fun programs, just to give the taste of it.

In order to be a good programming language we should find a good representation of at least some basic data types (booleans, couples, lists,...), implement the basic boolean operations (and, or, not), and maybe some arithmetic.
We will see that to *verify* that the proposed term does what we claim is usually not that hard: the real deal is finding it!

Booleans:
The standard way of representing booleans *true* and *false* in λ-calculus is the following:
we set `True`:$= \lambda x\lambda y x$ and `False`:$= \lambda x\lambda y y$.
This are exactly the projections T and F introduced in the previous section.

In order for them to be good booleans we should at least find the implementation `if` of the - "if then else" program:
input: three programs B, *then* and *else*;
output: *then* if $B \to_\beta$ `True`, *else* if $B \to_\beta$ `False`. Not important otherwise.
Here it is:
`if`:$= \lambda b\lambda t\lambda e(b)te$.
In fact one can easily verify that we have: (`if`) B *then else* $\to_\beta \begin{cases} then & \text{if } B \to_\beta \texttt{True} \\ else & \text{if } B \to_\beta \texttt{False} \end{cases}$

[23] Verifying it is just a simple computation. As usual, the real difficulty is finding it.
[24] Famous functional programming languages are: Lisp, OCaml, Haskell, etc. They all are completely based on λ-calculus.

We can easily program the boolean operations:

- "not" program:
input: a program B
output: $\texttt{True}$ if $B \to_\beta \texttt{False}$, $\texttt{False}$ if $B \to_\beta \texttt{True}$. Not important otherwise.
Here it is:
$\texttt{Not} := \lambda b(b)\texttt{False}\ \texttt{True}$.
In fact one can easily verify that: $(\texttt{Not})\ B \to_\beta \begin{cases} \texttt{True} & \text{if } B \to_\beta \texttt{False} \\ \texttt{False} & \text{if } B \to_\beta \texttt{True} \end{cases}$

- "and" program:
$\texttt{and} := \lambda b\lambda b'(b)b'\texttt{False}$ has the following behaviour:
$(\texttt{and})BB' \to_\beta \begin{cases} \texttt{True} & \text{if } B \to_\beta \texttt{True} \text{ and } B' \to_\beta \texttt{True} \\ \texttt{False} & \text{if } B \to_\beta \texttt{True} \text{ and } B' \to_\beta \texttt{False} \\ \texttt{False} & \text{if } B \to_\beta \texttt{False} \text{ and } B' \to_\beta \texttt{True} \\ \texttt{False} & \text{if } B \to_\beta \texttt{False} \text{ and } B' \to_\beta \texttt{False} \end{cases}$
For example, also the term $\lambda b\lambda b'((\texttt{if})b(\texttt{if})\ b'\ \texttt{True}\ \texttt{False})\texttt{False}$ has the same behaviour.

- "or" program:
$\texttt{or} := \lambda b\lambda b'((b)\texttt{True})(b')\texttt{True}\ \texttt{False}$ implements the "or":
$(\texttt{or})BB' \to_\beta \begin{cases} \texttt{True} & \text{if } B \to_\beta \texttt{True} \text{ and } B' \to_\beta \texttt{True} \\ \texttt{True} & \text{if } B \to_\beta \texttt{True} \text{ and } B' \to_\beta \texttt{False} \\ \texttt{True} & \text{if } B \to_\beta \texttt{False} \text{ and } B' \to_\beta \texttt{True} \\ \texttt{False} & \text{if } B \to_\beta \texttt{False} \text{ and } B' \to_\beta \texttt{False} \end{cases}$
Thanks to truth tables we can state that also the term $\lambda b\lambda b'(\texttt{Not})((\texttt{and})(\texttt{Not})b)(\texttt{Not})b'$ implements the "or".

Couples:
The standard representation for the couples is the following:
for $u, v \in \Lambda$, we set $< u, v > := \lambda f(f)uv$.

For this to be a good representation, we shall find the constructors of the couples and the projections on its elements:
- "couple constructor" program:
input: two programs u and v
output: the term $< u, v >$.
It is a trivial exercise to verify that if we set:
$\texttt{couple} := \lambda u\lambda v < u, v >$
then we have $(\texttt{couple})uv \to_\beta < u, v >$.

Let's find the projections on its left and right argument:
- "left projection" program:
input: $< u, v >$

output: u.
Here it is:
$\texttt{Lproj} := \lambda c(c)\texttt{True}$
It is a simple exercise to verify that $(\texttt{Lproj}) < u, v > \to_\beta u$.
Analogously we find the right projection term:
$\texttt{Rproj} := \lambda c(c)\texttt{False}$

We could now represent lists, trees, which would be a lot of fun... but it's not the point of this thesis.
Instead we will look at some arithmetic.

Integers:
The standard way of representing integers is by the so called "Church's numerals", defined below: for every $n \in \mathbb{N}$ we set the term $\mathtt{n} := \lambda f \lambda x (f)^n x$,
where with the notation $(f)^n$ we mean the word (it is not a λ-term by itself) $(f)\dots(f)$ (n times). Therefore for example $(f)^4 x$ is just syntactic sugar for the term $(f)(f)(f)(f)x$, and $(f)^0 x$ stands for the term x.
We see that the Church's numeral of n is what is called an "iterator", i.e. a program that takes a function and an argument as inputs and returns a program that iterates the function on that argument n times.
In other words, we are representing integers in base 1.
For example $\mathtt{0}$=$\texttt{False}$, $\mathtt{1} = \lambda f \lambda x (f)x$, $\mathtt{4} = \lambda f \lambda x (f)(f)(f)(f)x$, etc.

Definition 2.25. *A partial*[25] *function* $f : \mathbb{N}^k \to \mathbb{N}$ *is said to be* λ*-representable iff* $\exists \mathtt{f} \in \Lambda$ *s.t.* $\forall n =: (n_1, \dots, n_k) \in \mathbb{N}^k$, *we have:*
$$\begin{cases} (\mathtt{f})\mathtt{n}_1 \dots \mathtt{n}_k \to_\beta \mathtt{f(n)} & \text{if } f(n) \text{ is defined} \\ (\mathtt{f})\mathtt{n}_1 \dots \mathtt{n}_k \text{ is not normalizable} & \text{otherwise} \end{cases}$$

We know from the Church's thesis that all the intuitively computable functions are λ-representable.

The fundamental operation on integers is the successor. Let's program it:
- "successor" program:
input: a program p
output: the Church's numeral $\mathtt{n+1}$, if $p \to_\beta \mathtt{n}$, for some $n \in \mathbb{N}$. Not important otherwise.
Here it is:
$\texttt{succ} := \lambda n \lambda f \lambda x ((n)f)(f)x$.
Another way to implement it could be the term $\lambda n \lambda f \lambda x (f)(n) f x$.
The first way consist in iterating the function f n-times but not on x, but on $(f)x$, so to get at the end $n+1$ total iterations. The second way consists in applying f to the nth iteration of f on x, so to get at the end always the same result of $n+1$ iterations.
It's a simple computation the verification that in fact in both cases we have: $(\texttt{succ})\mathtt{n} \to_\beta \mathtt{n+1}$.

- "sum" program:

[25] We recall that a *partial function* from $\mathbb{N}^k$ to $\mathbb{N}$ is a function with domain $\mathrm{dom}(f) \subseteq \mathbb{N}^k$. We already mentioned that computability theory finds out that in order to deal with the notion of computation it is obligatory to deal with partial functions and not total functions.

we look for a term $\mathtt{sum} \in \Lambda$ s.t. $(\mathtt{sum})\ \mathtt{n}\ \mathtt{m} \to_\beta \mathtt{n+m}$.
We have to sum the number of iteration, and in order to do that we can apply the nth iteration of f to the mth iteration of f on x, which corresponds to writing $n + m = (m + 0) + n$.
We are thus led to:
$\mathtt{sum} := \lambda n\lambda m\lambda f\lambda x(n)f(m)fx$, and we can verify that it works.
But we can also follow the idea that $n + m = ((m + 1) + \dots) + 1$ and iterate n times directly the successor function on the mth iteration of f, which, as the reader can check, leads to the term: $\lambda n\lambda m((n)\mathtt{succ})m$, which also works.

- "multiplication" program:
we look for a term $\mathtt{mult} \in \Lambda$ s.t. $(\mathtt{mult})\ \mathtt{n}\ \mathtt{m} \to_\beta \mathtt{nm}$.
Arithmetic helps us find that $nm = (0+n)+n+\dots+n$ (m iterations of the function that sums n, starting from 0), so that one is led to the term:
$\mathtt{mult} := \lambda n\lambda m((m)(\mathtt{sum})n)\mathtt{0}$, which in fact works.

- "exponential" program:
we look for a term $\mathtt{exp} \in \Lambda$ s.t. $(\mathtt{exp})\ \mathtt{n}\ \mathtt{m} \to_\beta \mathtt{n}^{\mathtt{m}}$.
With the help of arithmetic ($n^m = (1 \cdot n) \cdot \dots \cdot n$ (m times)) we find:
$\mathtt{exp} := \lambda n\lambda m((m)(\mathtt{mult})n)\mathtt{1}$, which works.
There is also another very simple term that computes exponentiation:
it is a good (non trivial) exercise to convince oneself that also $\lambda n\lambda m(m)n$ works.

- "minimum" program: another really good (and non trivial) exercise would be to convince himself that the following term (called the Maurey's term) computes the minimum:
$\mathtt{min} := \lambda n\lambda m(((n)\varphi)\lambda dn)(m)\varphi\lambda dm$, where $\varphi = \lambda f\lambda g(g)f$, is such that:

$$(\mathtt{min})\ \mathtt{n}\ \mathtt{m} \to_\beta \begin{cases} \mathtt{n} & \text{if } n \leq m \\ \mathtt{m} & \text{otherwise} \end{cases}$$

Now that we have the basic arithmetical functions, we would like to do more.
Things start to become harder and more interesting, but since it is not the aim of this thesis to give an exhaustive course on functional programming we shall only mention some remarkable facts.

- "predecessor" program:
a typical excellent and not easy exercise is to convince himself (actually, it is a lot better to find it by himself[26]) that $\mathtt{prec}\ \mathtt{n} \to_\beta \mathtt{n\text{-}1}$ (as habit, we take $0 - 1 := 0$), where:
$\mathtt{pred} := \lambda n(\mathtt{Lproj})((n)S) < *, \mathtt{0} >$, where $*$ is any λ-term and $S := \lambda c((\mathtt{couple})(\mathtt{Rproj})c)(\mathtt{succ})(\mathtt{Rproj})c$.

- "subtraction" program:
We can simply iterate m times the predecessor on n to get that: $\mathtt{sub}\ \mathtt{n}\ \mathtt{m} \to_\beta \mathtt{n\text{-}m}$
(again, $n - m := 0$ if $m \geq n$, of course!), with $\mathtt{sub} = \lambda n\lambda m(m)\ \mathtt{prec}\ n$.

[26] Church believed for a while that it was impossible to find it, until one day when while having his wisdom teeth pulled, his student Kleene had the idea of the solution.

But there is another method that we can use. In fact, we don't have yet a method for handle inductive definitions. Let's develop it here for this particular case in order to find another representation of subtraction:

arithmetic helps us find a recursive definition of sub : $\mathbb{N}^2 \to \mathbb{N}$, $\text{sub}(n,m) := n - m$, which is:
$\text{sub}(n,m) = \begin{cases} n & \text{if } m = 0 \\ \text{pred}(\text{sub}(n, \text{pred}(m))) & \text{if } m \geq 1 \end{cases}$
where clearly pred is the predecessor function represented by $\texttt{pred}$.
Now let's consider a new function S', which has the same recursive definition of sub but with an additional parameter which is a function from $\mathbb{N}^2$ to $\mathbb{N}$ at the place of sub:
$S' : \mathbb{N}^{(\mathbb{N}^2)} \times \mathbb{N}^2 \to \mathbb{N}$, $S'(S,n,m) := \begin{cases} n & \text{if } m = 0 \\ \text{pred}(S(n, \text{pred}(m))) & \text{if } m \geq 1 \end{cases}$
Let's remark that by construction we have $S'(\text{sub}, n, m) = \text{sub}(n,m)$ (i.e. sub is a "fixed point" of S'...)
We want to "represent" the function S'.
It should be clear at this point that it's not a problem to find a term $\texttt{iszero} \in \Lambda$ with the property that $(\texttt{iszero})ntu \to_\beta \begin{cases} t & \text{if } n \to_\beta \texttt{0} \\ u & \text{otherwise.} \end{cases}$
For example we can take $\texttt{iszero} := \lambda n \lambda t \lambda u((n)(\lambda d \texttt{False})\texttt{True})tu$.
This allows us to represent functions defined by cases, and we can "represent" S' with it. The reader can check that we can write:
$\texttt{S'} := \lambda s \lambda n \lambda m(((\texttt{iszero})m)n)(\texttt{pred})(s)n(\texttt{pred})m =: \lambda s C_s$.
Now we note the crucial point that if a term h has the property that $h \to_\beta C_h$, then h would, by definition of sub and of C_h, represent sub.
Thus we only need to find one with this property, if there are.
But by definition of C_h we clearly have: $(\texttt{S'})h \to_\beta C_h$, so that if we had a term h s.t. $h \to_\beta (\texttt{S'})h$, we would have found it. But this is precisely the property of the fixed point combinator Y!
So if we put $\texttt{sub} := (Y)\texttt{S'}$ then we have: $\texttt{sub} \to_\beta (\texttt{S'})(Y)\texttt{S'} = (\texttt{S'})\texttt{sub} \to_\beta C_{\texttt{sub}}$, from where we get that $(Y)\texttt{S'} = \texttt{sub}$ represents sub.

The powerful method described above is in fact general, and by the same technique of adding a "functional" parameter we become able to program in λ-calculus a lot of functions[27], like the Fibonacci's suite, the euclidean division, even the Ackerman function hides no mystery with that method, etc. Just for the fun, let's see how to implement the euclidean division:

- "euclidean division" program:
That is, we want to implement the programs for the quotient and the reminder on integers. That means finding (if they exist) $\texttt{quotient}, \texttt{reminder} \in \Lambda$ s.t. $\forall m,n,q,r \in \mathbb{N}$ the following equivalence holds:

$$\begin{cases} (\texttt{quotient})\ \texttt{m}\ \texttt{n} \simeq_\beta \texttt{q} \\ (\texttt{reminder})\ \texttt{m}\ \texttt{n} \simeq_\beta \texttt{r} \end{cases} \Leftrightarrow \begin{cases} m = nq + r \\ r < n \end{cases} \Leftrightarrow \begin{cases} q = \min\{t \leq m \text{ s.t. } (t+1)n > m\} & (\dagger) \\ r = m - nq & (\ddagger) \end{cases}$$

[27] Indeed, we are now really close to become able to prove the theorem mentioned at the beginning of the chapter, in particular the representation in λ-calculus of the recursive functions. The implementation of recursion that we just saw, thanks to fixed point combinators, is one of the fundamental ingredients of the proof, that we do not report here because we would still need say something more and particularly because we would have to open the subject of recursive functions, which we don't want to do in this framework.

Suppose now that we found $\mathtt{quotient} \in \Lambda$. Then from (‡) we can set:
$\mathtt{reminder} := \lambda m\lambda n((\mathtt{sub})m)((\mathtt{mult})n)(\mathtt{quotient})mn$.
So the problem is reduced to find (if it exist) $\mathtt{quotient}$.

Now if we set $f(t,m,n) := (t+1)n - m$ we can rewrite (†) as: $q(m,n) = \begin{cases} 0 & \text{if } f(0,m,n) > 0 \\ 1 & \text{if } f(1,m,n) > 0 \\ \vdots \end{cases}$

We note that the function f is representable by a term $\mathtt{f} \in \Lambda$ that the reader can easily find.
Let's now consider the function $Q' : \mathbb{N}^{(\mathbb{N}^3)} \times \mathbb{N}^3 \to \mathbb{N}$ defined as:

$$Q'(Q,m,n,t) := \begin{cases} t & \text{if } f(t,m,n) > 0 \\ Q(m,n,t+1) & \text{if } f(t,m,n) = 0 \end{cases}$$

Also, Q' is represented by the term $\mathtt{Q'} := \lambda q\lambda m\lambda n\lambda t(((\mathtt{iszero})(\mathtt{f})tmn)(q)mn(\mathtt{succ})t)t \in \Lambda$.
But then the term $\mathtt{Q} := (Y)\mathtt{Q'}$ represents, by construction, the function $Q : \mathbb{N}^3 \to \mathbb{N}$,

$$Q(m,n,t) = \begin{cases} t & \text{if } f(t,m,n) > 0 \\ Q(m,n,t+1) & \text{if } f(t,m,n) = 0 \end{cases}$$

We note now that $q(m,n) = Q(m,n,0)$, so that we finally obtain that we can set:
$\mathtt{quotient} := \lambda m\lambda n(\mathtt{Q})mn\mathtt{0} \in \Lambda$.

We end this programming section remarking that the computation expressed in λ-calculus is *sequential* and not *parallel* (we leave at the intuitive level the meaning of those expressions). Indeed, for example it can be shown[28] that it cannot be implemented a program for the "parallel or", i.e.

$$\nexists\, \mathtt{parallelOr} \in \Lambda \text{ s.t. } \forall t,u \in \Lambda, \begin{cases} (\mathtt{parallelOr})tu \simeq_\beta \mathtt{True} & \text{if } M \text{ or } N \text{ head} - \text{normalizable} \\ (\mathtt{parallelOr})tu \text{ not head} - \text{normalizable} & \text{otherwise.} \end{cases}$$

Remark that such a program would be clearly computed with a parallel computation: simply simultaneously head-reduce both t and u and if at a certain point you find a head-normal form return $\mathtt{True}$. If you never find one you will just keep reducing forever, never returning an output.

2.3 Intuitionistic logic

Let's return on deductive systems. We will adopt a different point of view from sequent calculus, and we will introduce the so called "natural deduction" (introduced again by G. Gentzen). This is the system to which the we will stick for the rest of the thesis.

Let's say we want to modelize expressions of the form "from some hypotheses $\Gamma = A_1, \ldots, A_n$ I can prove a formula A". We decide to write $\Gamma \vdash A$ for that expression. We could use the same formulas as in definition 1.1, but for the following is better to choose to explicitly deal with $\to$ and $\bot$ (which is to be thought as the "absurd", the "false"). If we want to modelize the natural way of logical deducting we are led to the following rules (which are indeed clearly "natural" de-

[28] Proving this is completely out of our scope, and it typically requires to consider some more advanced techniques such as the theory of "syntactical continuity" and "B$\acute{}$ohm trees"..., which we won't even mention. You can see chapter 14 of [Bare84].

duction rules), and are based on the duality introduction/elimination rules for the connectives; an introduction rule (indexed by the suffix "i") represents the way we *construct* a connective, the elimination rules (indexed by the suffix "e") represents the way we *use* a connective:

Definition 2.26 (Minimal natural deduction NM)**.** *Expressions are defined exactly as in definition 1.1.*
Formulas $A ::= X(\vec{t}) \mid \bot \mid \neg A \mid A \wedge A \mid A \vee A \mid A \to A$.
Proofs in NM are defined following the same exact construction of definition 1.3, but using instead the following deduction rules:

$$\frac{}{\Gamma, A \vdash A}\text{ax}$$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\neg_i \qquad \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot}\neg_e$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\to_i \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\to_e$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\wedge_i \qquad \frac{\Gamma \vdash A_1 \wedge A_2}{\Gamma \vdash A_j}\wedge_e^j,\ j = 1, 2$$

$$\frac{\Gamma \vdash A_j}{\Gamma \vdash A_1 \vee A_2}\vee_i^j,\ j = 1, 2 \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\vee_e$$

We will often indicate the fact that π *is a proof of* $\Gamma \vdash A$ *by the writing* $\pi : (\Gamma \vdash A)$.

We will introduce other rules. When we write $\vdash$ it will always be non ambiguous what rules we are referring to, otherwise we will explicitly indicate the new rules by suffixing their names.

Remark 2.27.
- *Just a word on the* $\vee_e$*-rule: it is the reasoning by cases.*
- *We have not included the rules for the quantifications* $\forall$ *and* $\exists$*. Including them would cause no problem at all, but since in the following section we are going to focus on an even smaller fragment of NM we omit them from now.*

We have not mentioned an equivalent of the structural rules. This is because of the following theorem:

Proposition 2.28. *The structural rules in natural deduction, which are:*

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B}W \qquad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}C$$

are admissible, *in the sense that:* $\Gamma \vdash_{W+C} A \ \Rightarrow\ \Gamma \vdash A$.

Proof. One can see that it is enough to show that:
(1): if $\Gamma, A, A \vdash B$ provable with W and C then $\Gamma, A \vdash B$ provable without W and C, and
(2): if $\Gamma \vdash B$ provable with W and C then $\Gamma, A \vdash B$ provable without W and C.
We shall only deal with (1), (2) being analogous.
We prove (1) as follows: by induction on π we prove that: $\pi : (\Gamma, A, A \vdash_{W+C} B) \Rightarrow \exists\, \pi' : (\Gamma, A \vdash B)$.
(base case): the last rule of π is an ax-rule. Hence by definition of ax-rule it must be $B \in \Gamma, A$ (it is clear what we mean by saying that a formula belong to a multiset of formulas). But then the proof $\pi' := \dfrac{}{\Gamma, A \vdash B}\mathrm{ax}$ is a proof of $\Gamma, A \vdash B$, which is what we wanted.
(inductive step): let's consider the last rule R of π. We must treat all the possible cases, which are many. So we will only show one of them, the others being analogous. Let's see the case $R = \wedge_i$:
but then by definition of $\wedge_i$-rule it must be:
$B = C \wedge D$, for some C, D, and $\pi = \dfrac{\pi_1 : (\Gamma, A, A \vdash_{W+C} C) \qquad \pi_2 : (\Gamma, A, A \vdash_{W+C} D)}{\Gamma, A, A \vdash_{W+C} B} R.$
By inductive hypothesis on π_1 and π_2 we get that $\exists \pi_1', \pi_2'$ s.t. $\pi_1' : (\Gamma, A \vdash C)$ and $\pi_2' : (\Gamma, A \vdash D)$.
But then $\pi' := \dfrac{\pi_1' : (\Gamma, A \vdash C) \qquad \pi_2' : (\Gamma, A \vdash D)}{\Gamma, A \vdash C \wedge D}\wedge_i$ is a proof of $\Gamma, A \vdash B$, which is what we wanted. □

Definition 2.29. *Let's extend our system by introducing two other important rules:*

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\bot_e \qquad\qquad \frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}\mathrm{abs}$$

We set NJ:=NM+$\bot_e$ which is called intuitionistic *natural deduction,*
and NK:=NJ+abs *which is called* classical *natural deduction.*
In fact, as one should have remarked, the abs-rule corresponds to the so loved/hated reasoning by contradiction: if assuming the negation of A entails a contradiction, then we must have A.
It is an instructive non trivial exercise to prove that abs-rule is equivalent to both the following rules:

$$\frac{\Gamma, A \vdash B \qquad \Gamma, \neg A \vdash B}{\Gamma \vdash B}\mathrm{t.e.} \qquad\qquad \frac{\Gamma, A \to B \vdash A}{\Gamma \vdash A}\mathrm{p.l.}$$

which are the well known tertium non datur (one among A and $\neg A$ must be true, even if we don't know which) and the much less known Pierce's law. It is this last formulation of classical logic that we will adopt in the next chapter.

Remark 2.30.
- *Exactly as we did for the rules of sequent calculus we could develop a semantic justification for NK with which "justify" the validity of its rules, and find that also this system is sound and complete with respect to that semantic. We will not do that.*
- *NK and LK are equivalent, in the sense that one can prove (not trivial) that:*
  $\Gamma \vdash_{NK} A \Leftrightarrow \vdash_{LK} \neg(\Gamma^*), A^*$, where $(B \to C)^* := \neg B \vee C$ and $A^* := A$ otherwise.
  And more, one can define sequent calculus style systems LJ and LM which correspond to NJ and NM in the same sense.

Remark 2.31. *The rules $\neg_i$ and* abs *are similar but are not the same rule. In fact taking $A = \neg B$ in $\neg_i$ leads us to deduce $\neg A = \neg\neg B$, which is not the same formula as B, and similarly for* abs.
Now, in NK we have $\vdash_{NK} \neg\neg A \leftrightarrow A$ for every formula A, so that the two rules are actually equivalent.

But in NJ the situation is completely different:
in fact, it's easy to show that $P_{A,B} := ((A \to B) \to A) \to A$ is equivalent to the Pierce's law (in the sense that one can include p.l. *in NJ iff $\vdash_{NJ} P_{A,B}$, for all A, B). But one can prove that $\nvdash_{NJ} P_{A,B}$ for some A, B, so that* p.l. *is not admissible in NJ, i.e.* abs *is not.*
As a consequence of the previous argument, we also get that $\nvdash_{NJ} \neg\neg A \leftrightarrow A$, and since it's trivial to show that $\vdash_{NJ} A \to \neg\neg A$ we find $\nvdash_{NJ} \neg\neg A \to A$.
At the end, we have found that in NJ $\neg_i$ and abs *are not equivalent, since in NJ the double negation of a formula is not equivalent to that formula.*

The crucial point, which is the proof that $\exists A, B$ s.t. $\nvdash_{NJ} P_{A,B}$, is not at all easy. It is a consequence of the "typing system", that we will see in the next section, and of the crucial non trivial Gentzen's Hauptsatz *theorem, that we are about to see. We will not prove that $\nvdash_{NJ} P_{A,B}$ for some A, B, but we will do it for NM at the end of the following section. It is, however, clearly crucial to always have it in mind, since it is the crucial property of intuitionistic logic.*

If intuitionistic logic NJ doesn't allow reasoning by contradiction, which is clearly essential in mathematics, why would one be interest in it?
The answer lies in what we will see in the rest of this chapter, and can be resumed by saying that the internal structure of NJ has some remarkable properties (expressed by the Hauptsatz) that makes it extremely important for computer science (we can program with intuitionistic proofs, as we are going to see), for constructive mathematics and for foundational issues. We shall thus continue our study of NJ.

From now on the symbol $\vdash$ refers to $\vdash_{NJ}$.

Remark 2.32. *We can code the negation by $\neg A := A \to \bot$, and change the proofs by substituting $\neg_i$ with $\to_i$ and $\neg_e$ with $\to_e$. With this mappings we obtain a bijection between formulas and formulas without negation, and also between proofs and proofs without the rules $\neg_i$ and $\neg_e$.*
So from now on we will not treat negation, being already treated with the rules for $\to$ thanks to this remark.

Definition 2.33. *[Detours] A* detour *of a proof π is any subproof*[29] *of π which is an "objection" to the subformula property, i.e. a subproof of one of the following forms ($j = 1, 2$):*
- logical detours:

$$\dfrac{\dfrac{\rho : (\Gamma, A \vdash B)}{\Gamma \vdash A \to B} \to_i \qquad \rho' : (\Gamma \vdash A)}{\Gamma \vdash B} \to_e$$

[29] A subproof of a proof is obviously a subtree of the proof which is also a proof.

$$\dfrac{\dfrac{\rho_1 : (\Gamma \vdash A_1) \qquad \rho_2 : (\Gamma \vdash A_2)}{\Gamma \vdash A_1 \wedge A_2}\wedge_i}{\Gamma \vdash A_j}\wedge_e^j$$

$$\dfrac{\dfrac{\rho^j : (\Gamma \vdash A_j)}{\Gamma \vdash A_1 \vee A_2}\vee_i^j \qquad \rho_1 : (\Gamma, A_1 \vdash B) \qquad \rho_2 : (\Gamma, A_2 \vdash B)}{\Gamma, A_j \vdash B}\vee_e$$

- bureaucratic detours:

$$\dfrac{\dfrac{\rho_1 : (\Gamma \vdash A_1 \vee A_2) \qquad \rho_2 : (\Gamma, A_1 \vdash B \to C) \qquad \rho_3 : (\Gamma, A_2 \vdash B \to C)}{\Gamma \vdash B \to C}\vee_e \qquad \rho_4 : (\Gamma \vdash B)}{\Gamma \vdash C}\to_e$$

$$\dfrac{\dfrac{\rho_1 : (\Gamma \vdash A_1 \vee A_2) \qquad \rho_2 : (\Gamma, A_1 \vdash B_1 \wedge B_2) \qquad \rho_3 : (\Gamma, A_2 \vdash B_1 \wedge B_2)}{\Gamma \vdash B_1 \wedge B_2}\vee_e}{\Gamma \vdash B_j}\wedge_e^j$$

$$\dfrac{\dfrac{\rho_1 : (\Gamma \vdash A_1 \vee A_2) \qquad \rho_2 : (\Gamma, A_1 \vdash A \vee B) \qquad \rho_3 : (\Gamma, A_2 \vdash A \vee B)}{\Gamma \vdash A \vee B}\vee_e \qquad \rho_4 : (\Gamma, A \vdash C) \qquad \rho_5 : (\Gamma, B \vdash C)}{\Gamma \vdash C}\vee_e$$

We say that a proof π is normal *iff it contains no detours.*

Detours in natural deduction style play the role of the cut-rule in sequent calculus style.

Definition 2.34. *[Gentzen's proofs' transformations] We define a relation $\rightsquigarrow$ over the set of all proofs of NJ. Let's first set some notation.*
Given a proof π let's injectively label every node with an $i \in \mathbb{N}$. Then define the set:
$\mathrm{AX}_\pi := \{(i, \dfrac{}{\Delta_i, C_i \vdash C_i}\mathrm{ax}) \in \mathbb{N} \times \text{Proofs s.t. the } \mathrm{i}-\mathrm{th} \text{ node of } \pi \text{ is } \dfrac{}{\Delta_i, C_i \vdash C_i}\mathrm{ax}\}$.
Let's call for brevity $\mathrm{ax}^\pi_{\Delta_i, C_i, i} := (i, \dfrac{}{\Delta_i, C_i \vdash C_i}\mathrm{ax}) \in \mathrm{AX}_\pi$.
Given also a formula A let's set also $\mathrm{AX}^A_\pi := \{\mathrm{ax}^\pi_{\Delta_i, A, i} \in \mathrm{AX}_\pi\} \subseteq \mathrm{AX}_\pi$.
Given a proof $\pi : (\Gamma \vdash A)$ and a sequent $\Delta = E_1, \ldots, E_k, \Gamma$ ($k \geq 0$), we write π_Δ to indicate the proof obtained from π when substituting all its axioms $\mathrm{ax}^\pi_{\Delta_j, D_j, j} \in \mathrm{AX}_\pi$ with the respective axiom $\dfrac{}{E_1, \ldots, E_k, \Delta_j, D_j \vdash D_j}\mathrm{ax}$. Note that of course $\pi_\Delta : (\Delta \vdash A)$.
Given a proof π, a formula A, and for each $\mathrm{ax}^\pi_{\Delta_j, A, j} \in \mathrm{AX}^A_\pi$ a proof $\pi'_j : (\Delta_j \vdash A)$, we write $\pi[\pi'_j / \mathrm{ax}^\pi_{\Delta_j, A, j} \in \mathrm{AX}^A_\pi]$ to indicate the proof obtained from π when substituting all its axioms $\mathrm{ax}^\pi_{\Delta_j, A, j} \in \mathrm{AX}^A_\pi$ with the respective proof π'_j and erasing the superfluous[30] *A from π.*
Note that of course $\pi : (\Gamma \vdash B) \;\Leftrightarrow\; \pi[\pi'_j / \mathrm{ax}^\pi_{\Delta_j, A, j} \in \mathrm{AX}^A_\pi] : (\Gamma \vdash B)$.

[30] It can be a bit unclear. But in order to avoid a pointless excessive rigour in this definitions, the best way of explaining them would be at a blackboard, so we invite you to see the example which follows this definition to get the idea clear.

Now in order to define $\rightsquigarrow$ we firstly define a relation $\rightsquigarrow^1$.
Since looking at the rules we see that the context of the hypothesis always contain the context of the conclusion, with the above notation we can set ($j = 1, 2$):
- for logical detours:

$$\dfrac{\dfrac{\rho : (\Gamma, A \vdash B)}{\Gamma \vdash A \to B} \to_i \qquad \rho' : (\Gamma \vdash A)}{\Gamma \vdash B} \to_e \qquad \rightsquigarrow^1 \qquad \rho[\rho'_{\Delta_i} / \mathrm{ax}^{\rho}_{\Delta_i, A, i} \in \mathrm{AX}^{A}_{\rho}] : (\Gamma \vdash B)$$

$$\dfrac{\dfrac{\rho_1 : (\Gamma \vdash A_1) \qquad \rho_2 : (\Gamma \vdash A_2)}{\Gamma \vdash A_1 \wedge A_2} \wedge_i}{\Gamma \vdash A_j} \wedge^j_e \qquad \rightsquigarrow^1 \qquad \rho_j : (\Gamma \vdash A_j)$$

$$\dfrac{\dfrac{\rho^j : (\Gamma \vdash A_j)}{\Gamma \vdash A_1 \vee A_2} \vee^j_i \qquad \rho_1 : (\Gamma, A_1 \vdash B) \qquad \rho_2 : (\Gamma, A_2 \vdash B)}{\Gamma \vdash B} \vee_e \qquad \rightsquigarrow^1 \qquad \rho_j[\rho^j_{\Delta_i} / \mathrm{ax}^{\rho_j}_{\Delta_i, A_j, i} \in \mathrm{AX}^{A_j}_{\rho_j}] : (\Gamma \vdash B)$$

- for bureaucratic detours (with the notation of definition 2.33):

$$\dfrac{\dfrac{\rho_1 : (\Gamma \vdash A_1 \vee A_2) \qquad \rho_2 : (\Gamma, A_1 \vdash B \to C) \qquad \rho_3 : (\Gamma, A_2 \vdash B \to C)}{\Gamma \vdash B \to C} \vee_e \qquad \rho_4 : (\Gamma \vdash B)}{\Gamma \vdash C} \to_e$$

$$\rightsquigarrow^1 \qquad \dfrac{\rho_1 \qquad \dfrac{\rho_2 \qquad (\rho_4)_{\Gamma, A_1}}{\Gamma, A_1 \vdash C} \to_e \qquad \dfrac{\rho_3 \qquad (\rho_4)_{\Gamma, A_2}}{\Gamma, A_2 \vdash C} \to_e}{\Gamma \vdash C} \vee_e$$

$$\dfrac{\dfrac{\rho_1 : (\Gamma \vdash A_1 \vee A_2) \qquad \rho_2 : (\Gamma, A_1 \vdash B_1 \wedge B_2) \qquad \rho_3 : (\Gamma, A_2 \vdash B_1 \wedge B_2)}{\Gamma \vdash B_1 \wedge B_2} \vee_e}{\Gamma \vdash B_j} \wedge^j_e$$

$$\rightsquigarrow^1 \qquad \dfrac{\rho_1 \qquad \dfrac{\rho_2}{\Gamma, A_1 \vdash B_j} \wedge^j_e \qquad \dfrac{\rho_3}{\Gamma, A_2 \vdash B_j} \wedge^j_e}{\Gamma \vdash B_j} \vee_e$$

$$\dfrac{\dfrac{\dfrac{\rho_1 : (\Gamma \vdash A_1 \vee A_2) \qquad \rho_2 : (\Gamma, A_1 \vdash A \vee B) \qquad \rho_3 : (\Gamma, A_2 \vdash A \vee B)}{\Gamma \vdash A \vee B} \vee_e \qquad \rho_4 : (\Gamma, A \vdash C) \qquad \rho_5 : (\Gamma, B \vdash C)}{\Gamma \vdash C} \vee_e$$

$$\rightsquigarrow^1 \qquad \dfrac{\rho_1 \qquad \dfrac{\rho_2 \qquad (\rho_4)_{\Gamma, A_1} \qquad (\rho_5)_{\Gamma, A_1}}{\Gamma, A_1 \vdash C} \vee_e \qquad \dfrac{\rho_3 \qquad (\rho_4)_{\Gamma, A_2} \qquad (\rho_5)_{\Gamma, A_2}}{\Gamma, A_2 \vdash C} \vee_e}{\Gamma \vdash C} \vee_e$$

Then, we extend $\rightsquigarrow^1$ by taking the smallest (w.r.t. to inclusion) relation $\supseteq \rightsquigarrow^1$ and which is preserved by the deduction rules (we are just saying that from now on we set $\pi \rightsquigarrow^1 \pi'$ when π' is obtained by π by using some of the transformations above in a subproof of π).
We finally set $\rightsquigarrow$ to be the transitive and reflexive closure of $\rightsquigarrow^1$.

Example:
Let's take the following detour (in which the writing $\neg A$ is syntactic sugar for $A \to \bot$, as already explained):

$$d \;:=\; \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A, A, \neg A \vdash A} \quad \dfrac{}{A, A, \neg A \vdash \neg A}}{A, A, \neg A \vdash \bot}}{A, A \vdash \neg\neg A}}{A \vdash A \to \neg\neg A} \quad \dfrac{}{A \vdash A}}{A \vdash \neg\neg A}$$

Let's call $\rho' := \dfrac{}{A \vdash A}$ its right branch, $\rho : (A \vdash A \to \neg\neg A)$ its left branch and $\Delta := A, \neg A$ in the axiom $\dfrac{}{A, A, \neg A \vdash A}$ of ρ.
So we have: $\rho'_\Delta = \dfrac{}{\Delta \vdash A}$ and the transformation is:

$$d \quad \rightsquigarrow^1 \quad \dfrac{\dfrac{\rho'_\Delta \qquad \dfrac{}{A, \cancel{A}, \neg A \vdash \neg A}}{A, \cancel{A}, \neg A \vdash \bot}}{A, \cancel{A} \vdash \neg\neg A} \;=\; \dfrac{\dfrac{\dfrac{}{\Delta \vdash A} \quad \dfrac{}{A, \neg A \vdash \neg A}}{A, \neg A \vdash \bot}}{A \vdash \neg\neg A}$$

Remark 2.35. *We have that: $\pi : (\Gamma \vdash A) \rightsquigarrow \pi' \;\Rightarrow\; \pi' : (\Gamma \vdash A)$.*
Also, a proof is normal iff $\nexists \pi'$ s.t. $\pi \rightsquigarrow \pi'$.

While the first logical detour (i.e. lemmas) allow to write a readable and modular proof of a theorem, this transformations give a proof where each time we want to use a general result, instead of applying it, we prove it for the particular case that we need, obtaining a long, redundant and unreadable proof. They don't seem very interesting... but they are: in fact we are of course not interested in the "cleverer proof" - which is something that regards the creative act of the mathematician - but we are interested studying the mathematical structure of proofs. And from that point of view, they are extremely important.

Remark 2.36. *There would be still some other detours to consider, in order to get the theorem for full natural deduction (in particular the case of $\bot$ and of quantifiers). Adding them would have not added too much difficulty, but since we won't go into details of the proof, and we won't need the full theorem in the following, we didn't do it.*
But the main reason is that the existence of bureaucratic transformations (which are necessary to get a subformula property, which is the usual goal in cut-elimination), are the trace of the impurity of syntax of natural deduction style. Indeed, cut-elimination (also in the classical case) is much

better described in a sequent calculus style (where detours are internalized with the syntactic cut-rule), which however suffers of other impurities. It is interesting to notice (as in [GirLafTay89], page 8) that even if natural deduction is best suited for intuitionism, the most satisfying fragment of intuitionistic natural deduction is that of the non typically intuitionistic connectives, i.e. $\rightarrow, \wedge$ (and $\forall$).
The difficulty of the search of an "always well behaving" representation for proofs is something that we mentioned in the introduction. A very significant (but not conclusive) step in that direction would be to consider linear logic's proof-nets.

The definition of the transformations on proof in the previous definition is one of the two crucial ingredients to prove the following theorem. The other one being that the transformations do what they are expected to do, and which is what we put in the proof of the theorem. We put the definition of the transformations out of the proof of the theorem because they are important by themselves.

Theorem 2.37 (Gentzen's Hauptsatz)**.** $\forall \pi$, $\exists\ \pi'$ *normal s.t.* $\pi \rightsquigarrow \pi'$.

Proof. The idea is to measure the size $\|\pi\|$ of a proof π in a well-founded ordered set s.t. $\forall \pi$ that is not-normal, $\exists \pi'$ s.t. $\pi \rightsquigarrow^1 \pi'$ and $\|\pi'\| \lneq \|\pi\|$. Since $<$ is well-founded there are no infinite decreasing sequences, so that following such a sequence we have to find at a certain point a normal proof, and this concludes the proof of the theorem. Let's just remark that the trivial idea to take $\|\pi\|$ as the number of detours in π does not work, because due to bureaucratic transformations we can increase the number of detours (in the non bureaucratic cases, instead, they decrease). Take for example the reduction:

$$\dfrac{\rho_1 \quad \dfrac{\dfrac{\rho_2}{\Gamma, A_1 \vdash B \to C}\to_i \qquad \dfrac{\rho_3}{\Gamma, A_2 \vdash B \to C}\to_i}{\Gamma \vdash B \to C}\vee_e \qquad \rho_4}{\Gamma \vdash C}\to_e \qquad \rightsquigarrow^1$$

$$\dfrac{\rho_1 \quad \dfrac{\dfrac{\rho_2}{\Gamma, A_1 \vdash B \to C}\to_i \qquad (\rho_4)_{\Gamma, A_1}}{\Gamma, A_1 \vdash C}\to_e \qquad \dfrac{\dfrac{\rho_3}{\Gamma, A_2 \vdash B \to C}\to_i \qquad (\rho_4)_{\Gamma, A_2}}{\Gamma, A_2 \vdash C}\to_e}{\Gamma \vdash C}\vee_e$$

The idea to overcome the problem is that even if a bureaucratic transformation introduces new detours, they are logical ones and that bureaucratic detour has disappeared. We won't go into details, because it would be too much tedious difficulty for not that much interest, as we explained in the previous remark. A more complete treatment can be found in [GirLafTay89] (but with a slightly different syntax for natural deduction styles). It is however important to give the main ideas:
(1): first define the size $|A| \in \mathbb{N}$ of a formula A as the number of its connectives, or equivalently by induction on A: $|X(\vec{t})| := |\bot| := 0$, $|B \to C| := |B \wedge C| := |B \vee C| := |B| + |C| + 1$.
Then one defines the size $|\mathrm{d}| \in \mathbb{N}$ for a detour d, and with it define the size $\|\pi\| \in \mathbb{N}^{\mathbb{N}}$ of a proof π by setting: $\|\pi\| : \mathbb{N} \to \mathbb{N}$, $\|\pi\|(n) := Card(\{\text{detours } \mathrm{d} \text{ in } \pi \text{ s.t. } |d| = n\})$.
Being a proof a *finite* tree, there cannot be an infinite number of detours in it, and so the sequence of integers $\|\pi\|$ is in fact definitely null (i.e. starting from a certain point on, its elements are all

zero). We denote the set of definitely null sequences of integers with $\mathbb{N}^!$.
(2): let's define now a well-founded order on $\mathbb{N}^!$.
for $f, g \in \mathbb{N}^{\mathbb{N}}$, set[31] $f < g$ iff $\exists k \in \mathbb{N}$ s.t. $g(k) \lneq f(k)$ and $g \equiv f$ on $\{k+1, k+2, \dots\}$.
Let's just notice that $(\mathbb{N}^{\mathbb{N}}, <)$ is *not* well-founded, in fact we have for example $(0,1,2,3,4,5,\dots) > (1,0,2,3,4,5,\dots) > (2,1,1,3,4,5,\dots) > (3,2,1,2,4,5,\dots) > (4,3,2,1,3,5,\dots) > \dots$.
However if we restrict on $\mathbb{N}^!$ we obtain that the order is what in set theory is the *ordinal*[32] ω^ω, which being an ordinal, is well founded[33].
(3): let's start by noting that in general: $\pi \rightsquigarrow \pi' \not\Rightarrow \|\pi'\| \lneq \|\pi\|$, as one can easily check (use the first and in the second logical and bureaucratic transformations in which you duplicate some subproofs). However, it exists a strategy to choose at each step which detour to eliminate s.t. we have the desired weaker property: $\forall \pi$ not-normal, $\exists \pi'$ s.t. $\pi \rightsquigarrow \pi'$ and $\|\pi'\| \lneq \|\pi\|$.
A possible strategy consists in doing, at each step, the following:
- nothing if π is normal;
- reducing a maximal detour (in the sense that it does not contain other detours) if π is not normal (it is clear that if there is a detour, then there is also a maximal one, being a proof a finite object).
This strategy decreases the size, as one should verify. We won't do it. □

Lemma 2.38. *$\pi : (\vdash A)$ normal $\Rightarrow$ the last rule of π is an introduction rule.*
(Note that if the context Γ of π is not empty, then the lemma is obviously false: take $\dfrac{}{A \vdash A}\mathrm{ax}$).

Proof. Induction on π. Let R be its last rule:
(base case): $R = \mathrm{ax}$. This is not possible since an ax-rule requires a non empty context. So the implication of the lemma is true.
(inductive step): R is an introduction rule, an elimination rule or $\bot_e$.
If $R = \bot_e$ then $\pi = \dfrac{\pi' : (\vdash \bot)}{\vdash A}$ with π' normal, so by induction π' ends with introduction rule. But there is no introduction rule for $\bot$, contradiction.
If R is an elimination rule, then we are in one of the following cases:
$\pi = \dfrac{\pi' : (\vdash A \to B) \qquad \rho : (\vdash A)}{\vdash B} \to_e$ (for some π', ρ)

$\pi = \dfrac{\pi' : (\vdash A_1 \wedge A_2)}{\vdash A_j} \wedge_e^j$ (for some π')

$\pi = \dfrac{\pi' : (\vdash A \vee B) \qquad \rho : (A \vdash C) \qquad \rho' : (B \vdash C)}{\vdash C} \vee_e$ (for some π', ρ, ρ').
In any case π' is normal and so by induction π' ends with an introduction rule. But then π is a logical detour, so it's not normal, contradiction. □

Corollary 2.39 (Coherence of NJ)**.** $\nvdash \bot$

[31] This is the "Hydra's order". Indeed, you can imagine a Hydra for which each time one cuts one of its heads, there grow as many other smaller heads as you wish. If you measure now the power of Hydras by counting the number of heads of each possible size, you get our order $<$.

[32] In the intuitive and contradictory set theory of Cantor (i.e. set theory currently used in current mathematics), an ordinal is the equivalence class of a well-ordered set modulo order-isomorphism. For a non-contradictory definition - modulo Gödel's incompleteness theorem - see for example the axiomatic set theory ZF of Zeremelo and Fraenkel in [AbrTdF18] or [Kriv98].

[33] To kill our Hydra, Hercules would just have to use the well-foundedness of the order.

Proof. Let $\pi : (\vdash \bot)$. W.l.o.g. π is normal (thanks to the Hauptsatz), so that by lemma 2.38 we get that there exist a $\bot$-introduction rule, which is a contradiction because there is no such rule. □

Remark 2.40. *One can argue that if all the work has been done to achieve the coherence result then we could have omitted it, because we could reach the same result by semantic considerations. But this is not the point: the real point is that we obtained such a result in a purely* internal *way (and not referring to an external construction such as semantic models).*
As we already mentioned, this is a very deep point, and it is the first step to an internal "explanation" *of logic.*

Internal coherence is not the only important consequent of the Hauptsatz for NJ.

We have not included the quantifiers just for the seek of brevity. Everything can be done exactly the same with them, and it is extremely significant to mention this result in the case of quantifiers too:

Corollary 2.41. *(1).* $\pi : (\vdash A_1 \vee A_2) \;\Rightarrow\; \pi \rightsquigarrow \dfrac{\pi' : (\vdash A_i)}{\vdash A_1 \vee A_2}\vee_i$ *for some* $i = 0$ *or* $i = 1$.
(2). $\pi : (\vdash \exists x A) \;\Rightarrow\; \pi \rightsquigarrow \dfrac{\pi' : (\vdash A[t/x])}{\vdash \exists x A}\exists_i$ *for some term* t.

Proof. Trivial from Hauptsatz and lemma 2.38. □

Intuitionistic logic was conceptually introduced (not as a formal system as presented here) by L.E.J. Brouwer against the formalist school of Hilbert, and formalized by the so called Brouwer–Heyting–Kolmogorov interpretation and Kleene's realizability[34]. His philosophical position was that mathematical proofs should always be constructive, i.e. when proving a disjunction "A *or* B" we should be able (at least in principle) to give a proof of A or give a proof of B, and when proving "$\exists$ a s.t. $P(a)$" we should be able to give (at least in principle) as witness one of those such objects a. This position clearly rejects reasoning by absurd (or, equivalently the tertium non datur), because they allow one to prove things in an indirect way without being able to produce a witness of a disjunction or of an existential.
We can now be more precise and state that it is not that a proof of, say, an existential on a property, is a proof of the fact that some given element enjoys such a property, but the proof is an object that $\rightsquigarrow$-reduces to a proof of the fact that a produced element enjoys that property: we're just reading corollary 2.41.
And since $\rightsquigarrow$ is in fact an algorithmic procedure, we can say that an intuitionistic proof is a *program* that computes a witness of its statement: if it is a proof of $\exists$ a s.t. $P(a)$ then it computes one such a, and if it is a proof of "A *or* B" then it computes a proof of A or it computes a proof of B.
Recall that in the previous section we studied a particular syntax for algorithms, i.e. a programming language: in the next section we'll see how to write proofs in λ-calculus, and vice-versa.

[34] Kleene's realizability, also known as intuitionistic realizability, has clearly something to do with classical realizability, which is the other big subject of this thesis. But in classical realizability the technical and conceptual issues are completely different, and in fact we will not talk about intuitionistic realizability and for us the word realizability is synonym of classical realizability.

2.4 Simply typed λ-calculus

Let's start as usual with an heuristic:
consider a program as a black box, i.e. something of which you don't know the internal implementation, which has an output plug and eventually an input one. To make two such black boxes communicate we shall use *interfaces*, which introduce modularity: we choose an interface for the exit plug of the first box and plug it to the input plug of the second by means of a copy of the chosen interface. So we can connect two plugs only by means of two same interfaces. The interface is the intuition behind the notion of type, and it is usually described as being the information of *what* in general the program does.
If a black box has an interface A on its input plug and an interface B on its exit plug, we say that the black box has interface $A \to B$.
We are obviously lead to the following terminology:

Notation 2.42. *We will call a* (simple) type A *any formula A of propositional implicative natural deduction, i.e. $A ::= X \mid A \to A$ (X being a 0-ary second order variables).*

Take now as programming language the λ-calculus:
- the free variables x_i of a program $t = t\{x_1, \ldots, x_k\}$ should be seen just as containers to fill up with data to work with, and they have only an exit plug (to retrieve information from). Choosing an interface for them means choosing the way in which we want to read the information of their content;
- the bound variables of a term indicate just that the program can receive an input in that location, i.e. it has an input plug there. So if we have a black box $t = t\{x\}$ plugged to a container x by means of interfaces A, and with interface B on the exit plug of t, we can remove x, forming a new black box which has an input plug at the place of x and we can give to it the interface A while keeping the exit plug with interface B.
These considerations lead to the following definition (with the following notation, read $\Gamma \vdash t : A$ as "assigning variables as told by Γ, I can use the interface A on the output plug of t"):

Definition 2.43 (Simply typed λ-calculus)**.** - *We call a typing context a partial function Γ from the set of variables of λ-calculus to the set of types, s.t.* $\mathrm{Dom}(\Gamma)$ *is finite. We will indicate such a Γ sometimes with the writing $x_1 : \Gamma(x_1), \ldots, x_k : \Gamma(x_k)$, where $\{x_1, \ldots, x_k\} = \mathrm{Dom}(\Gamma)$. We say also that the variables x_i are declared in Γ. The writing $\Gamma, x : A$ (with $x \notin \mathrm{Dom}(\Gamma)$) stands for the typing context Γ' obtained by setting $\Gamma'(x) = A$ and $\Gamma'(y) = \Gamma(y)$ for $y \neq x$. We will also write $x \notin \Gamma$ to indicate that $x \notin \mathrm{Dom}(\Gamma)$, and similarly $A \notin \Gamma$ to indicate that $A \notin \mathrm{Im}(\Gamma)$.*

- *A* typing function *is a partial function f from the cartesian product of the set of typing contexts and the set λ-terms to the set of types, s.t., indicating the fact that $f(\Gamma, t) = A$ by writing $\Gamma \vdash t : A$, one has:*

(var) : $x \in \mathrm{Dom}(\Gamma) \;\Rightarrow\; \Gamma \vdash x : \Gamma(x)$

(@) : $\Gamma \vdash t : A \to B$ *and* $\Gamma \vdash u : A$, *for some* $A \;\Rightarrow\; \Gamma \vdash tu : B$

(λ) : *if* $x \notin \Gamma$ *then:* $\Gamma, x : A \vdash t : B \;\Rightarrow\; \Gamma \vdash \lambda x t : A \to B$.

We say that the 3-ple (Γ, t, A) is a (simply) typed λ-term, and we say that "t can be given the type A in the context Γ", when $\Gamma \vdash t : A$ for some typing function. We just write $\Gamma \vdash t : A$

instead of (Γ, t, A).
We say that a term t *is typable iff it can be given some type in some context.*

A typical way of writing the conditions $(\mathrm{var}), (\lambda), (@)$ *is by the so-called typing rules:*

$$\frac{}{\Gamma, x : A \vdash x : A}\mathrm{var}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x t : A \to B}\lambda \text{ (with } x \notin \Gamma\text{)} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}@$$

We can form trees as usual by combining those rules forming trees that are called typing derivations. A typing derivation is just a way of inductively writing a possible typing function. We will write $\pi : (\Gamma \vdash t : A)$ *to indicate that* π *is a typing derivation yielding* $\Gamma \vdash t : A$.

If we think of λ-calculus as a "general theory of functions" (from the intentional point of view), then untyped λ-calculus is the general study of functions without the need of specification of their domain/codomain, while typing amounts just to specify these information.

Remark 2.44. *Observe that since an abstraction and an application cannot be the same term, in the typing rules* $(\mathrm{var}), (@), (\lambda)$ *it holds also the converse implication "*$\Leftarrow$*". So it holds the "iff". We say that "the syntax of the term forces the typing derivation".*

Note that we are still up to α-equivalence, so for example the two different typing derivations $\dfrac{\dfrac{}{x : A \vdash x : A}}{\vdash \lambda x x : A \to A}$ and $\dfrac{\dfrac{}{y : A \vdash y : A}}{\vdash \lambda y y : A \to A}$ yield the same typed term $\vdash I : A \to A$.

Remark 2.45. *By an easy induction on a typing derivation for* t *we have:*
$x_1 : A_1, \ldots, x_k : A_n \vdash t : A \;\Rightarrow\; \mathrm{FV}(t) \subseteq \{x_1, \ldots, x_k\}$*, i.e. the free variables of a typed term are all declared in its context. Also, we have:*
Let $\Gamma, x : A \vdash t : B$*. Then:* $x \notin \mathrm{FV}(t) \;\Rightarrow\; \Gamma \vdash t : B$.
And we obtain the typed versions of the weakening rules:
Let $\Gamma \vdash t : B$*. Then* $\forall x \notin \Gamma$ *we have* $\Gamma, x : A \vdash t : B$ *for all* A.

Remark 2.46. *Clearly every typing derivation* $\pi : (x_1 : A_1, \ldots, x_k : A_n \vdash t : A)$ *induces a proof* $\rho : (A_1, \ldots, A_n \vdash A)$ *in the propositional implicative fragment of NM, just by forgetting all the terms.*
This correspondence is obviously surjective but not injective, in fact for example both the two different typing derivations $\dfrac{}{x_i : A \vdash x_i : A}\mathrm{var}$*, for* $x_1 \neq x_2$*, induce the same proof* $\dfrac{}{A \vdash A}\mathrm{ax}$.
If we identify the typing derivations that induce the same proof we obtain a bijection.
If π *is a proof, when writing* $\pi : (\Gamma \vdash t : A)$ *we mean that there is a typing derivation of* $\Gamma \vdash t : A$ *that induces* π*, and when writing* $\pi : (\Gamma \vdash t : A) \rightsquigarrow \pi'$*, for a proof* π'*, we mean that* $\pi \rightsquigarrow \pi'$ *and* $\pi : (\Gamma \vdash t : A)$.
The interest in all this constructions and simple observations is given by the following easy and fundamental result:

Lemma 2.47. *Detours transformation correspond exactly to β-reduction, i.e. (in our framework there is only one kind of detour) we have:*

$$\dfrac{\dfrac{\rho : (\Gamma, x : A \vdash t : B)}{\Gamma \vdash \lambda x t : A \to B} \qquad \rho' : (\Gamma \vdash u : A)}{\Gamma \vdash (\lambda x t)u : B} \quad \rightsquigarrow^1 \quad \pi : (\Gamma \vdash t[u/x] : B).$$

Thus we have:
$\pi : (\Gamma \vdash t : A)$ *and* $\pi \rightsquigarrow \pi' \;\Rightarrow\; \exists\, t'$ s.t. t $\to_\beta$ t$'$ and $\pi' : (\Gamma \vdash t' : A)$,
and conversely:
$\pi : (\Gamma \vdash t : A)$ *and* $t \to_\beta t' \;\Rightarrow\; \exists\, \pi'$ s.t. $\pi \rightsquigarrow \pi'$ and $\pi' : (\Gamma \vdash t' : A)$.

Proof. The second part follows immediately from the first one, for which we observe that:
by definition of $\rightsquigarrow^1$ we have $\pi = \rho[\rho'_{\Delta_i}/\mathrm{ax}^{\rho}_{\Delta_i,A,i} \in \mathrm{AX}^A_\rho]$ and one can check that the result follows from the definitions. □

Example: Recall the example of reduction of the detour:

$$d \;:=\; \dfrac{\dfrac{\dfrac{\dfrac{\overline{A, A, \neg A \vdash A} \qquad \overline{A, A, \neg A \vdash \neg A}}{A, A, \neg A \vdash \bot}}{A, A \vdash \neg\neg A}}{A \vdash A \to \neg\neg A} \qquad \overline{A \vdash A}}{A \vdash \neg\neg A}$$

done in the previous section.
Now type it in order to obtain[35] a typing derivation $d : (z : A \vdash (\lambda x \lambda y(y)x)z : \neg\neg A)$,
which yields a typing derivation $\pi : (\vdash \lambda z(\lambda x \lambda y(y)x)z : A \to \neg\neg A)$.
Note that reducing the only redex we have: $\lambda z(\lambda x \lambda y(y)x)z \to^1_\beta \lambda z \lambda y(y)z$.
And in fact using the reduction done in the example of the previous section we get:

$$\pi \qquad \rightsquigarrow^1 \qquad \dfrac{\dfrac{\dfrac{\overline{z : A, y : \neg A \vdash z : A} \qquad \overline{z : A, y : \neg A \vdash y : \neg A}}{z : A, y : \neg A \vdash yz : \bot}}{z : A \vdash \lambda y(y)z : \neg\neg A}}{\vdash \lambda z \lambda y(y)z : A \to \neg\neg A}$$

Observe that while the fact that simple types and implicative propositional formulas are identifiable is quite predictable, we have introduced the notions of β-reduction and transformation of detours from two point of view a priori completely different (and also historically they appeared in separate perspectives). The fact that they correspond perfectly is the sign that they are just two faces of a same behind phenomenon.
It is important to keep in mind that this is not only a coincidence due to the extremely simple

[35] Actually, in our definition the formula $\bot$ is not considered a type, so we couldn't do that. It's not a problem, since for example one can include it as a type and, as already mentioned, eliminate its corresponding detours etc. It is only for a brevity reason that we didn't include it, so in this example you can treat $\bot$ as a type without any problems.

framework that we decided to expose, but the same correspondence between normalization of proofs and execution of functional type systems still appears in much more advanced frameworks, such as, for example, Girard's second order system F.

Corollary 2.48 (Subject reduction). *Let* $t \to_\beta t'$. *Then:* $\Gamma \vdash t : A \ \Rightarrow \ \Gamma \vdash t' : A$.

Proof. Follows immediately from the lemma since the conclusion context and formula of a proof are invariant under detour transformation. □

Another way of writing those observations is:

Corollary 2.49 (Curry-Howard isomorphism). *Up to identification of typed terms which induce the same proof, our correspondence is an isomorphism between the rewriting systems* $(\mathrm{Proofs}, \rightsquigarrow)$ *and* $(\mathrm{TypedTerms}, \to_\beta)$, *if we set the reduction on typed terms as:* $\Gamma \vdash t : A \ \to_\beta \ \Gamma \vdash t' : A$ *iff* $t \to_\beta t'$.

Proof. The reduction is well defined thanks to subject reduction. The rest is lemma 2.47. □

In our system we can't type every term:

Remark 2.50. *A term of the form* $(x)x$ *(with* x *variable) is not typable. In fact, if* $\Gamma \vdash xx : A$ *then* $\Gamma \vdash x : B \to A$ *and* $\Gamma \vdash x : B$, *for the same* B. *So* $x \in \Gamma$ *and then by the typing rules we get* $B \to A = A$, *which is an equation with no solutions.*
Hence for example, $\Delta = \lambda x(x)x$, *and thus* $\Omega = \Delta\Delta$, *is not typable in our system.*
However, not every term of the form tt *is not typable. For example, for every* A, *the following is a typing derivation for* $\vdash (I)I : A \to A$.

$$\dfrac{\dfrac{\overline{x : A \to A \vdash x : A \to A}}{\vdash \lambda x x : (A \to A) \to (A \to A)} \qquad \dfrac{\overline{y : A \vdash y : A}}{\vdash \lambda y y : A \to A}}{\vdash (I)I : A \to A}$$

Observe also that the converse of the subject reduction, the "subject expansion", does not hold, i.e. even if $t \to_\beta t'$ *and* $\Gamma \vdash t' : A$ *it can be* $\Gamma \nvdash t : A$. *In fact take for example* $t = (\lambda dy)(x)x$ *(of course with* $y \neq d$*) and* $t' = y$, *which is typable for example under the context* $\Gamma = x : A$ *while* t *is not only not typable under the context* Γ, *but it is not typable under any context, since that would mean to also type* xx.

The main interest in typing system is to provide a priori characteristics of programs, such as:

Theorem 2.51 (Weak normalization). t *typable* $\Rightarrow$ t *normalizable.*

Proof. $\Gamma \vdash t : A \ \Rightarrow \ \exists \pi : (\Gamma \vdash t : A)$, and since π is normalizable by detour elimination, thanks to lemma we have $t \to_\beta t'$, for some t' normal. □

A direct application of the weak normalization theorem is that the fixed point operators Y is not typable. In fact it is obviously not normalizable, since by definition $Y \to_\beta \lambda f(f)(Y)f$.

We only mention another fundamental result that refine the previous one:

Theorem 2.52 (Strong normalization)**.** *t typable $\Rightarrow$ t strongly-normalizable.*

Proof. We omit the proof, since it would require some pages of work and new ideas. You can see for example [Kriv93]. □

Remark 2.53. *Observe that the converse implication does not hold; for example $(\lambda dy)(x)x$ is strongly normalizable but not typable.*

Let's fix $Iter_A := (A \to A) \to (A \to A)$, for A simple type. As the two following propositions say, this is the type of integers represented in base 1, i.e. of Church numerals.

Proposition 2.54. $\forall A$ type, $\forall n \in \mathbb{N}$, $\vdash \mathtt{n} : Iter_A$.

Proof. Easy induction on n:
- case $n = 0$: $\dfrac{\overline{f : A \to A, x : A \vdash x : A}^{\text{var}}}{\vdash \mathtt{0} : (A \to A) \to (A \to A)}\lambda$
- case $n \geq 1$: by inductive hypothesis we know that $\vdash \mathtt{n\text{-}1} : (A \to A) \to (A \to A)$ and so $f : A \to A, x : A \vdash (f)^{n-1}x : A$. But then we have:

$$\dfrac{\dfrac{\overline{f : A \to A, x : A \vdash f : A \to A}^{\text{var}} \quad \overline{f : A \to A, x : A \vdash (f)^{n-1}x : A}}{f : A \to A, x : A \vdash (f)^n x : A}@}{\vdash \mathtt{n} : (A \to A) \to (A \to A)}\lambda$$

□

Proposition 2.55. *If $\vdash t : Iter_X$ $\forall X$ 0-ary second order variable, then $t \to_\beta \mathtt{m}$ for some $m \in \mathbb{N}$, or $t = \lambda xx$.*

Proof. By weak normalization and subject reduction we get: $t \to_\beta u$, with u normal and s.t. $\vdash u : Iter_X$.
Now, one can easily prove that normal terms have the following inductive characterization:
u normal $\Leftrightarrow$ $u = \lambda x_1 \dots \lambda x_k (y) v_1 \dots v_n$ for some $k, n \in \mathbb{N}$, y variable (possibility equals to some x_i), and v_i normal.
Now, if $y \in \mathrm{FV}(u)$ then y is declared in the context of $\vdash u : Iter_X$, which is empty. So, $y = x_i$ for some $i = 1, \dots, k$. So in particular we also have $k \geq 1$. Since $Iter_X$ is of the form $B_1 \to B_2 \to X$ (for some types B_1, B_2), with X variable (and so $X \neq C \to D$ for any C, D), and since for every λ in u there is a $\to$ in its type, we must have $k \leq 2$. Now let's consider the two possible cases:
- case $k = 1$: then $y = x_1$, so that $u = \lambda x(x)v_1 \dots v_n$. But then from the typing rules we get $x : X \to X \vdash (x)v_1 \dots v_n : X \to X$.
Now, one can easily prove by induction on n that if $\Gamma \vdash (x)v_1 \dots v_n : D$, then $\exists A_1, \dots, A_n$ s.t. $\Gamma \vdash v_1 : A_1$, ..., $\Gamma \vdash v_n : A_n$, and $\Gamma \vdash x : A_1 \to (\cdots \to (A_n \to D))$.
Thus for our u it must be: $x : X \to X \vdash x : A_1 \to (\cdots \to (A_n \to (X \to X)))$, for some A_i. But

the only possibility for this to happen is to be $n = 0$. So we obtain $u = \lambda xx$, which is in the thesis.
- case $k = 2$: then $u = \lambda x_1 \lambda x_2 (y) v_1 \dots v_n$, with $y \in \{x_1, x_2\}$. From the typing rules we thus obtain $x_1 : X \to X, x_2 : X \vdash (y) v_1 \dots v_n$. We have again two cases:
- if $y = x_2$: then by using the same argument as before we get $n = 0$ and so $u = \lambda x \lambda y y = \mathtt{0}$, which is in the thesis.
- if $y = x_1$: using the same argument we get $n = 1$, so that $u = \lambda f \lambda x (f) v$, with v normal.
One can easily prove that if $f : X \to X, x : X \vdash v : X$ and v is normal, then $v = (f)^p x$ for some $p \in \mathbb{N}$.
So we obtain $u = \lambda f \lambda x (f) v = \lambda f \lambda x (f) (f)^p x = \mathtt{p+1}$, which is in the thesis. □

Remark 2.56. *As one can expect, it can be easily verified that:*
$\vdash \mathtt{sum} : Iter_A \to (Iter_A \to Iter_A)$ *just by straightforwardly writing a typing derivation, so that also* $\vdash (\mathtt{sum})\ \mathtt{n}\ \mathtt{m} : Iter_A$*, for all* $n, m \in \mathbb{N}$.
One can verify that what we have programmed in the programming section is well typed.
An important application of typing systems is to provide a method to prove the correctness *of some programs: roughly speaking, if for a program* `f` *one produces a typing derivation which corresponds to a proof of the proposition* $n \in \mathbb{N} \Rightarrow f(n) \in \mathbb{N}$ *for a recursive function* f *- we say in this case that* f *is "provably total", then the the program* `f` *must compute the values of* f*, so that we obtain the certitude of the correctness*[36] *of* `f`*. This paradigm is expressed by the slogan "programming with proofs" and you can consult [TdF06] for an introduction.*

Recall the formula $P_{A,B}$ equivalent to Pierce law: $P_{A,B} = ((A \to B) \to A) \to A$.
As announced, we have:

Theorem 2.57. *For all* X, Y *0-ary second order variables,* $\nvdash_{NM} P_{X,Y}$
Recall that this implies that reasoning by contradiction is not admissible in NM.

Proof. It is enough to show that $\forall t$ closed term, $\nvdash t : P_{X,Y}$. By absurd let $\vdash t : P_{X,Y}$.
W.l.o.g. (by Haupsatz) t is normal. So (we already made that remark) $t = \lambda x_1 \dots \lambda x_n (x) t_1 \dots t_k$, with t_i normal. Since t is closed then $x \in \{x_1, \dots, x_n\}$ so $n \geq 1$. But since X is a second order variable then $n \leq 1$. So $t = \lambda x (x) t_1 \dots t_k$. From the typing rule we have
$x : (X \to Y) \to X \vdash (x) t_1 \dots t_k : X$. For the typing constraints it must be $k = 1$, so $t = \lambda x (x) t_1$. From that and the typing rules we obtain $x : (X \to Y) \to X \vdash t_1 : X \to Y$. Now, t_1 being normal, we have $t_1 = \lambda y_1 \dots \lambda y_m (y) u_1 \dots u_l$. Since Y is a second order variable we get $m \leq 1$, and for the typing constraints it must be $m = 1$. Being t closed, we have $y \in \{x, y_1\}$, and since if $x = y$ we don't respect the typing constraints, we get $t_1 = \lambda y (y) u_1 \dots u_l$. But now, from $x : (X \to Y) \to X \vdash \lambda y (y) u_1 \dots u_l : X \to Y$ we get $x : (X \to Y) \to X, y : X \vdash (y) u_1 \dots u_l : Y$, so that y must be given a functional type in a context that declares it to be atomic. This is a contradiction. □

[36] For a programmer, the problem of being sure the program he just wrote does exactly what it was supposed to do (i.e. he really programmed what he thought to be programming) is a very important issue, and can have also catastrophical consequences if a non correct program - supposed to be correct - is employed in some real software.

2.5 Some topics in proof/programs correspondence

What we presented until now really was just the basic technical and conceptual framework of the Curry-Howard paradigm. We understand now that it provides a deeper insight in the question of proofs, in all the philosophical, mathematical and computer science senses. We will see in the next chapter how one can push to all its extension the mathematics/programs paradigm. But first, we want to conclude this chapter by just giving some ideas of some important topics that are founded on what we presented:

- **Other type systems:**
  logic tells us that there are other connectives than implication. Let's see if we can extend the Curry-Howard correspondence to some bigger natural deduction system. We can do it adding to the λ-calculus new constructions:
  Types: propositional minimal logic plus: $A \wedge B \mid A \vee B$.
  Programs: λ-calculus plus the constructions:

$$< t, u > \mid (\pi_1)t \mid (\pi_2)t \mid (\iota_1)t \mid (\iota_2)t \mid \mathtt{case}\{t; x_1.u_1, x_2.u_2\}$$

  (here x, x_1, x_2 are variables, $< t, u >$ is a *new* term and π_i is a *new* symbol. In particular $< t, u >$ is *not* the term defined in the previous section. Also `case` is a new symbol, as well as the punctuation symbols ".", ", ", ", "; ", "}", "{").
  The typing rules are the usual plus:

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash < t, u > : A \wedge B} \qquad \frac{\Gamma \vdash t : A_1 \wedge A_2}{\Gamma \vdash (\pi_j)t : A_j} \quad (j = 1, 2)$$

$$\frac{\Gamma \vdash t : A_j}{\Gamma \vdash (\iota_j)t : A_1 \vee A_2} \quad (j = 1, 2)$$

$$\frac{\Gamma \vdash t : A \vee B \qquad \Gamma, x_1 : A \vdash u_1 : C \qquad \Gamma, x_2 : B \vdash u_2 : C}{\Gamma \vdash \mathtt{case}\{t; x_1.u_1, x_2.u_2\} : C}$$

  We extend β-reduction setting (in addition to the usual reduction):

$$(\pi_i) < t_1, t_2 > \to^1_\beta t_i$$

$$\mathtt{case}\{(\iota_i)t; x_1.u_1, x_2.u_2\} \to^1_\beta u_i[t/x_i]$$

$$(\pi_j)\mathtt{case}\{t; x_1.u_1, x_2.u_2\} \to^1_\beta \mathtt{case}\{t; x_1.(\pi_j)u_1, x_2.(\pi_j)u_2\}$$

$$(\mathtt{case}\{t; x_1.u_1, x_2.u_2\})v \to^1_\beta \mathtt{case}\{t; x_1.u_1 v, x_2.u_2 v\}$$

$$\mathtt{case}\{\mathtt{case}\{t; x_1.u_1, x_2.u_2\}; y_1.v_1, y_2.v_2\} \to^1_\beta \mathtt{case}\{t; x_1.\mathtt{case}\{u_1; y_1.v_1, y_2.v_2\}, x_2.\mathtt{case}\{u_2; y_1.v_1, y_2.v_2\}\}$$

  The first two (and the usual reduction) are the logical transformations: the intuition for the first reduction is obvious; the second may be less clear, but it's just a case evaluation on t, proceeding then on the respective term. The other reductions operate bureaucratic transformations.

In this system Curry-Howard isomorphism still holds (see "Strong Normalization of Classical Natural Deduction with Disjunction" by Ph. de Groote) and, for example, subject reduction and strong normalization keep holding too.
There are many other type systems, in which one can characterize various notions of reduction (for example the "intersection types" characterize head-reduction,...).

It would have been natural to use the term $\lambda u \lambda v \lambda f(f)uv$ for the couple $< u, v >$ and the projections `Rproj`, `Lproj` that we defined in the previous chapter, instead of adding them as new instructions in the programming language. However it turns out that in a first order setting this is not possible.
Whereas, in a second order framework (so, dealing also with quantification), we don't need to add new constructions in the λ-calculus is order to treat couples.
The main second order system is *system* F, introduced in the '70 by J.-Y. Girard[37] in his PhD thesis [Gir72] (and two years later rediscovered by J.C. Reynolds), which is a *polymorphic* programming language corresponding, by Curry-Howard isomorphism, to second order intuitionistic logic. The syntax is the following:
Types: propositional minimal logic plus the second order quantification $\forall XA$.
Programs: λ-calculus.
The typing rules are the usual plus:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall XA}\forall_i^2 \quad \text{with } X \notin \Gamma \qquad\qquad \frac{\Gamma \vdash t : \forall XA}{\Gamma \vdash t : A[B/X]}\forall_e^2 \quad \text{for all } B \text{ type.}$$

Everything typable with simple types is obviously also typable in F, but second order quantification (i.e. polymorphism) makes the converse far from being true. Indeed, for example recall that xx and $\Delta = \lambda xxx$ are not simply typable, while in F we have, for any A, B types:

$$\dfrac{\dfrac{\dfrac{}{x : \forall XX \vdash x : \forall XX}\text{var}}{x : \forall XX \vdash x : A \to B}\forall_e^2 \qquad \dfrac{\dfrac{}{x : \forall XX \vdash x : \forall XX}\text{var}}{x : \forall XX \vdash x : A}\forall_e^2}{x : \forall XX \vdash xx : B}@$$

and so also $\vdash \Delta : \forall XX \to B$.
But still, system F is not trivial, in the sense that we can't type everything. For example, we can't type $\Omega = \Delta\Delta$.
Also the typing and type verification problems are highly non trivial: in 1998 J.B. Wells proved that those are both undecidable problems[38]!
Girard proved strong normalization for F, i.e. if t is typable in F then[39] t is strongly normalizable (which is clearly not the case for Ω). This is a difficult and fundamental result (proved introducing "Girard's idea" of "reducibility candidates[40]"), and corresponds, at the other side of the Curry-Howard isomorphism, to cut-elimination for second order intuitionistic logic[41]. One can see the difficulty noting that, being a cut-elimination proof,

[37] It seems that nobody knows what the "F" in "system F" stands for. Girard says he piked it by chance.

[38] This has some consequences in the "type inference problem" at the level of a concrete implementation of F as a real programming language running on a computer. The majority of the functional programming languages are based on F or some of its restrictions and variants.

[39] The converse implication is false, but it's not easy to find a counterexample.

[40] Idea that has then been adapted by J.-L. Krivine to the other cases to get an elegant proof of strong normalization of, for example, simple types. Indeed, also Krivine's classical realizability's construction - which is the content of the third chapter - is inspired by the very same idea.

[41] This result was the original motivation for which Girard introduced F. Indeed, cut-elimination for second order intuitionistic logic was an open problem till 1966 - the "Takeuti's conjecture" - and his result proved it in a syntactic way (the other known proof of some years before being semantic).

it gives as corollary a consistency proof of second order (classical) arithmetic (passing through the double-negation coding to handle classical logic in intuitionistic one), and by Gödel's incompleteness theorem it can't thus be proved only with arithmetical (i.e. "easy") methods.
In F we can encode couples (i.e. $\wedge$), disjoint sums (i.e. $\vee$), second order existential quantification (and also booleans, integers...) using the powerful second order quantification.
Recall the terms $< u, v >$, $\texttt{couple}, \texttt{Lproj}$ and $\texttt{Rproj}$ from section 2.2, and
fix $A \wedge B := \forall X((A \to (B \to X)) \to X)$. Then:
it is a simple exercise to see that for all A, B types, if $\rho : (\Gamma \vdash u : A)$ and $\rho' : (\Gamma \vdash v : B)$ then $\Gamma \vdash < u, v > : A \wedge B$ (and also $\vdash \texttt{couple} : \forall X \forall Y (X \to (Y \to X \wedge Y))$).
We just obtained hence the introduction rule for $\wedge$.
We can simulate also the elimination rules as follows:
if $\rho : (\Gamma \vdash t : A \wedge B)$ then $\Gamma \vdash (\texttt{Lproj})t : A$ (and $\vdash \texttt{Lproj} : \forall X \forall Y ((X \wedge Y) \to X)$).
In fact:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\Gamma, x : A \wedge B \vdash x : A \wedge B}}{\Gamma, x : A \wedge B \vdash x : (A \to (B \to A)) \to A}\forall_e^2 \qquad \dfrac{\textit{easy} \quad \textit{peasy}}{\Gamma, x : A \wedge B \vdash \texttt{True} : A \to (B \to A)}}{\Gamma, x : A \wedge B \vdash x\texttt{True} : A}}{\Gamma \vdash \lambda x(x)\texttt{True} : (A \wedge B) \to A} \qquad \rho : (\Gamma \vdash t : A \wedge B)}{\Gamma \vdash (\texttt{Lproj})t : A}$$

The same for $\texttt{Rproj}$.
In the same way we find that we can take $A \vee B := \forall X((A \to X) \to ((B \to X) \to X))$ and analogous for encoding $\exists X A := \forall Y((\forall X(A \to Y)) \to Y)$.

- **Denotational and categorical semantics:**
physics teaches us that in a dynamical system is always important to study the invariants of the dynamic; our main dynamic is that of $(\Lambda, \to_\beta)$ and in particular (Proofs, $\rightsquigarrow$).
The general idea is then to associate with every type A a mathematical object $[\![A]\!]$ (a set, a vector space, a topological space...), called its *interpretation*, to extend it to contexts Γ obtaining $[\![\Gamma]\!]$, and finally to define it on typed terms s.t. $[\![\pi : (\Gamma \vdash t : A)]\!] : [\![\Gamma]\!] \longrightarrow [\![A]\!]$ and $\pi \rightsquigarrow \pi' \;\Rightarrow\; [\![\pi]\!] = [\![\pi']\!]$ (and similarly for pure λ-calculus).
Such a construction is called *denotational semantics*, and it was introduced by D. Scott in the 70's using continuous functions between complete partial orders in order to model λ-calculus.
We just want to give the prototypical basic example, which is however instructive, placing ourselves in the propositional implicative fragment of NM:
let's fix a set $[\![X]\!]$ for every second order variable X;
we extend the interpretation to formulas by setting $[\![A \to B]\!] := [\![B]\!]^{[\![A]\!]}$ (functions from $[\![A]\!]$ to $[\![B]\!]$);
we extend to contexts setting $[\![A_1, \ldots, A_k]\!] := [\![A_1]\!] \times \cdots \times [\![A_k]\!]$;
finally we define $\pi : (\Gamma \vdash A) : [\![\Gamma]\!] \longrightarrow [\![A]\!]$ by induction as:
$\left[\!\!\left[\dfrac{}{\Gamma, A \vdash A}\text{ax} \right]\!\!\right] : (\vec{x}, a) \in [\![\Gamma, A]\!] \longrightarrow a \in [\![A]\!]$
$\left[\!\!\left[\dfrac{\rho : (\Gamma \vdash A \to B) \qquad \rho' : (\Gamma \vdash A)}{\Gamma \vdash B}@ \right]\!\!\right] : \vec{x} \in [\![\Gamma]\!] \longrightarrow [\![\rho]\!](\vec{x})([\![\rho']\!](\vec{x})) \in [\![B]\!]$
$\left[\!\!\left[\dfrac{\rho : (\Gamma, A \vdash B)}{\Gamma \vdash A \to B}\lambda \right]\!\!\right] : \vec{x} \in [\![\Gamma]\!] \longrightarrow [\![\rho]\!](\vec{x}, \cdot) \in [\![B]\!]^{[\![A]\!]}$.
We have: $\left[\!\!\left[\dfrac{\dfrac{\rho : (\Gamma, A \vdash B)}{\Gamma \vdash A \to B}\lambda \qquad \rho' : (\Gamma \vdash A)}{\Gamma \vdash B}@ \right]\!\!\right] = [\![\rho[\rho'_{\Delta_i}/\text{ax}^\rho_{\Delta_i, A, i} \in \text{AX}^A_\rho]]\!]$, and so we get a denotational semantics with sets.

In general we deal with some mathematical structures and "natural" functions between them... the intuition suggests us that we should be able to define an elegant and general categorical setting of it. And in fact it is true:
we can say that a denotational semantic for, let's say, typed λ-calculus, is the data of:
a cartesian[42] category $\mathcal{C}$ and an interpretation $[\![\cdot]\!]$ of types A, contexts $\Gamma = x_1 : A_1, \dots, x_k : A_k$, typing derivations $\pi : (\Gamma \vdash t : A)$, s.t.
$[\![x_1 : A_1, \dots, x_k : A_k]\!] = [\![A_1]\!] \times \cdots \times [\![A_k]\!]$ (parenthesis are not needed up to isomorphism),
$[\![\Gamma \vdash t : A]\!] \in \mathrm{Hom}_{\mathcal{C}}([\![\Gamma]\!], [\![A]\!])$ and $t \to_\beta u \ \Rightarrow \ [\![\Gamma \vdash t : A]\!] = [\![\Gamma \vdash u : A]\!]$.
Just to give the taste of it, we mention that for λ-calculus we have a categorical characterization, which is given by the following result, known sometimes as the *Curry-Howard-Lambek* isomorphism, which requires the notion of cartesian closed category. It is out of the aim of this thesis to introduce such notions, but we can just say that while the notion of cartesian category allows one to have products, the closed clause allows one to produce the so-called "curryfication", which in the world of functions is the fact that an $f : A \times B \to C$ is "the same" of an $f : A \to C^B$.

Theorem 2.58. *A cartesian closed category $\mathcal{C}$ induces a denotational semantic for typed λ-calculus.*
Conversely, the following category $\mathcal{C}$ is cartesian closed:
$\mathrm{Obj}_{\mathcal{C}} = \{(A_1, \dots, A_k)$ s.t. $k \in \mathbb{N}$ and A_i simple type$\}$
$\mathrm{Hom}_{\mathcal{C}}((A_1, \dots, A_k), (B_1, \dots, B_m)) =$ *the quotient set:*
$\{(t_1, \dots, t_m)$ terms s.t. $x_1 : A_1, \dots, x_k : A_k \vdash t_i : B_i\} / \simeq_{\beta\eta}$
(where $\simeq_{\beta\eta}$ is the β-equivalence together with the η-equivalence, which is another important equivalence relation that we didn't mention).

Proof. A proof of this would require precise definitions and results in category theory, but this is not a thesis on those topics. □

Let's just remark that such results say that the notion of computation has a mathematical structured meaning, and research in the so-called *categorical logic* plays a major role in nowadays research. You can see for example [AbramTze10] for an introduction to the subject.

- **Linearity:**
analysis teaches us that polynomial functions have the remarkable property of approximating a function f (with certain regularities) near a point. In particular the affine function $f(0) + x \left.\dfrac{df}{dx}\right|_{x=0}$ is the simpler approximation of f near 0. If we consider functions as algorithms, we see that the multiplicity of a variable in a polynomial indicates the number of times the function uses its argument in that place. In particular a linear function uses its argument exactly once. So we have the intuition that there can be a link between the analytic/algebraic notion of linearity and some resource sensitive notion in the programs' world.
This intuition is indeed a very solid one, and the introduction and development of *linear logic* in the 80's by J.-Y. Girard ([Gir87]) is one of the most important discoveries in proof-theory.

[42] A cartesian category $\mathcal{C}$ is the data of a category $\mathcal{C}$ together with a *product object* $A \times B$ and *projections* $\pi^A_{A\times B}, \pi^B_{A\times B}$, for every couple of objects A, B of $\mathcal{C}$. Actually, it is required to also give a *terminal object*, but we are just giving intuitions and we don't need to be too precise.

Historically, linear logic emerged at a semantical level when, refining the analysis of "Scott-domains" (some denotational models for λ-calculus) by means of the "coherent semantics", where formulas are *coherent spaces* (some graphs) and connectives are constructions on those, Girard found the possibility of decomposing the intuitionistic implication "$\to$" by the fundamental equation

$$A \to B = (!A) \multimap B$$

introducing !(.) which says that its argument can be used as many times as we want, and the *linear* implication $A \multimap B$ which says that in the passage from A to B we have to use A *exactly* once, i.e. we *consume* the resource A to produce B. Note that the usual implication $A \to B$ makes use of A an arbitrary number of times: this is the meaning of the structural rules C and W, which we already mentioned.
Then he introduced the decomposition at a syntactic level, by abolishing the structural rules (and recovering them only under some particular conditions, by the so-called *exponential* connectives), obtaining a *linear* sequent calculus.
One sees that once this has been done, the usual connectives such as $\wedge$ and $\vee$ split in two categories, the so-called *multiplicatives* ($\otimes$ for $\wedge$ and $\parr$ for $\vee$) and *additives* (& for $\wedge$ and $\oplus$ for $\vee$). The reader has probably noticed that those are exactly the names that we directly used in the first chapter for the *rules* for the connectives $\wedge$ and $\vee$ in LK.
In general linear logic is really a decomposition of classical logic which allows one to have a deep vision both on classical and intuitionistic logic, keeping the good features of both, in particular thanks to the role of the involutive *linear* negation $(.)^{\perp}$ (for which the use of the same symbol as "orthogonality" is not a coincidence). For, instance, since β-reduction corresponds to intuitionistic implication we understand that β-reduction is not an *elementary* step of execution, and we obtain a resource sensitive decomposition of it.
There are many news with the arrival of linear logic, from the categorical models to the so called "proof-nets" - a graph-theoretical representation of proofs enjoying cut-elimination and (several) topological characterization(s) - and from the introduction by T. Ehrhard and L. Regnier (years 2000) of the *differential* λ-calculus - in which one Taylor-expands λ-terms - to other consequences that we will mention at the very end... but this is another story and we would need another thesis to explore the technical and conceptual ideas and applications. So we will just content ourselves with this informal lines, that could not have been omitted.

3 Krivine's classical realizability theory

> *We see now a little better what a mathematician does: he writes programs in a low level language (like assembler) and he types them.*
> J.-L. Krivine, "Wigner, Curry et Howard"
> Exposé au colloque de sciences cognitives ARCo'04, Compiègne, 2004
> (my translation from french)

As we saw, Tarski semantics is a satisfactory modelization of the notion of satisfiability of formulas by a certain mathematical structure. It interprets a formula with a bit, 0 or 1, and so it is not interested in the different proofs of a formula. For first-order logic and also for some classes of second-order logic, thanks to the completeness theorem, this amounts to focusing on the sole question of provability, and not on the study of the proofs themselves.
As we have seen, a typical tool for approaching the study of proofs is Curry-Howard's correspondence between intuitionistic logic and λ-calculus. The correspondence was limited to the intuitionist fragment for a long time, and it seemed that it could not be extended to classical logic, given the link with the notion of calculation, and therefore of construction, which is clearly lost when reasoning by absurd.
The turning point came in the 90's, when in [Griff90] the computer scientist T. Griffin gave a computational interpretation of Pierce's law, which is equivalent to the rule of absurd. The idea was to associate Pierce's law with the "control for continuations" instructions, which were already known in languages like SCHEME, or also in Java with the "throw/catch" instruction to handle exceptions.
According to this programming instructions that explicitly deal with the execution stack, the associated notion of calculation is now to see programs running only in an environment, i.e. a lambda-term is executed in front of a stack, which is a finite sequence of arguments (the "execution stack"). Hence only the processes, pairs of a lambda-term and a stack, are executed, and not programs by itself.
Even if this is completely different from what we studied before (where for example cut-elimination = execution, while now cut-elimination plays no role), we still rest in the Curry-Howard spirit, in the sense that the approach is still the correspondence between proofs and programs. We simply adapt it to the classical case.
This is the point of view of the *classical realizability theory*, which was introduced and developed by the mathematician J.-L. Krivine in the years 2000 (starting with [Kriv04] and going on). Starting from the Curry-Howard correspondence for all classical logic he gives a new notion of semantics for formulas, which are interpreted in a Boolean algebra over the power set of the set of stacks, much larger than the $\{0,1\}$ of Tarski semantics, which makes it possible to discriminate between different proofs of a formula, and which is the link between the notion of semantics and that of computation: in fact intuitively the stacks associated with a formula A, which are called "the falsity values of A" or "tests for A", are the tests that a program must pass to be considered as a witness of A. We see that A can also be associated with another set, namely that of the terms that win against all the falsity values of A, i.e. the strategies that pass all the tests in the game "A". We obtain thus a semantic based on the notion of interaction between strategies and tests in a game, the game induced by the formula.
In [Kriv01] and other papers, Krivine shows how one can extend the proof-program correspondence to set theory (in which one can write "all" mathematics), thanks to the idea of "introducing new instructions" to the programming language, and obtaining in this way programs, for example, for the axiom of countable and dependent choice (which are enough to do analysis) the axiom of the ultrafilter (which we will discuss in this thesis, and which is a weaker form of the

general axiom of choice), the continuous hypothesis, and others.
It is important to understand that this is something new, because with the usual proof/programs correspondence one associates programs with... proofs! Now, an axiom has no proof, so we are stuck, unless introducing an ad hoc instruction. The point is to define a *meaningful* programming language, i.e. when introducing new instructions giving them a *meaningful* execution. For example, the real point of the introduction of the fundamental instruction for the absurd-rule is its execution, given by Griffin, and that is not ad hoc at all.
It naturally arises the problem, called by Krivine "the specification problem", of studying the common behaviour of the various programs associated with a fixed formula. Note that the technique of typing gives the information of "*what*" the program does; here we want to know "*how*" it does it. For some classes of interesting (and not too complicated) formulas the problem is solved (and gives very interesting programming interpretations), but in general it remains a difficult open problem.
Furthermore, quite surprisingly, it happens that the developed theory is in some sense a generalization of the *forcing*, the renowned theory developed by P. Cohen (and which earned him the Fields medal in 1966!) in order to construct new models of the set theory ZF and prove with that the independence from ZF of axioms such as the axiom of choice and the continuum hypothesis. In [Kriv08], which is our basic reference, it is introduced the notion of "structures de réalisabilité" (which we will call, as was done later, *realizability algebras*) which gives a general framework to manage at the same time realizability and forcing[43]. Following and generalizing Krivine's ideas, in [Miq12] and [Rieg14], A. Miquel and L. Rieg studied forcing from a syntactic point of view in the context of higher order arithmetic $PA\omega^+$: it becomes a transformation of formulas which is induced by a transformation of programs that has a very interesting computer science interpretation: it's about passing from the execution of lambda-terms in a "superuser" mode for the "normal" formulas, to the execution in a "protected" mode for the formulas of forcing.
As a generalization of forcing, also realizability (in the form of realizability algebras' theory) gives new models - for example of ZF - richer and more complicated than those of forcing. The idea is based on the completeness theorem (for first order logic or for a certain notion of second order logic): from a realizability algebra we construct a theory in a language which, under certain hypotheses, is a coherent theory, and therefore, for a certain notion of model, admits some models - called realizability models. Understanding their structure (and in particular the role of the axiom of choice) is one of the major concerns in nowadays research in classical realizability.

What we present here is a quick treatment of the standard case of second order Peano's arithmetic $PA2$, introduced in [Kriv04], followed by the detailed discussion of the realization of the ultrafilter axiom ("it exists a non trivial ultrafilter on $\mathbb{N}$") in the framework of $PA2$, following [Kriv08].

It is important to know that the current research in classical realizability is nowadays generally done in the first order framework of ZF, by means of an intermediate theory introduced by Krivine, called ZF_ϵ, for which you can see [Kriv01].

[43] It must be said that this is not the last, nor the better way of doing that: see for example A. Miquel's "Implicative algebras", as an evolution of realizability algebras and forcing.

3.1 Curry-Howard correspondence for classical logic

We follow [Kriv04]. The programming language we use is the λ-calculus extended with new instructions, including one called `callcc` (which stands for "call-with-current-continuation"), and some called "continuations". It's called λ_c-calculus. The execution, as we have already said, is no longer a relation between λ-terms but we must define the execution environment, which is given by a stack, i.e. a finite sequence of terms.

Definition 3.1 (λ_c-calculus). *Let* Var *be a countable set whose elements are called "variables", let* $\Pi_0 \neq \varnothing$ *be a set whose elements are called "stack constants", and let* $I \ni$ `callcc` *be a set whose elements are called "instructions". We define by double induction the sets* Λ_c *of* λ_c*-terms and* Π_c *of stack as follows:*
$\Lambda_c ::= \ x \mid c \mid \kappa_\pi \mid (t)u \mid \lambda x t$, *where* $x \in Var,\ c \in I,\ \pi \in \Pi_c,\ t, u \in \Lambda_c$
$\Pi_c ::= \ \alpha \mid t.\pi$, *where* $\alpha \in \Pi_0,\ t \in \Lambda_c,\ \pi \in \Pi_c$.

As usual, we will sometimes write $t\{x_1, \ldots, x_n\}$ *to indicate that the free variables of* t *are all in the set* $\{x_1, \ldots, x_n\}$*, and we define substitution of a term for a variable in a term avoiding the capture of variables, and we consider the terms modulo* α*-equivalence.*
The terms of the form κ_π *are called* continuations *(associated with the stack* π*).*
We set $QP := \{t \in \Lambda_c \text{ s.t. } t \text{ does not contain continuations}\}$*, and its elements are called proof-like terms (from the french "quasi-preuves").*
We note $\Lambda_c \times \Pi_c$ *by* $\Lambda_c \star \Pi_c$*,* (t, π) *by* $t \star \pi$*, and we say that the couple* $t \star \pi$ *is a process.*

Notation 3.2. *As usual, we write* $(t)u_1, \ldots, u_n$*, or also* $(t)\vec{u}$ *when we don't want to specify the integer* n*, instead of* $(\ldots((t)u_1)\ldots)u_n$.
We write $t[\vec{u}/\vec{x}]$ *to indicate the substitution* $t[u_1/x_1, \ldots, u_n/x_n]$ *when we don't want to specify the integer* n.
We write sometimes $t \star \vec{u}.\pi$ *instead of* $t \star u_1.\ldots.u_n.\pi$ *when we don't want to specify the integer* n.

The execution of processes is given by the *Krivine Abstract Machine* (noted: KAM).

Definition 3.3 (execution in the KAM). *We say that a relation* $\succ$ *over* $\Lambda_c \star \Pi_c$ *is an* execution relation *iff it is* transitive, irreflexive, *and if* $\forall t, u \in \Lambda_c, \forall \pi, \rho \in \Pi_c$ *we have:*

(push) $(t)u \star \pi \succ t \star u.\pi$;

(grab) $\lambda x t \star u.\pi \succ t[u/x] \star \pi$;

(save) `callcc` $\star\, t.\pi \succ t \star \kappa_\pi.\pi$;

(restore) $\kappa_\pi \star t.\rho \succ t \star \pi$.

Remark 3.4. *The execution in the KAM simulates the weak head reduction*[44] *in the usual* λ*-calculus, in the sense that one can prove that:*
let t, u *be usual* λ*-terms s.t.* t *reduces to* u *by weak head reduction and* u *is in normal form for that*

[44] We say that a λ-term t reduces to a λ-term u by weak head reduction iff $t = (\lambda x t')v'\vec{v}$ *and* $u = (t'[v'/x])\vec{v}$. One can prove that a λ-term u is in normal form for that reduction iff $u = \lambda x v$ or $u = (x)v$, for some variable x and some term v.

reduction. Then we have:

$$t \star \pi \succ' \begin{cases} \lambda x v \star \pi, & \text{if } u = \lambda x v \\ x \star \vec{v}.\pi, & \text{if } u = (x)\vec{v} \end{cases}$$ *where $\succ'$ is $\succ$ limited at (push) and (grab) only.*

Hence in particular: $\forall \pi \in \Pi_c \ \exists p \in \Lambda_c \star \Pi_c \ s.t. \ t \star \pi \succ' p \prec' u \star \pi$.

The logical system that we use is the $\{\rightarrow, \forall\}$ fragment of the second order classical natural deduction (noted: $NK2$), which has the following syntax:

Definition 3.5 (NK2)**.** *Let's fix a countable infinity of individual variables, of second order variables (each one with a certain arity) and some function symbols (each one with a certain arity). We define the set $\mathcal{T}$ of expressions and the set $\mathcal{F}$ of* NK2 *formulas as follows:*
$\mathcal{T} ::= x \mid f(\vec{e})$*, where x is an individual variable, f is a function symbol of arity k and $\vec{e}$ is a finite sequence of k expressions $e_i \in \mathcal{T}$;*
$\mathcal{F} ::= X(\vec{e}) \mid A \rightarrow B \mid \forall x A \mid \forall X A$*, where X is a second order variable of arity k, $\vec{e}$ is a finite sequence of k expressions $e_i \in \mathcal{T}$, $A, B \in \mathcal{F}$ and x is an individual variable.*
As usual, we will sometimes write X^k to indicate that the arity of the second order variable X is k. As habit, we write $A\{x_1, \ldots, x_n\}$ [resp. $A\{X_1, \ldots, X_n\}$] to indicate that the variables $x_1, \ldots, x_n$ [resp. $X_1, \ldots, X_n$] potentially occur free in A (and as usual when we don't want to specify he integer n).
As usual, we define the substitution of an expression e for the individual variable x in a formula $A\{x\}$, noted by $A[e/x]$, and the substitution[45] *of a formula B for the second order variable X^k and individual variables $x_1, \ldots, x_k$ in a formula $A\{X\}$, noted by $A[B/Xx_1, \ldots, x_k]$ or also $A[B/X\vec{x}]$ when we don't want to specify the integer k. We will write sometimes $A[\vec{e}/\vec{x}]$ instead of $A[e_1/x_1, \ldots, e_k/x_k]$ when we don't want to specify the integer k.*
One can define a notion of bound occurrence of a(n individual or second order) variable in a formula by saying that the quantifier $\forall$ binds the quantified variable. With that, we will consider formulas modulo α-equivalence.

Curry-Howard for classical logic amounts to the following definition:

Definition 3.6 (λ_c-calculus typing system for NK2)**.** *The* NK2 *formulas are the types for the λ_c-terms. Proofs are constructed as always, using the following deduction rules of* NK2 *which are the typing rules for λ_c-calculus:*

$$\frac{}{\Gamma, x : A \vdash x : A}(ax) \qquad \frac{}{\Gamma \vdash \texttt{callcc} : ((A \rightarrow B) \rightarrow A) \rightarrow A}(l.p.)$$

$$\frac{\Gamma \vdash t : A \rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash (t)u : B}(@) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x t : A \rightarrow B}(\lambda)$$

[45] It may be useful to report the definition of this operation here: the formula $A[B/X\vec{x}]$ is defined to be the formula obtained from A when, for each free occurrence of X in A, replacing all the subformulas of A of the form $X(\vec{e})$ with the formula $B[\vec{e}/\vec{x}]$.

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x A}(\forall_i) \text{ with } x \text{ not free in } A \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X A}(\forall_i^2) \text{ with } X \text{ not free in } A$$

$$\frac{\Gamma \vdash t : \forall x A}{\Gamma \vdash t : A[e/x]}(\forall_e) \text{ with } e \in \mathcal{T} \qquad \frac{\Gamma \vdash t : \forall X^k A}{\Gamma \vdash t : A[B/Xx_1, \ldots, x_k]}(\forall_e^2) \text{ with } B \in \mathcal{F}$$

Remark 3.7. *The considered fragment is enough to encode all the other logical symbols, using the usual coding (for example that of system F):*
$\bot := \forall X^0 X$; $\neg A := A \to \bot$;
$A \wedge B := \forall X^0((A \to B \to X) \to X)$; $A \vee B := \forall X^0((A \to X) \to (B \to X) \to X)$;
$\exists y A := \forall X^0((\forall y(A \to X)) \to X)$; $\exists Y A := \forall X^0((\forall Y(A \to X)) \to X)$,
where we use the notation: $A_1 \to \ldots \to A_n := A_1 \to (A_2 \to \ldots (A_{n-1} \to A_n)\ldots)$.

We now give the realizability theory for the second order Peano's arithmetic (noted: PA2).

Definition 3.8 (PA2)**.** *The language of* PA2 *is a second order language where among the function symbols there are: a constant (function of arity 0)* 0*, two functions* $succ$ *and* $pred$ *of arity 1, two functions* $+$ *and* $\cdot$ *of arity 2;*
We set $x = y$ *as syntactic sugar for the formula* $\forall X^1(X(x) \to X(y))$.
The theory PA2 *is given by the usual second order Peano's axioms:*

1. $\forall x \neg(succ(x) = 0)$
2. $\forall x \forall y(succ(x) = succ(y) \to x = y)$
3. $\forall x(x + 0 = x)$
4. $\forall x \forall y(x + succ(y) = succ(x + y))$
5. $\forall x(x \cdot 0 = 0)$
6. $\forall x \forall y(x \cdot succ(y) = (x \cdot y) + x)$
7. $\forall x Nat\{x\}$, *where* $Nat\{x\} := \forall X^1((\forall y(Xy \to Xsucc(y))) \to X0 \to Xx)$.

Definition 3.9 (Standard realizability for PA2)**.** *Let's fix an interpretation, i.e. an application* $[\![.]\!] : \mathrm{Var} \to \mathbb{N}$ *extended to* $\mathcal{T}$ *by* $[\![0]\!] = 0$, $[\![succ(e)]\!] = [\![e]\!] + 1$, $[\![pred(e)]\!] = [\![e]\!] - 1$, $[\![e + e']\!] = [\![e]\!] + [\![e']\!]$, $[\![e \cdot e']\!] = [\![e]\!] \cdot [\![e']\!]$, $[\![f(\vec{e})]\!] = \overline{f}(\vec{[\![e]\!]})$, *where* $\overline{f} : \mathbb{N}^k \to \mathbb{N}$, *with* k *the arity of* f *(note that* $[\![x]\!] \in \mathbb{N}$*);*
let $\succ$ *be an execution relation over* $\Lambda_c \star \Pi_c$*, and let* $\perp\!\!\!\perp \subseteq \Lambda_c \star \Pi_c$ *s.t.*

$$\forall p, p' \in \Lambda_c \star \Pi_c \ s.t. \ p \succ p', \ p' \in \perp\!\!\!\perp \Rightarrow p \in \perp\!\!\!\perp$$

(we'll say that $\perp\!\!\!\perp$ *is* $\succ$-saturated*);*
let $\overline{(.)} : \{\text{second order variables}\} \to \bigcup_{n \in \mathbb{N}} \mathscr{P}(\Pi_c)^{\mathbb{N}^n}$ *s.t.* $\overline{X^k} : \mathbb{N}^k \to \mathscr{P}(\Pi_c)$, *and let's set* $U^\perp := \{t \in \Lambda_c \ s.t. \ \forall \pi \in U, \ t \star \pi \in \perp\!\!\!\perp\}$, *for an* $U \subseteq \Pi_c$.
For a better readability, we will indicate in this definition with ρ *the function s.t.* $\rho(e) = [\![e]\!]$ *and* $\rho(X) = \overline{X}$.
We define inductively the set $\|A\|_{\rho, \perp\!\!\!\perp} \subseteq \Pi_c$ *of* tests *(or falsity values) for a formula* A *of* PA2 *as follows:*

$\|X(\vec{e})\|_{\rho,\perp\!\!\!\perp} := \overline{X}(\vec{[\![e]\!]})$

$\|A \to B\|_{\rho,\perp\!\!\!\perp} := \{t.\pi\}_{t\in[\![A]\!]^{\perp}_{\rho,\perp\!\!\!\perp},\pi\in\|B\|_{\rho,\perp\!\!\!\perp}} =: \|A\|^{\perp}_{\rho,\perp\!\!\!\perp}.\|B\|_{\rho,\perp\!\!\!\perp}$

$\|\forall x A\|_{\rho,\perp\!\!\!\perp} := \bigcup_{n\in\mathbb{N}} \|A\|_{\rho[n/x],\perp\!\!\!\perp}$, *where* $\rho[n/x](x) = n$ *and* $\rho[n/x](y) = \rho(y)$ *if* $y \neq x$

$\|\forall X^k A\|_{\rho,\perp\!\!\!\perp} := \bigcup_{\mathcal{H}\in\mathscr{P}(\Pi_c)^{\mathbb{N}^k}} \|A\|_{\rho[\mathcal{H}/X],\perp\!\!\!\perp}$, *where* $\rho[\mathcal{H}/X](X) = \mathcal{H}$ *and* $\rho[\mathcal{H}/X](Y) = \rho(Y)$ *if* $Y \neq X$.

We set $|A|_{\rho,\perp\!\!\!\perp} := \|A\|^{\perp}_{\rho,\perp\!\!\!\perp}$ *and we call it the set of the* realizers *(or truth values) of* A.
We write $t \Vdash_{\rho,\perp\!\!\!\perp} A$ *instead of* $t \in |A|_{\rho,\perp\!\!\!\perp}$. *For a closed formula all those definitions don't depend on* $[\![.]\!], |.|$ *and we will omit the dependencies.*

Remark 3.10. *1): One can prove that:*
if $\perp\!\!\!\perp$ *is s.t.* $\forall\theta \in QP,\ \theta \notin \|\perp\|^{\perp}$, *then the relation* $\leq$ *on* $\mathscr{P}(\Pi_c)$ *defined by:*
$U \leq V$ iff $\exists\theta \in QP\ s.t.\ \theta \in (U^{\perp}.V)^{\perp}$
is a preorder and $(\mathscr{P}(\Pi_c), \leq)$ *is a boolean algebra.*

2): All Peano's axioms but the induction principle (axiom 7) are realized by a proof-like term (any proof-like term for axiom 2 and λxx *for the others), and the term is* independent *from the choice of the pole* $\perp\!\!\!\perp$. *Using our KAM, the induction principle cannot be realized independently from the pole not even adding new instructions. This is an important result, since it has the consequence that in the realizability models there are integers, non-standard integers, but also elements that are not integers at all. Even if it will not play a significant role for us, we will however prove it (theorem 3.11). In particular, it is related to the fact that the execution in the KAM is deterministic; in fact one can define a non-deterministic KAM adding a new instruction* $\pitchfork$ *and in this setting one can realize induction. See Rieg's thesis [Rieg14] for details.*

We will prove a fundamental result, the "adequation lemma", in the next section: it's theorem 3.24. We do it there because it is in the general setting that we will need, but it can be proven, exactly in the same way, also at this point. So we will use it for the next results of this section, because they are instructive to report here.

Theorem 3.11. $\nexists t \in \mathrm{QP}$ s.t. $\forall\perp\!\!\!\perp$ pole, $t \Vdash_{\perp\!\!\!\perp} \forall x Nat\{x\}$.

Proof. By absurd suppose that such t exists. Then, $\forall\perp\!\!\!\perp$ pole, $t \Vdash_{\perp\!\!\!\perp} Nat\{0\}$ and $t \Vdash_{\perp\!\!\!\perp} Nat\{1\}$.
Let's fix $\Delta = \lambda x(x)x$, $\Omega_0 := ((\Delta)\Delta)\mathtt{0}$, $\Omega_1 := ((\Delta)\Delta)\mathtt{1}$ and $\pi \in \Pi_c$.
Let $\perp\!\!\!\perp_i := \{p \in \Lambda_c \star \Pi_c \text{ s.t. } p \succ \Omega_i \star \pi\}$, $i = 0, 1$. Both $\perp\!\!\!\perp_0$ and $\perp\!\!\!\perp_1$ are poles. Let's fix X, Y two 1-ary second order variables and let's take an interpretation of 1-ary second order variables ρ s.t. $\rho(X)(0) = \{\pi\}$ and $\rho(X)(j) = \varnothing$ if $j > 0$, and $\rho(Y)(0) = \varnothing$ and $\rho(X)(j) = \{\pi\}$ if $j > 0$.
Now we have $\Omega_0 \Vdash_{\rho,\perp\!\!\!\perp\perp\!\!\!\perp_0} X[0]$.
And since $\|X[m] \to X[succ(m)]\|_{\rho,\perp\!\!\!\perp_0} = |X[m]|_{\rho,\perp\!\!\!\perp_0}.\|X[succ(m)]\|_{\rho,\perp\!\!\!\perp_0} = |X[m]|_{\rho,\perp\!\!\!\perp_0}.\varnothing = \varnothing\ \forall m \in$

$\mathbb{N}$, then $\forall \xi \in \Lambda_c,\ \xi \in \|\forall y(X[m] \to X[succ(m)])\|_{\rho,\perp\!\!\!\perp_0}$, i.e. $|\forall y(X[m] \to X[succ(m)])|_{\rho,\perp\!\!\!\perp_0} = \Lambda_c$.
More, we have that $\forall \xi \in \Lambda_c,\ \xi \Vdash_{\rho,\perp\!\!\!\perp_1} Y[0]$, and since $\|Y[m] \to Y[succ(m)]\|_{\rho,\perp\!\!\!\perp_1} = |Y[m]|_{\rho,\perp\!\!\!\perp_1}.\{\pi\}\ \forall m \in \mathbb{N}$ (we're using what we found below), we have that $\forall \xi \in \Lambda_c,\ \lambda x\Omega_1 \star \xi.\pi \succ \Omega_1[\xi/x] \star \pi = \Omega_1 \star \pi \in \perp\!\!\!\perp_1$, i.e. $\lambda x\Omega_1 \Vdash_{\rho,\perp\!\!\!\perp_1} \forall y(Yy \to Ysucc(y))$.
But now recall that $t \Vdash_{\perp\!\!\!\perp} Nat\{0\}$ and $t \Vdash_{\perp\!\!\!\perp} Nat\{1\}$ for any pole $\perp\!\!\!\perp$. We have:
$t \Vdash_{\perp\!\!\!\perp_0} Nat\{0\} \ \Rightarrow\ t \Vdash_{\rho,\perp\!\!\!\perp_0} (\forall y(X[y] \to X[succ(y)])) \to X[0] \to X[0] \ \Rightarrow\ t \star \lambda x\Omega_1.\Omega_0.\pi \in \perp\!\!\!\perp_0 \ \Rightarrow\ t \star \lambda x\Omega_1.\Omega_0.\pi \succ \Omega_0 \star \pi$; and:
$t \Vdash_{\perp\!\!\!\perp_1} Nat\{1\} \ \Rightarrow\ t \Vdash_{\rho,\perp\!\!\!\perp_1} (\forall y(Y[y] \to Y[succ(y)])) \to Y[0] \to Y[1] \ \Rightarrow\ t \star \lambda x\Omega_1.\Omega_0.\pi \in \perp\!\!\!\perp_1 \ \Rightarrow\ t \star \lambda x\Omega_1.\Omega_0.\pi \succ \Omega_1 \star \pi$.
But this is a contradiction, since we found a process $(t \star \lambda x\Omega_1.\Omega_0.\pi)$ which reduces both to $\Omega_0 \star \pi$ and $\Omega_1 \star \pi$. □

Just to give the taste of it we will solve the *specification* problem for two extremely easy cases: second order (i.e. polymorphic) identity and booleans.
We will use, only for the following two results, the symbol $\lambda x_1 \dots \lambda x_k t \sim u$ meaning that $\forall v_1, \dots, v_k \in \Lambda_c, \forall \pi \in \Pi, \exists p \in \Lambda_c \star \Pi$ s.t. $\lambda \vec{x} t \star \vec{v}.\pi \succ p \prec u \star \vec{v}.\pi$.

Proposition 3.12 (Specification for the polymorphic identity)**.** $\vdash \theta : \forall X^0(X \to X) \ \Rightarrow\ \theta \sim \lambda xx$.

Proof. We show that $\theta \star v.\pi \succ v \star \pi$. Let $\perp\!\!\!\perp := \{p \in \Lambda_c \star \Pi \text{ s.t. } p \succ v \star \pi\}$. If we show that $\theta \star v.\pi \in \perp\!\!\!\perp$ we have finished, since it would mean that $\theta \star v.\pi \succ v \star \pi$.
Observe that $\perp\!\!\!\perp$ is a pole. Let's fix an interpretation ρ of 0-ary second order variables s.t. $\rho(X) = \{\pi\} \subseteq \Pi$. From $\vdash \theta : \forall X^0(X \to X)$ by adequation lemma (theorem 3.24) we get $\theta \Vdash_{\perp\!\!\!\perp} \forall X^0(X \to X)$, and so $\theta \Vdash_\rho X \to X$, i.e. $\theta \in (\|X\|_\rho \to \|X\|_\rho)^\perp = (\rho(X)^\perp.\rho(X))^\perp = (\{\pi\}^\perp.\pi)^\perp$. But by definition of $\perp\!\!\!\perp$ we have $v \in \{\pi\}^\perp$, and hence $\theta \star v.\pi \in \perp\!\!\!\perp$. □

Proposition 3.13 (Specification for the booleans)**.** *Let* $\mathrm{Bool}\{\mathrm{x}\} := \forall X^1(X[1] \to X[0] \to X\{x\})$. *We have:*
$\vdash \theta : \mathrm{Bool}[0] \ \Rightarrow\ \theta \sim$ **`False`** $(= \lambda x\lambda y y)$
$\vdash \theta : \mathrm{Bool}[1] \ \Rightarrow\ \theta \sim$ **`True`** $(= \lambda x\lambda y x)$.

Proof. We only prove the second case, the first being analogous.
As in the last proof, we want to show that $\theta \star v_1.v_2.\pi \in \{p \in \Lambda_c \star \Pi_c \text{ s.t. } p \star v_1.\pi\} =: \perp\!\!\!\perp$.
We can see that $\perp\!\!\!\perp$ is a pole. Let's fix an interpretation ρ of 1-ary second order variables s.t. $\rho(X^1)(1) = \{\pi\}$ and $\rho(X^1)(n) = \varnothing$ if $n \neq 1$. From $\vdash \theta : \mathrm{Bool}[1]$ we get $\vdash \theta : X[1] \to X[0] \to X[1]$ and so by adequation lemma $\theta \Vdash_{\rho,\perp\!\!\!\perp} X[1] \to X[0] \to X[1]$, i.e. $\forall s \in \|X[0]\|_\rho^\perp = \varnothing^\perp = \Lambda_c$, $\forall s' \in \|X[1]\|_\rho^\perp = \{\pi\}^\perp$, $\forall \pi' \in \|X[1]\|_\rho = \{\pi\}$, $\theta \star s'.s.\pi' \in \perp\!\!\!\perp$. But since $t \in \{\pi\}^\perp$, we have $\theta \star v_1.v_2.\pi \in \perp\!\!\!\perp$, which is what we wanted. □

We end here this small introduction to realizability for PA2. But first, let's sum up with the following recipe:

Remark 3.14 (Recipe for computational content of analysis)**.** *In the following chapters we will add a new programming instruction, called σ, which is used to deal with the axiom of dependent choice, which is enough to develop usual analysis. Now, a realizer of a formula is a program written in*

λ_c-calculus + σ which realizes the formula in the sense of realizability. That is to say, it justifies it in the sense that it is a winning strategy as explained earlier.
Now, in order to extract a realizer from a proof of a theorem in analysis you:
- write the formal proof and the formula A corresponding to the proof and the theorem in a second order language for PA2*;*
- "decorate" the formal proof using λ_c-terms + σ adding the term information in it, as given in definition (3.6);
- the final term of type A is the wanted realizer.

For the seek of completeness it must be said that λ_c-calculus is not the only way of extending the Curry-Howard paradigm to classical logic. There are in fact other possible systems, such as Parigot's $\lambda\mu$-calculus. The point is that λ_c-calculus is well-suited for a realizability framework, allowing an interactive interpretation which induces Tarski models, linked with forcing and providing not only computational content for proofs, but also for axioms, as we will see for a specific example.

3.2 Realizability algebras and interpretations

We introduce the general notion of *realizability algebra*. In the following sections we will define a family of realizability algebras $SR_1(\mathrm{P})$ parametrized by semi-lattices P called "forcing structures" - that correspond to the forcing transformation from a syntactical point of view - and study the relations with the standard algebra SR_0. Finally, we will choose a particular forcing structure, and we will apply to the associated algebra the results that we will have found, in order to realize some formulas that express in our language some properties. At the end we obtain that with a proof of a theorem in analysis (= PA2 + axiom of dependent choice) that uses the existence of a non trivial (also selective) ultrafilter on $\mathbb{N}$, we can associate a program written in λ_c-calculus that justifies the theorem, in the sense that it realizes it in SR_0.

The following definition is the generalized notion of "program" and "execution" that we will deal with. Note that now, contrary to usual λ-calculus, we are no more dealing with words on alphabets but with abstract axiomatic structures.

Definition 3.15 (Realizability algebras)**.** *A realizability algebra is 9-ple* $SR = (\Lambda,\ \Pi,\ \Lambda \star \Pi,\ cons,\ cont,\ proc,\ \mathscr{S},\ sub,\ \perp\!\!\!\perp)$*, where:*

- $\Lambda,\ \Pi,\ \Lambda \star \Pi$ *are sets, called set of terms, set of stack, set of processes;*
- $cons : \Lambda \times \Pi \to \Pi$, *and we write* $t.\pi$ *instead of* $cons(t, \pi)$*;*
- $cont : \Pi \to \Lambda$, *and we write* κ_π *instead of* $cont(\pi)$*;*
- $proc : \Lambda \times \Pi \to \Lambda \star \Pi$, *and we write* $t \star \pi$ *instead of* $proc(t, \pi)$*;*
- $\mathscr{S} \subseteq \mathscr{P}_{fin}(\mathrm{Var} \times \Lambda)$ *s.t.* $\varnothing \in \mathscr{S}$ *and if* $\{(x, \xi), (x_1, \xi_1), \ldots, (x_n, \xi_n)\} \in \mathscr{S}$ *then* $\{(x_1, \xi_1), \ldots, (x_n, \xi_n)\} \in \mathscr{S}$*. We call* substitution *an element of* $\mathscr{S}$*, and we write* $\{\overrightarrow{(x, \xi)}\}$ *instead of* $\{(x_1, \xi_1), \ldots, (x_n, \xi_n)\}$ *when we don't want to specify the integer* n*;*

- $sub : \mathscr{S} \times QP \to \Lambda$. *We write* $t[\vec{\xi}/\vec{x}]$ *instead of* $sub(\{\overrightarrow{(x,\xi)}\}, t)$ *and* $t[]$ *instead of* $sub(\varnothing, t)$*;*
- $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$, *called pole, s.t. if we define the relation* $\succ$ *on* $\Lambda \star \Pi$ *by:* $p \succ p'$ *iff* $p' \in \perp\!\!\!\perp \Rightarrow p \in \perp\!\!\!\perp$ *and if we define the relation* $\leq$ *on* Λ *by:*
  $\xi \leq \eta$ *iff* $\forall \pi \in \Pi, \xi \star \pi \succ \eta \star \pi$,
  then $\forall \pi, \rho \in \Pi,\ \forall \{\overrightarrow{(x,\xi)}\} \in \mathscr{S},\ \forall t, u \in QP$, *we have:*

(var) $x_i[\vec{\xi}/\vec{x}] \star \pi \succ \xi_i \star \pi$, *for* x_i *variable, if* $\{\overrightarrow{(x,\xi)}\} \neq \varnothing$

(push) *if* $\forall \xi_i' \leq \xi_i$ *s.t.* $\{\overrightarrow{(x,\xi')}\} \in \mathscr{S}$, $t[\vec{\xi'}/\vec{x}] \star u[\vec{\xi'}/\vec{x}].\pi \in \perp\!\!\!\perp$, *then* $(t)u\ [\vec{\xi}/\vec{x}] \star \pi \in \perp\!\!\!\perp$

(grab) *if* $\forall \xi_i' \leq \xi_i, \eta' \leq \eta$ *s.t.* $\{\overrightarrow{(x,\xi')},\ (y,\eta')\} \in \mathscr{S}$, $t[\eta'/y, \vec{\xi'}/\vec{x}] \star \pi \in \perp\!\!\!\perp$,
then $\lambda y t\ [\vec{\xi}/\vec{x}] \star \eta.\pi \in \perp\!\!\!\perp$

(save) $\mathrm{callcc}[\vec{\xi}/\vec{x}] \star \eta.\pi \succ \eta \star \kappa_\pi.\pi$

(restore) $\kappa_\pi \star \eta.\rho \succ \eta \star \pi$.

Given a pole $\perp\!\!\!\perp$, *we define the orthogonal of a set of* λ_c*-terms and of a set of stacks as follows:*
$V^\perp := \{\pi \in \Pi\ s.t.\ \forall \xi \in V,\ \xi \star \pi \in \perp\!\!\!\perp\}$, *for* $V \subseteq \Lambda$,
$U^\perp := \{\xi \in \Lambda\ s.t.\ \forall \pi \in U,\ \xi \star \pi \in \perp\!\!\!\perp\}$, *for* $U \subseteq \Pi$.

We note by $[\mathscr{S}]t$ *the set* $\{t[\vec{\xi}/\vec{x}]\}_{\xi_i \in \Lambda} \subseteq \Lambda$.

Remark 3.16. *We can easily check that:*

(i) $\forall \{\overrightarrow{(x,\xi)}\} \in \mathscr{S},\ \forall \xi_i' \leq \xi_i,\ \forall \eta_j' \leq \eta_j\ s.t. \{\overrightarrow{(x,\xi')},\ \overrightarrow{(y,\eta')}\} \in \mathscr{S}$,
$\lambda y_1, \ldots, \lambda y_n t\ [\vec{\xi}/\vec{x}] \star \eta_1.\ldots.\eta_n.\pi \succ t[\vec{\xi'}/\vec{x}, \vec{\eta'}/\vec{y}] \star \pi$

(ii) $\leq$ *is a preorder on* Λ.

Example 3.17. *The previous section gives us a first realizability algebra, the one obtained taking as set of terms the set* Λ_c*, as stacks the set* Π_c*, set of processes* $\Lambda_c \times \Pi_c$, $cons$, $cont$ *and* $proc$ *as defined in §1.2,* $\mathscr{S}$ *and* sub *the usual substitutions for the* λ_c*-terms, and a* $\perp\!\!\!\perp \succ$*-saturated. We call it the standard realizability algebra and we note it by* SR_0.

Notation 3.18. *Given two sets* Π *and* M*, we note by* $A[a/x]$ *and* $A[\mathcal{H}/X]$ *respectively the (formal) substitution of the individual variable* x *by* $a \in M$ *in the formula* A *and the (formal) substitution of the second order variable* X^k *by* $\mathcal{H} \in \mathscr{P}(\Pi)^{M^k}$ *in the formula* A*. We call* a *and* $\mathcal{H}$ *parameters and we call* $A[a/x]$ *and* $A[\mathcal{H}/X]$ *formulas with parameters. We will sometimes call* $\mathcal{H}$ *a predicate. We say that a formula with parameters is closed iff each one of her variables (individuals or second order) is replaced by a parameter.*
Given a set of formulas $\mathcal{F}$ *in a second order language, we note with* $\mathcal{F}_{par}$ *the set of formulas with parameters (we omit the dependence from* M *and* Π*, because there won't be any confusion) and with* $\mathcal{F}_{par}^{close}$ *the set of closed formulas with parameters. We note that the formulas with parameters have the same inductive syntax of the formulas.*

We can follow the exact same scheme to define realizability in this framework.

Definition 3.19 (realizability interpretation)**.** *Let* SR *be a realizability algebra with stacks* Π *and let* $\mathcal{F}$ *be a set of formulas in a second order language with set of function symbols* Fonct *and set of expressions* $\mathcal{T}$*.*
A SR*-interpretation for* $\mathcal{F}$ *is a triplet* $\mathscr{I}_{SR} = (M,\ [\![.]\!],\ \overline{(.)})$*, where:*

- $M \neq \varnothing$ *called the set of individuals;*
- $[\![.]\!] : \mathcal{T} \cup \mathrm{Fonct} \to M$ *s.t.* $[\![f(\vec{e})]\!] = [\![f]\!]([\![\vec{e}]\!])$*;*
- $\overline{(.)} : \{\text{second order variables}\} \to \bigcup_{n\in\mathbb{N}} \mathscr{P}(\Pi)^{M^n}$ *s.t.* $\overline{X^k} : M^k \to \mathscr{P}(\Pi)$.

Given a SR*-interpretation* $\mathscr{I}$*, we set* $A\{\vec{x}, \vec{X}\}^{\mathscr{I}} := A[[\![\vec{x}]\!]/\vec{x}, \overrightarrow{\overline{X}}/\vec{X}] \in \mathcal{F}_{par}^{close}$*; it is well defined a function:* $\|.\| : \mathcal{F}_{par}^{close} \to \mathscr{P}(\Pi)$ *by:*

$$\|X(\vec{e})^{\mathscr{I}}\| := \overline{X}([\![\vec{e}]\!])$$

$$\|(A \to B)^{\mathscr{I}}\| := \|A^{\mathscr{I}}\|^{\perp}.\|B^{\mathscr{I}}\|$$

$$\|(\forall x A)^{\mathscr{I}}\| := \bigcup_{a\in M} \|A[a/x]^{\mathscr{I}}\|$$

$$\|(\forall X^k A)^{\mathscr{I}}\| := \bigcup_{\mathcal{H}\in\mathscr{P}(\Pi)^{M^k}} \|A[\mathcal{H}/X]^{\mathscr{I}}\|.$$

We set $|A^{\mathscr{I}}| := \|A^{\mathscr{I}}\|^{\perp}$ *and we call it the set of* realizers *(or truth values) of* A*.*
We write $t \Vdash A^{\mathscr{I}}$ *instead of* $t \in |A^{\mathscr{I}}|$*.*
For a closed formula A*the definitions don't depend on* $(.)^{\mathscr{I}}$*.*

Remark 3.20. *One can notice the similarity between the definition of* SR*-interpretation and the one of Tarki models* $\mathfrak{M}$*, which is obtained by taking* $\{0,1\}$ *instead of* $\mathscr{P}(\Pi)$ *(and setting* $\overline{b} = 1 - b,\ \text{for } b = 0,1$*, fixing the condition for the implication and setting* $\mathfrak{M} \vDash A\ iff\ |A^{\mathscr{I}}| = 1$*).*

Example 3.21. *The previous section gives us a* SR_0*-interpretation for* $\mathrm{PA2}$*, which is obtained taking* $M = \mathbb{N}$*. We call that the* standard SR*-interpretation for* $\mathrm{PA2}$*.*

In all that follows we will use the *standard* interpretation or some of its extensions. We suppose now that we have fixed a realizability algebra SR.

Notation 3.22. *Given* $U, V \subseteq \Pi$*, we set* $U \to V := U^{\perp}.V$ *and* $\neg U := U \to \perp = U^{\perp}.\Pi$*. Given* $L \subseteq \Lambda$ *we set* $L \to U := L.U \subseteq \Pi$*.*
Given a term t*, a formula* A *and* $L \subseteq \Lambda$ *we note* $\|L \to A\| := L \to \|A\|$ *and we shall write from now* $t \Vdash L \to A$ *in order to indicate that* $t \in \|L \to A\|^{\perp}$*.*

Lemma 3.23. *Let* $\pi \in U \subseteq \Pi$*. Then* $\forall V \subset \Pi$*,* $\kappa_\pi \in (U \to V)^{\perp}$*.*
So $\forall A, B$ *formulas with parameters,* $\forall \pi \in \|A\|$*,* $\kappa_\pi \Vdash A \to B$*, and in particular* $\kappa_\pi \Vdash \neg A$*.*

Proof. Let $\rho \in V$ and $\xi \in U^\perp$. Hence $\kappa_\pi \star \xi.\rho \succ \xi \star \pi \in \perp\!\!\!\perp$. □

The following result is the basic fundamental result of all the realizability theory, saying that realizability is compatible with deduction in NK2.

Theorem 3.24 (Adequation lemma)**.** *Let $A\{\vec{y},\vec{Y}\}, A_i\{\vec{y},\vec{Y}\}$, $i = 1,\dots,k$ formulas s.t. $x_1 : A_1,\dots,x_k : A_k \vdash t : A$ and let $\{\overrightarrow{(x,\xi)}\} \in \mathscr{S}$. Then:*

$$\xi_i \Vdash A_i[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}],\ i = 1,\dots,k \quad \Rightarrow \quad t[\vec{\xi}/\vec{x}] \Vdash A[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$$

where $a_i \in \mathbb{N}$, and $\mathcal{H}_i \in \mathscr{P}(\Pi)^{\mathbb{N}^{k_i}}$ (where k_i is the arity of Y_i).
Therefore in particular: $\vdash t : A \ \Rightarrow\ \forall \varsigma \in \mathscr{S},\ t[\varsigma] \Vdash A$. In this case we write $[\mathscr{S}]t \Vdash A$.

Proof. By induction on a derivation of $x_1 : A_1,\dots,x_k : A_k \vdash t : A$. Let R be its last rule.

$R = ax$: then $t = x_j$, so $t[\vec{\xi}/\vec{x}] \star \pi \succ \xi_j \star \pi$ thanks to (var), and $A = A_j$, from which the result for $\pi \in \|A[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]\|$.

$R = l.p.$: then $t = \texttt{callcc}$ and $A = ((B \to C) \to B) \to B$.
We want to prove that: $\forall \eta \Vdash (B \to C) \to B$, $\forall \pi \in \|B\|$, $\texttt{callcc}[\vec{\xi}/\vec{x}] \star \eta.\pi \in \perp\!\!\!\perp$.
Since $\texttt{callcc}[\vec{\xi}/\vec{x}] \star \eta.\pi \succ \eta \star \kappa_\pi.\pi$, and since the lemma 3.23 gives us $\kappa_\pi \Vdash B \to C$, we obtain the result from thanks to the hypothesis on η.

$R = @$: then $t = (u)v$, $x_1 : A_1,\dots,x_k : A_k \vdash u : B \to A$ and $x_1 : A_1,\dots,x_k : A_k \vdash v : B$;
thanks to (push) it is enough to show that:
$\forall \xi'_i \le \xi_i$ s.t. $\{\overrightarrow{(x,\xi')}\} \in \mathscr{S}$, $\forall \pi \in \|A[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]\|$, $u[\vec{\xi'}/\vec{x}] \star v[\vec{\xi'}/\vec{x}].\pi \in \perp\!\!\!\perp$.
Now, it is a trivial fact that if $\xi'_i \le \xi_i \Vdash A_i[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$ then $\xi'_i \Vdash A_i[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, and so by the induction hypothesis we have $v[\vec{\xi'}/\vec{x}] \Vdash B[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, and hence $v[\vec{\xi'}/\vec{x}].\pi \in \|B[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}] \to A[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]\|$. But again by inductions hypothesis we have also $u[\vec{\xi'}/\vec{x}] \Vdash B[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}] \to A[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, and finally the result.

$R = \lambda$: then $t = \lambda z u$, $A = B \to C$ and $x_1 : A_1,\dots,x_k : A_k, z : B \vdash u : C$.
Thanks to (grab) it is enough to show that:
$\forall \eta \Vdash B[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, $\forall \pi \in \|C[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]\|$,$\forall \xi'_i \le \xi_i, \eta' \le \eta$ s.t. $\{\overrightarrow{(x,\xi')},\ (z,\eta')\} \in \mathscr{S}$, $u[\eta'/z,\vec{\xi'}/\vec{x}] \star \pi \in \perp\!\!\!\perp$.
Since $\eta' \le \eta \Vdash B[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, we have $\eta' \Vdash B[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, and similarly $\xi'_i \Vdash A_i[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$. Therefore by induction hypothesis we have $u[\eta'/z,\vec{\xi'}/\vec{x}] \Vdash C[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, from which the result.

$R = \forall_i$: then $A = \forall x B$ with x not free in $A_1,\dots,A_k$, and $x_1 : A_1,\dots,x_k : A_k \vdash t : B$.
We have to show that: $t[\vec{\xi}/\vec{x}] \Vdash A[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, i.e. $t[\vec{\xi}/\vec{x}] \Vdash B[b/x,\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, $\forall b \in \mathbb{N}$.
But since x is not free in the A_i, then $\|A_i[b/x,\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]\| = \|A_i[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]\|$, and since by hypothesis $\xi_i \Vdash A_i[\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$, we have also $\xi_i \Vdash A_i[b/x,\vec{a}/\vec{y},\vec{\mathcal{H}}/\vec{Y}]$. We finally obtain the result using the induction hypothesis.

$R = \forall^2_i$: like the case $R = \forall_i$.

$R = \forall_i^2$: then $A = B\{C/X\vec{z}\}$ and $x_1 : A_1, \ldots, x_k : A_k \vdash t : \forall X^m B$.
We have to show that: $t[\vec{\xi}/\vec{x}] \Vdash B\{C/X\vec{z}\}[\vec{a}/\vec{y}, \vec{\mathcal{H}}/\vec{Y}]$.
The induction hypothesis gives us: $t[\vec{\xi}/\vec{x}] \Vdash (\forall XB)[\vec{a}/\vec{y}, \vec{\mathcal{H}}/\vec{Y}]$,
i.e. $t[\vec{\xi}/\vec{x}] \Vdash B[\mathcal{Y}/X][\vec{a}/\vec{y}, \vec{\mathcal{H}}/\vec{Y}]$, $\forall \mathcal{Y} \in \mathscr{P}(\Pi)^{\mathbb{N}^m}$.
The result follows then from the following general fact: $\|B\{C/X^m\vec{z}\}[\vec{a}/\vec{y}, \vec{\mathcal{H}}/\vec{Y}]\| = \|B[\mathcal{C}/X][\vec{a}/\vec{y}, \vec{\mathcal{H}}/\vec{Y}]\|$, where $\mathcal{C} : \mathbb{N}^m \to \mathscr{P}(\Pi)$, $\mathcal{C}(\vec{n}) := \|C[\vec{n}/\vec{x}][\vec{a}/\vec{y}, \vec{\mathcal{H}}/\vec{Y}]\|$, which can be easily shown by induction.

$R = \forall_e$: like the case $R = \forall_i^2$.

□

Remark 3.25. *The adequation lemma says that the notion of realizability is compatible with that of provability and it has important consequences:*
let SR be a realizability algebra with $\mathscr{S} = \mathscr{P}_{fin}(Var \times \Lambda)$ and let $\mathscr{I}$ be a SR-interpretation for a second order language with set formulas $\mathcal{F}$. We can consider the second order theory
$T_{\mathscr{I}} := \{A \in \mathcal{F} \ s.t. \ \exists \theta \in QP, \exists s \in \mathscr{S} \ s.t. \ \theta[s] \Vdash A^{\mathscr{I}}\}$.
The adequation lemma gives us that this theory is deductively closed, i.e.:
$T_{\mathscr{I}} \vdash B \Rightarrow B \in T_{\mathscr{I}}$.
And more, suppose the pole $\perp\!\!\!\perp$ of SR is coherent, i.e. $\nexists \theta \in QP \ s.t. \ \exists s \in \mathscr{S} \ s.t. \ \theta[s] \Vdash \bot$. Then, since she is deductively closed, we immediately obtain that $T_{\mathscr{I}}$ is coherent, i.e. $T_{\mathscr{I}} \nvdash \bot$.
Sine she is coherent, by the completeness theorem $T_{\mathscr{I}}$ admits Tarski models (with a "non full" semantics). We call those models realizability models *for $\mathscr{I}$.*

Proposition 3.26. *Let $\neq$: $\mathbb{N}^2 \to \mathscr{P}(\Pi)$ be the predicate defined by $\neq(n, n) := \Pi$ and $\neq(n, m) := \varnothing$ if n and m are different. We write $n \neq m$ instead of $\neq(n, m)$. Then we have:*

1 $[\mathscr{S}]\lambda x(x)\lambda xx \Vdash \forall x \forall y(\neg(x = y) \to x \neq y)$

2 $[\mathscr{S}]\lambda x\lambda y(y)x \Vdash \forall x \forall y(x \neq y \to \neg(x = y))$.

Proof. 1: Let $m, n \in \mathbb{N}$. We have to show that $\lambda x(x)\lambda xx \ [\varsigma] \Vdash (m = n \to \bot) \to m \neq n$, for $\varsigma \in \mathscr{S}$. There are two cases:
if $m \neq n$ then $\|(m = n \to \bot) \to m \neq n\| = |m = n \to \bot|.\varnothing = \varnothing$,
and so $|(m = n \to \bot) \to m \neq n| = \Lambda \ni \lambda x(x)\lambda xx \ [\varsigma]$;
if $m = n$ then, since $\|\bot\| = \Pi$, $\|(m = n \to \bot) \to m \neq n\| = \|(m = m \to \bot) \to \bot\|$, and since one can easily check that $\vdash \lambda x(x)\lambda xx : (m = m \to \bot) \to \bot$, we obtain the result by the adequation lemma.

2: similar to 1.

□

Remark 3.27. *The results as the one we have just proved tells us that*
$\mathcal{T}_{\mathscr{I}} \ni \forall x \forall y(\neg(x = y) \leftrightarrow x \neq y)$, and so in order to prove that a realizability model $\mathfrak{M}$ satisfy $\neg(x = y)$ it is necessary and sufficient realizing $(x \neq y)^{\mathscr{I}}$, which is in general easier to realize than $\neg(x = y)^{\mathscr{I}}$.

Notation 3.28. *From now on we suppose that in the language that we use there is a constant symbol (function of arity 0)* 0 *and a symbol of unary function* $succ$ *that are interpreted, in the realizability interpretation over* $\mathbb{N}$ *that we suppose fixed, respectively by* $0 \in \mathbb{N}$ *and by the successor function on* $\mathbb{N}$.
Recall the formula $Nat\{x\} := \forall X^1((\forall y(Xy \to Xsucc(y))) \to X0 \to Xx)$.

Notation 3.29. *1 Given a sequence* $s = (u_i)_{i\in\mathbb{N}}$ *in* Λ, *we note with* $T_s, S_s \in \Lambda$, *when the exist, two* λ_c*-terms s.t.* $T_s \star \phi.\nu.\pi \succ \nu \star S_s.\phi.u_0.\pi$ *and* $S_s \star \psi.u_n.\pi \succ \psi \star u_{n+1}.\pi$

2 Recall the representation of arithmetic in λ*-calculus via the Church numerals*

$$\mathtt{n} = \lambda f\lambda x(f)^n x \in QP \text{ and } \mathtt{succ} = \lambda n\lambda f\lambda x((n)f)(f)x \in QP$$

.

Theorem 3.30. *Let* $s = (u_i)_{i\in\mathbb{N}}$ *be a sequence in* Λ *and suppose that* $\exists T_s, S_s \in \Lambda$.
Then: $T_s \Vdash \forall n\forall X^0((\{u_n\} \to X) \to Nat\{n\} \to X)$

Proof. Let $n \in \mathbb{N}$, $\mathcal{H} \subseteq \mathscr{P}(\Pi)$, $\phi \Vdash \{u_n\} \to \mathcal{H}$, $\nu \Vdash Nat\{n\}$ and $\pi \in \mathcal{H}$; by the reduction rule of T_s it is enough to show that $\nu \star S_s.\phi.u_0.\pi \in \perp\!\!\!\perp$.
Since $\nu \Vdash Nat\{n\}$, it is enough to show that $\exists \mathcal{Y}^1 : \mathbb{N} \to \mathscr{P}(\Pi)$ s.t. $S_s \Vdash \forall y(\mathcal{Y}(y) \to \mathcal{Y}(succ(y))$, $\phi \Vdash \mathcal{Y}(0)$ and $u_0.\pi \in \|\mathcal{Y}(n)\|$. Let's check that $\mathcal{Y}(i) = \begin{cases} \{u_{n-i}.\pi\}, & 0 \le i \le n \\ \varnothing, & i > n \end{cases}$ suits:
by definition of $\mathcal{Y}$, $u_0.\pi \in \mathcal{Y}(n) = \|\mathcal{Y}(n)\|$, and by hypothesis on ϕ, $\phi \Vdash \mathcal{Y}(0)$; now, let $i \in \mathbb{N}$:
if $i \ge n$ then $\|\mathcal{Y}(succ(i))\| = \varnothing$ and so $|\mathcal{Y}i \to \mathcal{Y}succ(i)| = \Lambda \ni S_s$;
if $i < n$, then $\forall \psi \Vdash \mathcal{Y}i$, $S_s \star \psi.u_{n-i-1}.\pi \succ \psi \star u_{n-i}.\pi \in \perp\!\!\!\perp$.
Hence in every case $S_s \Vdash \mathcal{Y}i \to \mathcal{Y}succ(i)$, which ends the proof. □

Corollary 3.31 (Storage operators for the integers)**.** *Consider the sequence* $N := \{(\mathtt{succ})^n \mathtt{0}\ []\}_{n\in\mathbb{N}} \subseteq \Lambda$ *and suppose that in* Λ *they exist* T_N *and* S_N. *We have:*

i): $T_N \Vdash \forall X^1((\forall n(\{(\mathtt{succ})^n \mathtt{0}\ []\} \to Xn)) \to \forall n(Nat\{n\} \to Xn))$

ii): $\lambda xx\ [] \Vdash \forall X^1((\forall n(Nat\{n\} \to Xn)) \to \forall n(\{(\mathtt{succ})^n \mathtt{0}\ []\} \to Xn))$

Proof. The result follows from the fact that one can easily prove, using theorem 3.30 and the adequation lemma, that $T_N \Vdash \forall X^0\forall n((\{(\mathtt{succ})^n \mathtt{0}\ []\} \to X) \to (Nat\{n\} \to X))$ and $\lambda xx\ [] \Vdash \forall X^0\forall n((Nat\{n\} \to X) \to (\{(\mathtt{succ})^n \mathtt{0}\ []\} \to X))$. □

Remark 3.32. *The meaning of the theorems like the one that we've just proved is that, from the point of view of the realization of formulas, the formulas* $\forall n(Nat\{n\} \to Xn)$ *and* $\forall n(\{(\mathtt{succ})^n \mathtt{0}\ []\} \to Xn))$ *are interchangeable, in the sense that:*
if $h \Vdash \forall n(\{(\mathtt{succ})^n \mathtt{0}\ []\} \to Xn))$ *then* $(x)y\ [T_N/x, h/y] \Vdash \forall n(Nat\{n\} \to Xn)$ *(and a similar inverse implication), and so one can realize one iff one can realize the other.*

Theorem 3.33. *Let's suppose that* $\forall \xi \in \Lambda$, $\forall \pi \in \Pi$, *there exist in* Λ *two terms, denoted by* (ξ, π) *and* V, *s.t.* $(\xi, \pi) \star \varpi \succ \xi \star \kappa_\pi.\varpi$ *and* $V \star \xi.\eta.\pi \succ \eta.(\xi, \pi).\pi$.
Then $\forall \mathscr{H}, \mathscr{Y} \subseteq \Pi$, *if we set* $K_{\mathscr{H}} := \{\kappa_\pi\}_{\pi\in\mathscr{H}}$, *we have:*

i): $[\mathscr{S}]\lambda xx \Vdash (\neg\mathscr{H} \to \mathscr{Y}) \to (K_{\mathscr{H}} \to \mathscr{Y})$

ii): $V \Vdash (K_{\mathscr{H}} \to \mathscr{Y}) \to ((\mathscr{Y} \to \mathscr{H}) \to \mathscr{H})$

Proof. Easy. □

3.3 The forcing theory in the realizability

We will not talk about the forcing theory in set theory ZF. That is for several reasons: firstly, we would need at least an entire course on ZF in order to approach it, and this is not a thesis on set theory; secondly, what follows is technically completely independent from set theoretic forcing theory. The link between the two is in the development of the tools that we will construct and use: many of them are inspired from techniques in set theory, and particularly forcing.
What is however important to say about forcing is just its very general idea, which we resume here:
in a given model of ZF we fix a partially ordered set of "forcing conditions" and a filter G of "good conditions" among them, and we then define the "forcing relation" $p \Vdash A$ (read[46] "p forces A"), between forcing conditions p and formulas A. Cohen shows how to construct a new model $\mathfrak{M}[G]$ of ZF with the property that a formula A is true in that model iff $\exists p \in G$ s.t. $p \Vdash A$. Now, with an intelligent choice of the forcing conditions, Cohen was able to "force" some interesting formulas to be true in the model $\mathfrak{M}[G]$, such as for example the negation of the axiom of choice, or the negation of the continuum hypothesis. That proves that the axiom of choice, or the continuum hypothesis, are not consequences of ZF, since there are models of ZF in which those formulas are false.
Together with the results of Gödel which proved that neither the negation of those formulas were consequences of ZF (a result obtained using a whole other technique, that of the "internal models"), one obtains the *undecidability* of them, which is one of the most significant theorems of set theory and of modern mathematics in general (we recall that for example the continuum hypothesis was the problem number 1 in the famous list of "23 unsolved[47] problems of mathematics" proposed by Hilbert in 1900).
The present work has nothing to do with the undecidability of those formulas, but as one will be able to remark we will follow the ideas of forcing in our constructions, which are in fact a syntactic treatment of it.

3.3.1 The algebras SR_0 and SR_1

Notation 3.34 (The algebra SR_0)**.** *We recall that in the example 2.3 we defined a realizability algebra, called SR_0, which was constitute on the λ_c-calculus.*
We will consider from now on the same algebra, but with set of terms the Λ_c extended with three new instructions σ, χ and χ':
we use the following notation: if $\pi = \xi_1.\ldots.\xi_n.\alpha$ (with α constant of end of stack), then

[46] Note the use of the same notation in realizability.

[47] Note that this undecidability result does not mean the resolution of the problem, which is indeed still an important field of research in set theory. See for example the work of the American mathematician W.H. Woodin "The continuum hypothesis. Part I" and "Part II", Notices of the American Mathematical Society, 2001.

$\pi^\tau := \xi_1.\ldots.\xi_n.\tau.\alpha$.
The new three instructions have the following reduction rules:

(read) $\chi \star \xi.\pi^\tau \succ \xi \star \tau.\pi$

(write) $\chi' \star \xi.\tau.\pi \succ \xi \star \pi^\tau$

(clock/signature) [48] $\sigma \star \xi.\pi \succ \xi \star (\mathtt{succ})^{j_\pi}0.\pi$, *where* $\pi \to j_\pi$ *is a fixed bijection between* Π *and* $\mathbb{N}$. *(This instruction is linked with the axiom of choice and we will use it only later).*

We will keep calling Λ_c *the extended set and* SR_0 *the realizability algebra, since there won't be any confusion.*

Remark 3.35. *With the notation 2.15, in* Λ_c *there are the integer storage instructions* $T_{\{(\mathtt{succ})^n 0\}_{n\in\mathbb{N}}}$ *and* $S_{\{(\mathtt{succ})^n 0\}_{n\in\mathbb{N}}}$, *that we will simply call* T_0 *and* S_0. *In fact, one can easily check that for example the following terms work:*
$S_0 := \lambda g\lambda x(g)(\mathtt{succ})x$ *and* $T_0 := \lambda f\lambda n(n)S_0 f 0$.

We will define an other realizability algebra, for which we need the following notion:

Definition 3.36 (Forcing structure)**.** *A forcing structure is a quadruple* $(P, C, 1, \cdot)$, *where:*

- P *is a set, called the set of* forcing conditions
- $C : P \to \mathscr{P}(\Lambda_c)$
- $1 \in P$ *is a fixed element of* P
- $\cdot : P \times P \to P$, *and we will often write* $p \cdot q$ *as* pq

s.t. there exists $\alpha_0, \ldots, \alpha_4 \in QP$, *called* forcing combinators, *s.t.* $\forall \tau \in \Lambda_c$, $\forall p, q \in P$

$\tau \in C(pq) \Rightarrow (\alpha_0)\tau \in C(p)$

$\tau \in C(pq) \Rightarrow (\alpha_1)\tau \in C(qp)$

$\tau \in C(p) \Rightarrow (\alpha_2)\tau \in C(pp)$

$\tau \in C(p(qr)) \Rightarrow (\alpha_3)\tau \in C((pq)r)$

$\tau \in C(p) \Rightarrow (\alpha_4)\tau \in C(p1)$.

[48] The name of the reduction rule indicates that this rule can be interpreted as a clock instruction or a signature instruction of the usual computers. See [Kriv04] for details. Conversely, for the justification of the name (read) and (write) for the reduction rule of other two new instructions, you will have to wait some more pages.

When talking about a forcing structure $\alpha_0, \ldots, \alpha_4$ will always denote its forcing combinators. We call condition terms *the expressions t^P built by the syntax: $t^P ::= p \mid t^P \cdot t^P$, where $p \in P$.*

Notation 3.37. *We call* compound of the α_i *the words γ of the form $\gamma = (\alpha_{i_1}) \ldots (\alpha_{i_k})$. Those words are not λ_c-terms, but $\gamma\xi \in \Lambda_c$ for all $\xi \in \Lambda_c$.*
If β is a compound or a λ_c-term s.t. $\forall \tau \in \Lambda_c$, $\forall \vec{p} \in P$, $\tau \in C(t^P(\vec{p})) \Rightarrow (\beta)\tau \in C(u^P(\vec{p}))$, where t^P, u^P are condition terms, we will often indicate that fact by the notation: $t^P(\vec{p}) \xrightarrow{\beta} u^P(\vec{p})$.

Remark 3.38. *1): There are 6 compounds $\gamma_0, \ldots, \gamma_5$ s.t.*

$$pq \xrightarrow{\gamma_0} p(pq)$$
$$(pq)r \xrightarrow{\gamma_1} pr$$
$$(pq)r \xrightarrow{\gamma_2} qr$$
$$p(qr) \xrightarrow{\gamma_3} q(rr)$$
$$p(qr) \xrightarrow{\gamma_4} qp$$
$$(pq)r \xrightarrow{\gamma_5} p(qr).$$

In fact we have:
$pq \xrightarrow{\alpha_2} (pq)(pq) \xrightarrow{\alpha_3} ((pq)p)q \xrightarrow{\alpha_0} (pq)p \xrightarrow{\alpha_1} p(pq)$, and so we can set $\gamma_0 := (\alpha_1)(\alpha_0)(\alpha_3)(\alpha_2)$; similarly we can set $\gamma_5 := (\alpha_1)(\alpha_3)(\alpha_1)(\alpha_3)(\alpha_1)$, and in the same way for the others.

2): For every compound γ, there exist $\overline{\gamma} \in \Lambda_c$ s.t. $\overline{\gamma} \star \xi.\pi^\tau \succ \xi \star \pi^{\gamma\tau}$.
We can indeed set for example $\overline{\gamma} := \lambda x(\chi)\lambda y((\chi')x)(\gamma)y$.

Remark 3.39. *1): We set $\mathscr{C} := C^{-1}(\mathscr{P}(\Lambda_c) \setminus \varnothing) = \{p \in P \ s.t.\ C(p) \neq \varnothing\}$.*
If $p \in \mathscr{C}$ we say that p is a non trivial condition; if $p, q \in P$ are s.t. $pq \in \mathscr{C}$ we say that p and q are compatibles conditions.
We define a relation $\sqsubseteq$ on P by: $p \sqsubseteq q$ ssi $\forall r \in P,\ pr \in \mathscr{C} \Rightarrow qr \in \mathscr{C}$.
We can verify that:

i): $(P, \sqsubseteq)$ is a preorder (i.e. $\sqsubseteq$ is reflexive and transitive), of which 1 is a $\max(P, \sqsubseteq)$ and which admits infima $\inf\{p, q\} = pq$.

ii): $p \notin \mathscr{C} \Rightarrow p \sqsubseteq q$ $\forall q \in P$ (c.à.d. $P \setminus \mathscr{C} \subseteq min(P, \sqsubseteq)$).

Therefore $(P, \sqsubseteq)$ is a semi-lattice, bounded if there are trivial conditions.

2): We note by $\sim$ the equivalence relation induced by the preorder $\sqsubseteq$:
$p \sim q$ ssi $p \sqsubseteq q$ and $q \sqsubseteq p$.
We obtain that: $p, q \notin \mathscr{C} \Rightarrow p \sim q$ (the trivial conditions are all equivalent).
And more: $p \in \mathscr{C}$ and $p \sqsubseteq q \Rightarrow q \in \mathscr{C}$.

Definition 3.40 (The realizability algebra $SR_1(P)$)**.** *Let $(P, C, 1, \cdot)$ be a forcing structure. We associate to it a realizability algebra $SR_1(P) = (\Lambda_1,\ \Pi_1,\ \Lambda_1 \star \Pi_1,\ cons_1,\ cont_1,\ proc_1,\ \mathscr{S}_1,\ sub_1,\ \perp\!\!\!\perp_1)$ defined as follows, where we have set $\widehat{k} := \lambda x(\chi)\lambda y(k)((\chi')x)(\gamma_4)y$ for $k \in \Lambda_c$.*

- $\Lambda_1 := \Lambda_c \times P$, $\Pi_1 := \Pi_c \times P$, $\Lambda_1 \star \Pi_1 := (\Lambda_c \star \Pi_c) \times P$

- $(\xi,p).(\pi,q) := (\xi.\pi, pq)$ *(here $\xi.\pi$ is the operation of the algebra SR_0),*
  $\kappa_{(\pi,p)} := (\widehat{\kappa_\pi}, p)$, *and* $(\xi,p)\star(\pi,q) := (\xi\star\pi, pq)$
- $\mathscr{S} := \{\{(x_{i_1},(\xi_{i_1},p)),\ldots,(x_{i_k},(\xi_{i_k},p))\}\ s.t.\ x_i\in \mathrm{Var},\ \xi_i\in\Lambda_c,\ p\in P,\ k\in\mathbb{N}\}$
- *for $t\in QP$ we set:*

  $t[] := (t^*,1)\in\Lambda_1$
  $t[(\vec{\xi},p)/\vec{x}] := (t^*[\vec{\xi}/\vec{x}],p)\in\Lambda_1$ *(here the substitution at right is clearly the one of the algebra SR_0)*

  where we have used the λ_c-terms transformation $(.)^*:\Lambda_c\to\Lambda_c$ defined by:

  $x^* := x$
  $((t)u)^* := (\overline{\gamma_0})t^*u^*$
  $(\lambda y t\{\vec{x}\})^* := (\overline{\alpha_3})\lambda y\ t^*[(\overline{\gamma_2})y/y, (\vec{\overline{\gamma_1}})x/\vec{x}]$
  $\mathtt{callcc}^* := \lambda x(\chi)\lambda y(\mathtt{callcc})\lambda k(((\chi')x)(\gamma_3)y)\widehat{k}$
- $\perp\!\!\!\perp_1 := \{(\xi\star\pi,p)\in\Lambda_1\star\Pi_1\ s.t.\ \forall\tau\in C(p),\ \xi\star\pi^\tau\in\perp\!\!\!\perp_0\}$

We will simply call it SR_1, as there will be no confusion.

Remark 3.41. *We have: $\mathtt{callcc}^*\star\xi.\pi^\tau \succ \xi\star\widehat{k_\pi}.\pi^{(\gamma_3)\tau}$ and $\widehat{k_\pi}\star\xi.\rho^\tau\succ\xi\star\pi^{(\gamma_4)\tau}$.*
In fact: $\mathtt{callcc}^*\star\xi.\pi^\tau = \lambda x(\chi)\lambda y(\mathtt{callcc})\lambda k(((\chi')x)(\gamma_3)y)\widehat{k}\star\xi.\pi^\tau \succ \chi\star\lambda y(\mathtt{callcc})\lambda k(((\chi')\xi)(\gamma_3)y)\widehat{k}.\pi^\tau \succ \lambda y(\mathtt{callcc})\lambda k(((\chi')\xi)(\gamma_3)y)\widehat{k}\star\tau.\pi \succ \mathtt{callcc}\star\lambda k(((\chi')\xi)(\gamma_3)\tau)\widehat{k}.\pi \succ \chi'\star\xi.(\gamma_3)\tau.\widehat{k_\pi}.\pi \succ \xi\star(\widehat{k_\pi}.\pi)^{(\gamma_3)\tau} = \xi\star\widehat{k_\pi}.\pi^{(\gamma_3)\tau}$
and the same for the other.

We just said to have defined a realizability algebra. In order for that to be true we must check that it satisfies the appropriate properties. It is precisely the content of the following proposition.

Proposition 3.42. *$\perp\!\!\!\perp_1$ satisfies the properties (var), (push), (grab), (save), (restore) of the realizability algebras' definition.*

Proof. (var): Trivial since $x_i[\overrightarrow{(\xi,p)}/\vec{x}] = (\xi_i,p)$.

(push): Similar to the (grab) case.

(grab): Suppose that $t[(\eta',p')/y,\overrightarrow{(\xi',p')}/\vec{x}]\star(\pi,r)\in\perp\!\!\!\perp_1$, with $(\eta',p')\le(\eta,q)$ and $(\xi'_i,p')\le(\xi_i,p)$ (the case of the empty substitution is the same); we want to show that: $\lambda y t\ [\overrightarrow{(\xi,p)}/\vec{x}]\star(\eta,q).(\pi,r)\in\perp\!\!\!\perp_1$. Now, $\lambda y t\ [\overrightarrow{(\xi,p)}/\vec{x}] = ((\lambda y t)^*[\vec{\xi}/\vec{x}],p) = ((\overline{\alpha_3})\lambda y\ \underbrace{t^*[(\overline{\gamma_2})y/y,\overrightarrow{(\overline{\gamma_1})x}/\vec{x}]}\ [\vec{\xi}/\vec{x}],p) =$

$((\overline{\alpha_3})\lambda y\ \underbrace{t^*[(\overline{\gamma_2})y/y,\overrightarrow{(\overline{\gamma_1})\xi}/\vec{x}]},p)$, and so we want to prove that

$((\overline{\alpha_3})\lambda y\ t^*[(\overline{\gamma_2})y/y,\overrightarrow{(\overline{\gamma_1})\xi}/\vec{x}]\star\eta.\pi,p(qr))\in\perp\!\!\!\perp_1$. Then, let $\tau\in C(p(qr))$;
since $(\overline{\alpha_3})\lambda y\ t^*[(\overline{\gamma_2})y/y,\overrightarrow{(\overline{\gamma_1})\xi}/\vec{x}]\star\eta.\pi^\tau\succ t^*[(\overline{\gamma_2})\eta/y,\overrightarrow{(\overline{\gamma_1})\xi}/\vec{x}]\star\pi^{(\alpha_3)\tau}$,

it is enough to prove that $t^*[(\overline{\gamma_2})\eta/y, \overrightarrow{(\overline{\gamma_1})\xi}/\vec{x}] \star \pi^{(\alpha_3)\tau} \in \perp\!\!\!\perp_0$.
One can easily prove that: $((\overline{\gamma_1})\xi_i, pq) \leq (\xi_i, p)$ and $((\overline{\gamma_2})\eta, pq) \leq (\eta, q)$ (it's true in general).
By the hypothesis then:
$\perp\!\!\!\perp_1 \ni\ t[((\overline{\gamma_2})\eta, pq)/y, \overrightarrow{((\overline{\gamma_1})\xi, pq)}/\vec{x}] \star (\pi, r) = (t^*[(\overline{\gamma_2})\eta/y, \overrightarrow{(\overline{\gamma_1})\xi}/\vec{x}] \star \pi, (pq)r)$.
Since $(\alpha_3)\tau \in C((pq)r)$, we obtain what we wanted to show.

(save): Suppose $(\xi, q) \star (\widehat{\kappa_\pi}, r).(\pi, r) \in \perp\!\!\!\perp_1$, i.e. $(\xi \star \widehat{\kappa_\pi}.\pi, q(rr)) \in \perp\!\!\!\perp_1$; we want to show that $\texttt{callcc}[\overrightarrow{(\eta, p)}/\vec{x}] \star (\xi, q).(\pi, r) \in \perp\!\!\!\perp_1$, i.e. $(\texttt{callcc}^* \star \xi.\pi, p(qr)) \in \perp\!\!\!\perp_1$ (the case of the empty substitution is the same). Then, let $\tau \in C(p(qr))$, from where $(\gamma_3)\tau \in C(q(rr))$.
But $\texttt{callcc}^* \star \xi.\pi^\tau \succ \xi \star \widehat{\kappa_\pi}.\pi^{(\gamma_3)\tau}$, and $\xi \star \widehat{\kappa_\pi}.\pi^{(\gamma_3)\tau} \in \perp\!\!\!\perp_0$ by the hypothesis.

(restore): Similar to (save).

□

Remark 3.43. *The transformation $(.)^* : \Lambda_c \to \Lambda_c$ that we have just defined is important and it has a very interesting computer science interpretation: we can verify that the execution in the KAM of a program of the form t^* corresponds to the execution of the program t but with the environment that adds in the execution stack (i.e. in the stacks in front of who the program is executed) an area for a global memory that it updates at each reduction step and that cannot be modified by the user (i.e. the program t): it is precisely what is call an execution in a* protected mode*, and we will see that this is linked with the forcing transformations.*
In our framework, this global memory area is the last spot at the end of a stack, just before the end stack constant, and we can then see that the instructions χ and χ' with their reduction rules (read) and (write) operate the extraction of the information from this zone and the update of this information: thus, their naming as (read) and (write).

Proposition 3.44. *1): Let α and β be compounds s.t. $1(p(qr)) \xrightarrow{\alpha} q(1(p(1r)))$ and $1p \xrightarrow{\beta} p$.*
Then the terms $(T, 1), (S, 1) \in \Lambda_1$, with $S := (\overline{\alpha})\lambda g \lambda x (g)(\texttt{succ})x$ and $T := \overline{\beta}\lambda f \lambda n (n) S f \texttt{0}$, are integers storage operators for SR_1, i.e.

$$(T, 1) \star (\phi, p).(\nu, q).(\pi, r) \succ (\nu, q) \star (S, 1).(\phi, p).(\texttt{0}, 1).(\pi, r)$$
$$(S, 1) \star (\psi, p).((\texttt{succ})^n \texttt{0}, 1).(\pi, q) \succ (\psi, p) \star ((\texttt{succ})^{n+1} \texttt{0}, 1).(\pi, q).$$

2): SR_1 satisfies the hypothesis of the theorem **??**, *i.e. there exist a term $(V, 1) \in \Lambda_1$ and for each $(\xi, p) \in \Lambda_1$, $(\pi, q) \in \Pi_1$ there exist a term $((\xi, p), (\pi, q)) \in \Lambda_1$ s.t.*
$((\xi, p), (\pi, q)) \star (\varpi, r) \succ (\xi, p) \star \kappa_{(\pi,q)}.(\varpi, r)$ and $(V, 1) \star (\xi, p).(\eta, q).(\pi, r) \succ (\eta, q) \star ((\xi, p), (\pi, r)).(\pi, r)$.

Proof. 1): simple calculus.

2): We define a binary operation $(.,.)$ on Λ_1 by: $((\xi, p), (\eta, q)) := ((\overline{\gamma_5})\xi\eta, pq) \in \Lambda_c$.
We set $((\xi, p), (\pi, q)) := ((\xi, p), \kappa_{(\pi,q)})$. This works because we show that in general we have: $((\xi, p), (\eta, q)) \star (\pi, r) \succ (\xi, p) \star (\eta, q).(\pi, r)$. In fact: let $(\xi \star \eta.\pi, p(qr)) \in \perp\!\!\!\perp_1$; we want to show that $((\overline{\gamma_5})\xi\eta \star \pi, (pq)r) \in \perp\!\!\!\perp_1$, i.e. that if $\tau \in C((pq)r)$ then $(\overline{\gamma_5})\xi\eta \star \pi^\tau \in \perp\!\!\!\perp_0$. But we have $(\overline{\gamma_5})\xi\eta \star \pi^\tau \succ \xi \star \eta.\pi^{(\overline{\gamma_5})\tau}$, and by hypothesis $\xi \star \eta.\pi^{(\overline{\gamma_5})\tau} \in \perp\!\!\!\perp_0$, considering that $(pq)r \xrightarrow{\gamma_5} p(qr)$.
Finally, let's find V: let's verify that $V := (\chi)\lambda\tau\lambda x\lambda y(\texttt{callcc})\lambda k((\chi')((\overline{\gamma})y)(\overline{\gamma_5})xk')\tau$, where

$k' := \lambda x(\chi)\lambda y(k)((\chi')x)(\gamma_4)y$ and $p(qr) \xrightarrow{\gamma} q((pr)r)$ (and we can find such γ), works: we have to show that if $(\eta \star (\overline{\gamma_5})\xi\widehat{\kappa_\pi}.\pi, q((pr)r)) \in \perp\!\!\!\perp_1$ then $(V \star \xi.\eta.\pi, 1(p(qr))) \in \perp\!\!\!\perp_1$.
For that, let $\tau \in C(1(p(qr)))$; then $(\gamma)\tau \in C(q((pr)r))$ and by hypothesis $\eta\star(\overline{\gamma_5})\xi\widehat{\kappa_\pi}.\pi^{(\gamma)\tau} \in \perp\!\!\!\perp_0$. We want to show that $V \star \xi.\eta.\pi^\tau \in \perp\!\!\!\perp_0$; but since we can calculate that $V \star \xi.\eta.\pi^\tau \succ \eta \star (\overline{\gamma_5})\xi\widehat{\kappa_\pi}.\pi^{(\gamma)\tau}$, we have finished.

□

3.3.2 Realizability interpretation of forcing

We will consider the forcing as a transformation of formulas. In order to define that, we need to extend the language used (and also the notion of model for such a language)).

It is important to note the following disclaimer:
considering models for the languages that we will use is not necessary, and could have been omitted. Indeed, the definition of those models is purely ad hoc, since the aim of the language that we will introduce is not to be interpreted in a model - as it is for the languages in a Tarski semantic perspective. They have to be intended as purely technical tools, their only meaning being to write formulas in them in a way such that a notion of realizability for their formulas is easy definable and manageable. Then, we link the realizability for such ad hoc languages with the realizability for other meaningful languages, such as that for PA2 and we (hopefully!) obtain the properties that we wanted. We decided to include them only because - at least that is what happened to the author while first studying the subject - it can be useful to still have in mind a semantic notion for them, otherwise potentially leaving the reader a bit confused when treating such formulas with a certain idea in mind, but not having a notion of model for them.

Definition 3.45. *Let's fix:*

- *infinitely many variables x called individual variables and also a set of variables p called condition variables;*
- *function symbols f (each one with an arity), among which there is a 0-ary function denoted by 0 and a 1-ary function denoted by $succ$;*
- *a 0-ary function symbol denoted by 1 and a 2-ary function symbol denoted by $\cdot$;*
- *infinitely many second order variables X (each one with an arity), to which we associate new other second order variables X^+ with the same arity of X.*

We define the set of individual terms t by the syntax: $x \mid f(\vec{t})$;
We define the set of condition terms t^P by the syntax: $p \mid t^P \cdot t^P$; when there is no ambiguity we shall write $t^P t^P$ instead of $t^P \cdot t^P$.
We define the set $\mathcal{F}_0$ of formulas A by syntax:

$$t \neq u \mid t^P \neq u^P \mid X(\vec{t}) \mid t^P \not\in X^+(\vec{t}) \mid A \to A \mid \mathscr{C}(t^P) \to A \mid \forall x A \mid \forall^{\mathbb{N}} x A \mid \forall p A \mid \forall X A \mid \forall X^+ A$$

we define $\mathcal{F}_0^{usu} \subseteq \mathcal{F}_0$ of formulas A by the syntax:

$$t \neq u \mid X(\vec{t}) \mid A \to A \mid \forall x A \mid \forall^{\mathbb{N}} x A \mid \forall X A$$

we define $\mathcal{F}_0^{\mathbb{N}} \subseteq \mathcal{F}_0^{usu}$ of formulas A by syntax:

$$t \neq u \mid X(\vec{t}) \mid A \to A \mid \forall^{\mathbb{N}} x A \mid \forall X A$$

we define a set $\mathcal{F}_1$ of formulas A by syntax:

$$t \neq u \mid X(\vec{t}) \mid X^+(\vec{t}) \mid A \to A \mid \forall x A \mid \forall^{\mathbb{N}} x A \mid \forall X A \mid \forall X^+ A$$

we define $\mathcal{F}_1^{\mathbb{N}} \subseteq \mathcal{F}_0$ of formulas A by syntax:

$$t \neq u \mid X(\vec{t}) \mid X^+(\vec{t}) \mid A \to A \mid \forall^{\mathbb{N}} x A \mid \forall X A \mid \forall X^+ A.$$

Notice that $\mathcal{F}_0^{usu} \subseteq \mathcal{F}_1$ and $\mathcal{F}_0^{\mathbb{N}} \subseteq \mathcal{F}_1^{\mathbb{N}}$.

Note that the formulas of $\mathcal{F}_0$ allow to write forcing information directly in the language, while those of $\mathcal{F}_1$ no dot allow to talk about it. We will recover the symmetry in a moment, when interpreting the language in which we cannot talk about forcing in a realizability interpretation which directly deals with forcing informations, and the one which talks about forcing in a realizability interpretation in which there is no forcing information allowed.

Definition 3.46 (Interpretation of terms).

- *An interpretation $\mathbf{C}$ of condition terms on a set P is the data of P, an element $1 \in P$, a function $\cdot_{\mathbf{C}} : P \times P \to P$ and a function $t^P \to t^P_{\mathbf{C}} \in P$ (for t^P condition term) s.t. $(pq)_{\mathbf{C}} = p_{\mathbf{C}} \cdot_{\mathbf{C}} q_{\mathbf{C}}$ and $1_{\mathbf{C}} = 1 \in P$.*
  It is clear that a forcing structure naturally induces an interpretation of condition terms.
- *An interpretation of individual terms on a set M is the data of M, a function $[\![f]\!] : M^k \to M$ for each k-ary function symbol, and $[\![t]\!] \in M$ for each individual term t, s.t. $[\![f(\vec{x})]\!] := [\![f]\!]([\![\vec{x}]\!]) \in M$. Notice that in particular we have an element $[\![0]\!] \in M$ and a function $[\![succ]\!] : M \to M$.*

We fix from now on an interpretation of condition terms and of individual term, and we will omit the dependences, i.e. we write t^P instead of $t^P_{\mathbf{C}}$ for the condition terms and t instead of $[\![t]\!]$ for the individual terms.

The notion of model for the formulas that we have defined is the following:

Definition 3.47. *A model $\mathfrak{M}$ is the data of: an interpretation of condition terms on a set P; a subset $\mathscr{C} \subseteq P$; an interpretation of individual terms on a set M; for each $k \in \mathbb{N}$, the sets $H1_k, H2_k \subseteq \mathscr{P}(M^k)$ and a function $@_k : \mathscr{P}(M^k) \times M^k \to \mathscr{P}(P)$; for each second order variable X^k, the sets $X_{\mathfrak{M}} \in H1_k$, $X^+_{\mathfrak{M}} \in H2_k$.*
For a formula A in $\mathcal{F}_0$ or $\mathcal{F}_1$ we define $\mathfrak{M} \models A$ according to the following conditions:

$\mathfrak{M} \models t \neq u$ *iff* $t_{\mathfrak{M}} \neq u_{\mathfrak{M}}$; $\mathfrak{M} \models t^P \neq u^P$ *iff* $t^P_{\mathfrak{M}} \neq u^P_{\mathfrak{M}}$; $\mathfrak{M} \models X(\vec{t})$ *iff* $(\vec{t_{\mathfrak{M}}}) \in X_{\mathfrak{M}}$; $\mathfrak{M} \models X^+(\vec{t})$ *iff* $(\vec{t_{\mathfrak{M}}}) \in X^+_{\mathfrak{M}}$; $\mathfrak{M} \models t^P \not\in X^+(\vec{t})$ *iff* $t^P_{\mathfrak{M}} \in \mathscr{O}_k(X_{\mathfrak{M}}, \vec{t_{\mathfrak{M}}})$ *(with* k *the arity of* X*)*; $\mathfrak{M} \models B \to C$ *iff* $\mathfrak{M} \models B \Rightarrow \mathfrak{M} \models C$; $\mathfrak{M} \models \mathscr{C}(t^P) \to B$ *iff* $t^P_{\mathfrak{M}} \in \mathscr{C} \Rightarrow \mathfrak{M} \models B$; $\mathfrak{M} \models \forall x B$ *iff* $\forall a \in M,\ \mathfrak{M} \models B[a/x]$; $\mathfrak{M} \models \forall p B$ *iff* $\forall q \in P,\ \mathfrak{M} \models B[q/p]$; $\mathfrak{M} \models \forall^{\mathbb{N}} x B$ *iff* $\mathfrak{M} \models \forall x(Nat\{x\} \to B)$; $\mathfrak{M} \models \forall X^k B$ *iff* $\forall \mathcal{H} \in H1_k,\ \mathfrak{M} \models B[\mathcal{H}/X]$; $\mathfrak{M} \models \forall X^+ B$ *iff* $\forall \mathcal{H} \in H2_k,\ \mathfrak{M} \models B[\mathcal{H}/X^+]$ *(where* k *is the arity of* X*).*

As announced, the important is the notion of realizability for those languages:

Definition 3.48 (Realizability interpretation for $\mathcal{F}_0$ and $\mathcal{F}_1$)**.** *Let* $(P, C, 1, \cdot)$ *be a forcing structure and fix the interpretation of condition terms induced by it (here we will denote that by* $[\![.]\!]$*). Let's fix a realizability algebra* SR_1 *on this forcing structure. Let's fix also an interpretation of individual terms over* $\mathbb{N}$ *(here we will denote that by* $[\![.]\!]$ *too, since there won't be confusion). Finally fix two parameters* $\overline{X} : \mathbb{N}^k \to \mathscr{P}(\Pi_c)$ *and* $\overline{X^+} : \mathbb{N}^k \to \mathscr{P}(\Pi_c \times P)$ *for each second order variable* X^k*. We obtain the notion of parameters formulas, i.e. words of the form* $A[\vec{n}/\vec{x}, \vec{q}/\vec{p}, \vec{\overline{X}}/\vec{X}, \vec{\overline{X^+}}/\vec{X^+}]$*, where* A *is a formula of* $\mathcal{F}_0$ *or of* $\mathcal{F}_1$*. (We recall that the the parameters formulas have the same inductive construction of the formulas). Finally we can then define:*

- *a* SR_0*-realizability interpretation on* $\mathbb{N}$ *for* $\mathcal{F}_0$ *defining* $\|.\|_0 : \mathcal{F}^{close}_{0par} \to \mathscr{P}(\Pi_c)$ *by:*

$$\|[\![t]\!] \neq [\![u]\!]\|_0 := \begin{cases} \varnothing, & [\![t]\!] \neq [\![u]\!] \\ \Pi_c, & \textit{otherwise} \end{cases}$$

$$\|[\![t^P]\!] \neq [\![u^P]\!]\|_0 := \begin{cases} \varnothing, & [\![t^P]\!] \neq [\![u^P]\!] \\ \Pi_c, & \textit{otherwise} \end{cases}$$

$$\|\overline{X}(\vec{[\![t]\!]})\|_0 := \overline{X}(\vec{[\![t]\!]})$$

$$\|[\![t^P]\!] \not\in \overline{X}^+(\vec{[\![t]\!]})\|_0 := \{\pi \in \Pi_c \ s.t.\ (\pi, [\![t^P]\!]) \in \overline{X^+}(\vec{[\![t]\!]})\}$$

$$\|A \to B\|_0 := \|A\|_0^{\perp}.\|B\|_0$$

$$\|\mathscr{C}(t^P) \to B\|_0 := C([\![t^P]\!]).\|B\|_0$$

$$\|\forall x A\|_0 := \bigcup_{n \in \mathbb{N}} \|A[n/x]\|_0$$

$$\|\forall p A\|_0 := \bigcup_{q \in P} \|A[q/p]\|_0$$

$$\|\forall^{\mathbb{N}} x A\|_0 := \bigcup_{n \in \mathbb{N}} \{(\mathtt{succ})^n \mathtt{0}\}.\|A[n/x]\|_0$$

$$\|\forall X^k A\|_0 := \bigcup_{\mathcal{H} \in \mathscr{P}(\Pi_c)^{\mathbb{N}^k}} \|A[\mathcal{H}/X]\|_0$$

$$\|\forall X^+ A\|_0 := \bigcup_{\mathscr{H} \in \mathscr{P}(\Pi_c \times P)^{\mathbb{N}^k}} \|A[\mathscr{H}/X^+]\|_0 \ \textit{(where } X \textit{ has arity } k\textit{)}.$$

- *a SR_1-realizability interpretation on $\mathbb{N}$ for $\mathcal{F}_1$ defining $\|.\|_1 : \mathcal{F}_{1par}^{close} \to \mathscr{P}(\Pi_c \times P)$ by:*

$$\|[\![t]\!] \neq [\![u]\!]\|_1 := \begin{cases} \varnothing, & [\![t]\!] \neq [\![u]\!] \\ \Pi_c \times P, & \textit{otherwise} \end{cases}$$

$$\|\overline{X}(\vec{t})\|_1 := \|\overline{X}(\vec{t})\|_0 \times P = \overline{X}([\![\vec{t}]\!]) \times P$$

$$\|\overline{X^+}(\vec{t})\|_1 := \overline{X^+}([\![\vec{t}]\!])$$

$$\|A \to B\|_1 := \|A\|_1^{\perp}.\|B\|_1$$

$$\|\forall x A\|_1 := \bigcup_{n\in\mathbb{N}} \|A[n/x]\|_1$$

$$\|\forall^{\mathbb{N}} x A\|_1 := \bigcup_{n\in\mathbb{N}} \{((\mathtt{succ})^n \mathtt{0}, 1)\}.\|A[n/x]\|_1$$

$$\|\forall X^k A\|_1 := \bigcup_{\mathcal{H}\in\mathscr{P}(\Pi_c)^{\mathbb{N}^k}} \|A[\mathcal{H}/X]\|_1$$

$$\|\forall X^+ A\|_1 := \bigcup_{\mathscr{H}\in\mathscr{P}(\Pi_c\times P)^{\mathbb{N}^k}} \|A[\mathscr{H}/X^+]\|_1 \textit{ (where } X \textit{ has arity } k\textit{).}$$

We set as usual $|A|_0 := \|A\|_0^{\perp}$ and $|A|_1 := \|A\|_1^{\perp}$.
We write $t \Vdash_0 A$ instead of $t \in |A|_0$ and $t \Vdash_1 A$ instead of $t \in |A|_1$.

Remark 3.49. *One can easily show that, said $(t = u) := \neg(t \neq u)$, we have:*
for all $A \in \mathcal{F}_0$, $\lambda x\lambda y(\mathtt{callcc})\lambda k(x)(k)y \Vdash_0 \forall^{\mathbb{N}} x \forall^{\mathbb{N}} y(x = y \to F\{x\} \to F\{y\})$ and
$\lambda x\lambda y(\mathtt{callcc})\lambda k(x)(k)y \Vdash_0 \forall p \forall q(p = q \to F\{p\} \to F\{q\})$.

Remark 3.50. *We have for example $\lambda a\lambda b(b)aa \Vdash_0 Nat\{0\}$, and in [Kriv04] the theorem 13 gives us that the formula $Nat\{x\} \to Nat\{succ(x)\}$ is realized by a proof-like term. Since Peano's axioms 1-6 (but not the induction) are realized by the standard interpretation for $\mathrm{PA2}$ then by the definition it follows that they are so also in the interpretation for SR_0. Therefore in a realizability model we can identify the subset*
$\{a \in M \text{ s.t. } \mathfrak{M} \models Nat[a]\} \subseteq M$ with $\mathbb{N}$, since it satisfies all the Peano's axioms (including the induction, by construction!).
The same for the interpretation for SR_1.

We are now interested in defining a syntactical notion of forcing, and study what happens to the realizability of formulas under forcing:

Definition 3.51. *The forcing transformation is the map $\Vdash : P \times \mathcal{F}_1 \to \mathcal{F}_0$, and we shall write $p \Vdash A$ instead of $\Vdash(p, A)$, defined by:*

$$(p \Vdash t \neq u) := \mathscr{C}(p) \to t \neq u$$

$(p \Vdash X(\vec{t})) := \mathscr{C}(p) \to X(\vec{t})$

$(p \Vdash X^+(\vec{t})) := \forall q(\mathscr{C}(pq) \to q \not\in X^+(\vec{t}))$

$(p \Vdash A \to B) := \forall q((p \Vdash A) \to (pq \Vdash B))$

$(p \Vdash \forall x A) := \forall x(p \Vdash A)$

$(p \Vdash \forall^{\mathbb{N}} x A) := \forall^{\mathbb{N}} x(p \Vdash A)$

$(p \Vdash \forall X A) := \forall X(p \Vdash A)$

$(p \Vdash \forall X^+ A) := \forall X^+(p \Vdash A)$.

Notation 3.52. *Given $A \in \mathcal{F}_1$ and $p \in P$ we define the set* $[p \Vdash A] := \bigcup_{q \in P}\{\pi^\tau\}_{\substack{\pi \in \Pi_c \ t.q \\ (\pi,q) \in \|A\|_1, \\ \tau \in C(pq)}} \subseteq \Pi_c$.

Lemma 3.53. *Let $\xi \in \Lambda_c$ and $p \in P$. Then: $(\xi, p) \Vdash_1 A \Leftrightarrow \xi \Vdash_0 [p \Vdash A]$.*

Proof. $(\xi, p) \Vdash_1 A \Leftrightarrow \forall \pi \in \Pi_c, \forall q \in P \ s.t. \ (\pi, q) \in \|A\|_1, (\xi, p) \star (\pi, q) \in \perp\!\!\!\perp_1 \Leftrightarrow$
$\forall \pi \in \Pi_c, \forall q \in P, \forall \tau \in C(pq) \ s.t. \ (\pi, q) \in \|A\|_1, \xi \star \pi^\tau \in \perp\!\!\!\perp_0 \Leftrightarrow \forall \varpi \in [p \Vdash A], \xi \star \varpi \in \perp\!\!\!\perp_0 \Leftrightarrow$
$\xi \in [p \Vdash A]^\perp$. □

Theorem 3.54. $\forall A \in \mathcal{F}^{close}_{1par}, \exists \chi_A, \chi'_A \in QP$ *s.t.*

i): χ_A, χ'_A *depend only on the propositional structure*[49] *of A (and so in particular they don't depend on the parameters of A)*

ii): $\chi_A \Vdash_0 (p \Vdash A) \to [p \Vdash A]$

iii): $\chi'_A \Vdash_0 [p \Vdash A] \to (p \Vdash A)$.

Proof. By induction on A:

$A = (n \neq m)$: from the definitions we get
$[p \Vdash A] = \begin{cases} \varnothing, & n \neq m \\ \{\pi^\tau \ s.t. \ \pi \in \Pi_c \ and \ \tau \in \bigcup_{q \in P} C(pq)\}, & n = m \end{cases}$ and $|p \Vdash A|_0 = \begin{cases} \Lambda_c, & n \neq m \\ (C(p).\Pi_c)^\perp, & n = m \end{cases}$.
One can easily check that we can then take $\chi_A = \lambda x(\chi)\lambda y(x)(\alpha_0)y$
and $\chi'_A = \lambda x \lambda y((\chi')x)(\alpha_4)y$.

$A = \overline{X}(\vec{n})$: similarly we can set χ_A and χ'_A.

$A = \overline{X^+}(\vec{n})$: we easily find that we can take $\chi_A = \chi$ and $\chi'_A = \chi'$.

[49] The propositional structure $stp(A)$ of a formula $\in \mathcal{F}_1$ is defined as: $stp(t \neq u) := stp(X(\vec{t})) := *_1$, $stp(X^+(\vec{t})) := *_2$, $stp(A \to B) := stp(A) \to stp(B)$, $stp(\forall x A) := stp(\forall X A) := stp(\forall X^+ A) := stp(A)$, $stp(\forall^{\mathbb{N}} x A) := stp(A)^*$ (where $*$ is a generic symbol).

$A = B \to C$: ii): let $\xi \Vdash_0 (p \digamma (B \to C))$ and $\varpi^\tau \in [p \digamma (B \to C)]$. Then $\tau \in C(pq')$ and $(\varpi, q') \in \|B \to C\|_1$ for a $q' \in P$. But then by the definition we have $\varpi = \eta.\pi$, for a $\eta \in \Lambda_c$ and a $\pi \in \Pi_c$, $q' = qr$, for some $q, r \in P$, with $(\eta, q) \Vdash_1 B$ and $(\pi, r) \in \|C\|_1$. By lemma 3.53 then $\eta \Vdash_0 [q \digamma B]$, and therefore by the induction hypothesis $(\chi'_B)\eta \Vdash_0 (q \digamma B)$, and so by definition of forcing $(\xi)(\chi'_B)\eta \Vdash_0 (pq \digamma C)$.
So again thanks to the induction hypothesis we have $(\chi_C)(\xi)(\chi'_B)\eta \Vdash_0 [pq \digamma H]$ and so again thanks to the lemma 3.53 $((\chi_C)(\xi)(\chi'_B)\eta, pq) \Vdash_1 C$. SO we obtain $((\chi_C)(\xi)(\chi'_B)\eta \star \pi, (pq)r) \in \perp\!\!\!\perp_1$ and, since $(\alpha_3)\tau \in C((pq)r)$, we have $(\chi_C)(\xi)(\chi'_B)\eta \star \pi^{(\alpha_3)\tau} \in \perp\!\!\!\perp_0$. We finally easily check that we can take $\chi_A := (\overline{\alpha_3})\lambda x \lambda y (\chi_C)(x)(\chi'_B)y$.
iii): in a similar way we find that we can set $\chi'_A := \lambda x \lambda y (\chi'_C)((\overline{\gamma_5})x)(\chi_B)y$.
i): clear.

$A = \forall x B\{x\}$: by the definitions we have $\|p \digamma A\|_0 = \bigcup_{n\in\mathbb{N}} \|p \digamma B[n/x]\|_0$ and $[p \digamma A] = \bigcup_{\substack{q\in P \\ n\in\mathbb{N}}} \{\pi^\tau \ s.t. \ \tau \in C(pq) \ and \ (\pi, q) \in \|B[n/x]\|_1\} = \bigcup_{n\in\mathbb{N}} [p \digamma B[n/x]]$. So we that, since by induction hypothesis $\chi_{B[n/x]}$ and $\chi'_{B[n/x]}$ don't depend on n, we can take $\chi_A = \chi_{B[n/x]}$ and $\chi_A = \chi'_{B[n/x]}$ for any integer n.

$A = \forall^{\mathbb{N}} x B\{x\}$: in this case we have:
$[p \digamma A] = \bigcup_{\substack{q\in P \\ n\in\mathbb{N}}} \{(\mathtt{succ})^n \mathtt{0}.\pi^\tau \ t.q \ \tau \in C(p(1q)) \ and \ (\pi, q) \in \|B[n]\|_1\}$
and $\|p \digamma A\|_0 = \bigcup_{n\in\mathbb{N}} \{(\mathtt{succ})^n \mathtt{0}.\|p \digamma B[n]\|_0\}$. Now:
iii): let's check that $\chi'_A := \lambda x \lambda n (\chi'_{B[m]})((\overline{\gamma})x)n$, where m is a generic integer and $\alpha : pq \longrightarrow p(1q)$ (for example $\alpha = (\gamma_5)(\alpha_1)(\gamma_5)(\alpha_4)(\alpha_1)$), works: let $\xi \Vdash_0 [p \digamma A]$ and $\varpi \in \|p \digamma A\|_0$. Then $\varpi = (\mathtt{succ})^n \mathtt{0}.\pi$ with $\pi \in \|p \digamma B[n]\|_0$, for an $n \in \mathbb{N}$. We have to show that $\chi'_A \star \xi.(\mathtt{succ})^n \mathtt{0}.\pi \in \perp\!\!\!\perp_0$, for which it is enough to prove that $\chi'_{B[m]} \star ((\overline{\gamma})\xi)(\mathtt{succ})^n \mathtt{0}.\pi \in \perp\!\!\!\perp_0$. But the induction hypothesis gives $\chi'_{B[m]} \Vdash_0 [p \digamma B[n]] \to (p \digamma B[n])$, since $\chi_{B[m]}$ don't depend on m, and so it is enough to show that $((\overline{\gamma})\xi)(\mathtt{succ})^n \mathtt{0} \Vdash_0 [p \digamma B[n]]$, i.e. $((\overline{\gamma})\xi)(\mathtt{succ})^n \mathtt{0} \star \rho \in \perp\!\!\!\perp_0 \ \forall \rho \in [p \digamma B[n]]$, i.e. $\forall \rho = \varpi^\tau$ with $(\varpi, q) \in \|B[n]\|_1$, $\tau \in C(pq)$ and $q \in P$. Now, since then $((\overline{\gamma})\xi)(\mathtt{succ})^n \mathtt{0} \star \rho \succ \xi \star (\mathtt{succ})^n \mathtt{0}.\varpi^{(\overline{\gamma})\tau}$ and $(\overline{\gamma})\tau \in C(pq)$, we finish thanks to the hypothesis on ξ.
ii): in the same way we find that we can take $\chi_A := \lambda x \lambda n ((\overline{(\alpha_3)(\gamma_1)})\chi_{B[m]})(x)n$ for any integer m.
i): clear.

$A = \forall X B\{X\}$: in the same way we find that we can take $\chi_A = \chi_{B[n/x]}$ and $\chi'_A = \chi'_{B[n/x]}$ for any integer n.

$A = \forall X^+ B\{X^+\}$: exactly as the case $A = \forall X B$.

□

Corollary 3.55. *$\forall A \in \mathcal{F}^{close}_{1par}$ we have:*

i): $\xi \Vdash_0 (p \digamma A) \Rightarrow ((\chi_A)\xi, p) \Vdash_1 A$

ii): $(\xi, p) \Vdash_1 A \Rightarrow (\chi'_A)\xi \Vdash_0 (p \digamma A)$.

Proof. Trivial from theorem 3.54 and lemma 3.53. □

Corollary 3.56. *$\forall A \in \mathcal{F}^{close}_{1par}$ we have:*

i): $\lambda x(\chi'_A)(\overline{\gamma_1})(\chi_A)x \Vdash_0 (p \digamma A) \to (pq \digamma A)$

ii): $\lambda x(\chi'_A)(\overline{\gamma_2})(\chi_A)x \Vdash_0 (p \digamma A) \to (qp \digamma A)$

Proof. i): Using corollary 3.55 we have $\xi \Vdash_0 (p \digamma A) \Rightarrow ((\chi_A)\xi, p) \Vdash_1 A \Rightarrow ((\overline{\gamma_1})(\chi_A)\xi, pq) \Vdash_1 A \Rightarrow (\chi'_A)(\overline{\gamma_1})(\chi_A)\xi \Vdash_0 (p \digamma A)$, from where the thesis (we recall that we have already observed that in general $((\overline{\gamma_1})\eta, pq) \leq (\eta, p)$, from where the second implication).

ii): The same as i).

□

Theorem 3.57. *$\forall A \in \mathcal{F}^{usu}_0$ closed with parameters, $\exists \chi^0_A, \chi^1_A \in QP$ s.t.*

i): *χ^0_A, χ^1_A depend only on the propositional structure[50] of A (and so in particular they don't depend on the parameters of A)*

ii): $\chi^0_A \Vdash_0 (p \digamma A) \to \mathscr{C}(p) \to A$

iii): $\chi^1_A \Vdash_0 (\mathscr{C}(p) \to A) \to (p \digamma A)$.

Proof. Induction on A:

$A = (n \neq m)$ or $A = \mathcal{H}(\vec{n})$: since in these cases $(p \digamma A) = (\mathscr{C}(p) \to A)$, we can take $\chi^0_A = \chi^1_A = \lambda xx$.

$A = \forall xB$: since in this case $(p \digamma A) = \forall n(p \digamma B\{x\})$, and by the induction hypothesis $\forall n \in \mathbb{N}$, $\chi^0_{B[n/x]}$ and $\chi^1_{B[n/x]}$ don't depend on n, one can easily check that we can take $\chi^0_A = \chi^0_{B[n/x]}$ and $\chi^1_A = \chi^1_{B[n/x]}$ for any n.

$A = \forall XB$: exactly as the case that we've just done.

$A = \forall^{\mathbb{N}} xB$: in this case $\|p \digamma A\|_0 = \bigcup_{n\in\mathbb{N}} \|\{(\mathtt{succ})^n \mathtt{0}\} \to (p \digamma B[n/x])\|_0$ and $\|\mathscr{C}(p) \to A\|_0 = \bigcup_{n\in\mathbb{N}} \|\mathscr{C}(p) \to \{(\mathtt{succ})^n \mathtt{0}\} \to B[n/x]\|_0$. One can easily check that we can then take $\chi^0_A = \lambda x\lambda y\lambda z((\chi^0_{B[n/x]})(x)z)y$ and $\chi^1_A = \lambda x\lambda y(\chi^1_{B[n/x]})\lambda z(x)zy$, for any n.

$A = B \to C$: in this case $\|p \digamma A\|_0 = \bigcup_{q\in P} \|(q \digamma B) \to (pq \digamma C)\|_0$.
Now, one can easily show that:
$\lambda x\lambda y\lambda z((\chi^0_C)(x)(\chi^1_B)\lambda dz)(\alpha_2)y \Vdash_0 ((p \digamma B) \to (pp \digamma C)) \to \mathscr{C}(p) \to A$ and
$\lambda x\lambda y(\chi^1_C)\lambda z((x))(\alpha_0)z)((\chi^0_B)y)(\alpha_0)(\alpha_1)z \Vdash_0 (\mathscr{C}(p) \to A) \to q \digamma B \to pq \digamma C$;
so we can take $\chi^0_A = \lambda x\lambda y\lambda z((\chi^0_C)(x)(\chi^1_B)\lambda dz)(\alpha_2)y$ and
$\chi^1_A = \lambda x\lambda y(\chi^1_C)\lambda z((x))(\alpha_0)z)((\chi^0_B)y)(\alpha_0)(\alpha_1)z$.

[50] The propositional structure $stp(A)$ of a formula $\in \mathcal{F}_0$ is defined as for the formulas of $\mathcal{F}_1$.

□

The following technical result says that the algebras SR_0 and SR_1 are well suited to "move the forcing" from the realizability side to the language side and vice-versa.

Theorem 3.58. *$\forall A \in \mathcal{F}_0^{usu}$ closed with parameters, $\exists \chi_A^+, \chi_A^- \in QP$ s.t.*

i): *χ_A^+, χ_A^- depend only on the propositional structure of A (and so in particular they don't depend on the parameters of A)*

ii): $(\xi, p) \Vdash A \Rightarrow (\chi_A^+)\xi \Vdash_0 \mathscr{C}(p) \to A$

iii): $\xi \Vdash_0 \mathscr{C}(p) \to A \Rightarrow ((\chi_A^-)\xi, p) \Vdash A$.

Proof. We can take $\chi_A^+ := \lambda x(\chi_A^0)(\chi_A')x$ and $\chi_A^- := \lambda x(\chi_A)(\chi_A^1)x$ and use corollary 3.56 and theorem 3.54. □

3.3.3 General properties of the "generic"

Definition 3.59.

- *We add to the condition terms the construction $\phi(\vec{t^P})$, for every ϕ function symbol (with arity) that we need. The definition of the interpretation of condition terms changes in the obvious way.*

- *We fix an unary predicate symbol J and we add to the formulas of $\mathcal{F}_1$, with the other construction, also the constructions: $J(t^P) \mid \mathscr{C}(t^P) \to A \mid \forall pA$. We call* extended formulas of $\mathcal{F}_1$ *this formulas.*
  The definition of models is changed in the obvious way, in particular giving a set $J_{\mathfrak{M}} \subseteq P$, called "the generic" *and setting $\mathfrak{M} \models J(t^P)$ iff $t^P_{\mathfrak{M}} \in J_{\mathfrak{M}}$.*

- *We define a SR_1-interpretation $\|.\|_1$ of the extended formula $\mathcal{F}_{1par}^{close}$ adding the clauses:*

$$\|J(\vec{[\![t^P]\!]})\|_1 := \Pi_c \times \{[\![t^P]\!]\}$$
$$\|\mathscr{C}([\![t^P]\!]) \to A\|_1 := \{(\tau, 1)\}_{\tau \in C([\![t^P]\!])} \cdot \|A\|_1$$
$$\|\forall pA\|_1 := \bigcup_{q \in P} \|A[q/p]\|_1.$$

- *We add to the formulas of $\mathcal{F}_0$, with the other constructions, also the construction: $t^P \not\varepsilon J^+(u^P)$.*
  The definition of model changes adding the clause $\mathfrak{M} \models t^P \not\varepsilon J^+(u^P)$ iff $t^P_{\mathfrak{M}} \neq u^P_{\mathfrak{M}}$.

- *We define a SR_0-interpretation $\|.\|_0$ of the extended formulas of $\mathcal{F}_{0par}^{close}$ adding the clause:* $\|[\![t^P]\!] \not\varepsilon J^+([\![u^P]\!])\|_0 := \{\pi \in \Pi_c \ s.t. (\pi, [\![t^P]\!]) \in \|J([\![u^P]\!])\|_1\} = \|[\![t^P]\!] \neq [\![u^P]\!]\|_0$.

- *We extend the forcing transformation to the extended formulas as follows:*

$$(p \Vdash J(t^P)) := \forall q(\mathscr{C}(pq) \to q \not\varepsilon J^+(t^P))$$
$$(p \Vdash \forall qA) := \forall q(q \Vdash A)$$

Note that $\|p \Vdash J(t^P)\|_0 = \|\neg\mathscr{C}(pt^P)\|_0$.

Remark 3.60.
- *We can easily prove that:* $(\xi, p) \Vdash_1 J(q) \Rightarrow (\chi')\xi \Vdash_0 \neg\mathscr{C}(pq)$ *and* $\xi \Vdash_0 \neg\mathscr{C}(pq) \Rightarrow ((\chi)\xi, p) \Vdash_1 J(q)$ *and that* $(\xi, p) \Vdash_1 J(q) \Rightarrow \lambda t(\chi')\xi(\alpha_1)t \Vdash_0 \neg\mathscr{C}(qp)$.
- *From the previous point it follows that the corollary 3.55 holds also for the extended formulas.*

Notation 3.61. *We define the formulas* $p \sqsubseteq q := \forall r(\neg\mathscr{C}(qr) \to \neg\mathscr{C}(pr))$.
We define the formulas $p \sim q := p \sqsubseteq q \wedge q \sqsubseteq p$.
Observe that in the models we have: $\mathfrak{M} \models p \sqsubseteq q$ *iff* $[\![p]\!] \sqsubseteq [\![q]\!]$ *(where the symbol* $\sqsubseteq$ *at right is the relation on* P *that we have defined in the remark 3.6).*
We will often write $A \to \mathscr{C}(p)$ *instead of* $\neg\mathscr{C}(p) \to \neg A$.

Lemma 3.62. *Let* $A \in \mathcal{F}_1$ *extended. We have:* $(\theta, 1) \Vdash_1 A \Rightarrow \exists\theta' \in QP\ s.t.\ \forall p \in P,\ (\theta', p) \Vdash_1 A$.

Proof. We can take $\theta' = (\overline{\alpha})\theta$, where $p'q' \xrightarrow{\alpha} 1q'$ (for example $\alpha = (\alpha_1)(\alpha_4)(\alpha_0)(\alpha_1)$). □

Theorem 3.63 (Properties of the generic)**.** *We have:*

i): $\exists\theta \in QP\ s.t.\ \forall r \in P,\ (\theta, r) \Vdash_1 \neg J[1]$

ii): $\exists\theta \in QP\ s.t.\ \forall r \in P,\ (\theta, r) \Vdash_1 \forall p(\neg\mathscr{C}(p) \to J(p))$

iii): $\exists\theta \in QP\ s.t.\ \forall r \in P,\ (\theta, r) \Vdash_1 \forall p\forall q(\neg J(p) \to J(pq) \to J(q))$

iv): $\exists\theta \in QP\ s.t.\ \forall r \in P,\ (\theta, r) \Vdash_1 \forall p((\forall q(\neg\mathscr{C}(pq) \to J(q))) \to \neg J(p))$

v): $\exists\theta \in QP\ s.t.\ \forall r \in P,\ (\theta, r) \Vdash_1 \forall p\forall q(J(p) \to q \sqsubseteq p \to J(q))$

vi): $\exists\theta \in QP\ s.t.\ \forall r \in P,\ (\theta, r) \Vdash_1 \forall p\forall q(\neg\mathscr{C}(pq) \to \neg J(p) \to J(q))$

Proof. i): Let $r(pq) \xrightarrow{\gamma} p'1$ (for example $\gamma = (\alpha_4)(\alpha_0)(\alpha_0)(\alpha_1)$). We show that $(\overline{\gamma}, r) \Vdash_1 \neg J(1)$. In fact this means that $(\overline{\gamma}, r) \star (\xi, p).(\pi, q) \in \perp\!\!\!\perp_1\ \forall(\pi, q) \in \Pi_c \times P,\ \forall\xi \Vdash_1 J(1)$, i.e. $(\overline{\gamma} \star \xi.\pi, r(pq)) \in \perp\!\!\!\perp_1$. But if $\tau \in C(r(pq))$ then $\overline{\gamma} \star \xi.\pi \succ \xi \star \pi^{(\overline{\gamma})\tau}$ and $(\overline{\gamma})\tau \in C(p1)$, and since the hypothesis on ξ gives $(\xi \star \pi, p1) \in \perp\!\!\!\perp_1$, we obtain that $\xi \star \pi^{(\overline{\gamma})\tau} \in \perp\!\!\!\perp_0$, i.e. the thesis.

ii): Let $(\eta, q) \Vdash_1 \neg\mathscr{C}(p)$, i.e. $\eta \star \xi.\rho^{\upsilon} \in \perp\!\!\!\perp_0,\ \forall p' \in P, \upsilon \in C(q(1p'))$. Let $(\pi, p) \in \|J(p)\|_1$. Let's set $\theta = (\chi)\lambda x\lambda y(((\chi')y)(\beta)x)(\alpha)x$, where $r(qp) \xrightarrow{\alpha} p$ and $r(qp) \xrightarrow{\beta} q(1r)$ (for example $\alpha = (\alpha_0)(\alpha_1)(\alpha_0)(\alpha_1)$ and $\beta = (\gamma_5)(\alpha_1)(\gamma_5)(\alpha_4)(\alpha_0)(\alpha_3)$). We have to show that $(\theta, r) \star (\eta, q).(\pi, p) \in \perp\!\!\!\perp_1$, i.e. $\theta \star \eta.\pi^{\tau} \in \perp\!\!\!\perp_0$ for $\tau \in C(r(qp))$. Since we can easily check that $\theta \star \eta.\pi^{\tau} \succ \eta \star (\alpha)\tau.\pi^{(\beta)\tau}$ it is enough to prove that $\eta \star (\alpha)\tau.\pi^{(\beta)\tau} \in \perp\!\!\!\perp_0$. Here it is: by the hypothesis on (η, q) we obtain $(\eta, q) \star ((\alpha)\tau, 1).(\pi, r) \in \perp\!\!\!\perp_1$, i.e. $(\eta \star (\alpha)\tau.\pi, q(1r)) \in \perp\!\!\!\perp_1$, from where the desired result.

iii): Let $(\xi, p') \Vdash_1 \neg J(p)$, $(\eta, q) \Vdash_1 J(pq)$ and $(\pi, q) \in \|J(q)\|_1$. Let's set $\theta = \lambda x\lambda y(\overline{\alpha})(x)(\overline{\beta})y$, where $r(p'(q'q)) \xrightarrow{\alpha} p'((qq')1)$ and $(qq')p \xrightarrow{\beta} q'(pq)$ (for example $\alpha =$ and $\beta = (\alpha_1)(\alpha_3)(\alpha_1)$). We need to show that $(\theta \star \xi.\eta.\pi, r(p'(q'q))) \in \perp\!\!\!\perp_1$, i.e. $\theta \star \xi.\eta.\pi^{\tau} \in \perp\!\!\!\perp_0$, for $\tau \in C(r(p'(q'q)))$. If we get to show that $((\overline{\beta})\eta, qq') \Vdash_1 J(p)$ then we have finished, as: by the hypothesis on (ξ, p') we obtain $(\xi, p') \star ((\overline{\beta})\eta, qq').(\pi, 1) \in \perp\!\!\!\perp_1$, i.e. $(\xi \star (\overline{\beta})\eta.\pi, p'((qq')1)) \in \perp\!\!\!\perp_1$, and so

$\xi \star (\overline{\beta})\eta.\pi^{(\overline{\alpha})\tau} \in \perp\!\!\!\perp_0$. But (easy to verify) $\theta \star \xi.\eta.\pi^{\tau} \succ \xi \star (\overline{\beta})\eta.\pi^{(\overline{\alpha})\tau}$ and so we have the desired result.
Let's show now that $((\overline{\beta})\eta, qq') \Vdash_1 J(p)$:
we have to prove that $((\overline{\beta})\eta, qq') \star (\varpi, p) \in \perp\!\!\!\perp_1 \ \forall \varpi \in \Pi_c$. So, let $\upsilon \in C((qq')p)$ and let's show that $(\overline{\beta})\eta \star \varpi^{\tau} \succ \eta \star \varpi^{(\beta)\tau} \in \perp\!\!\!\perp_0$. This clearly follows from the hypothesis on (η, q') and from the property of β.

iv): Thanks to lemma 3.62 it is enough to find a $\theta \in QP$ s.t. $(\theta, 1) \star (\eta, r).(\xi, q).(\pi, r') \in \perp\!\!\!\perp_1$, i.e. $(\theta \star \eta.\xi.\pi, 1(r(qr'))) \in \perp\!\!\!\perp_1$, for $(\xi, q) \Vdash_1 J(p)$, $(\eta, r) \Vdash_1 \forall q(\neg\mathscr{C}(pq) \to J(q))$ and $(\pi, r') \in \Pi_c \times P$. By the remark 4.2 we have $(\chi')\xi \Vdash_0 \neg\mathscr{C}(pq)$, and so also $\lambda d(\chi')\xi \Vdash_0 \mathscr{C}(1) \to \neg\mathscr{C}(pq)$; by theorem 3.57 then $((\chi^{-}_{\neg\mathscr{C}(pq)})\xi, 1) \Vdash_1 \neg\mathscr{C}(pq)$ and therefore (easy to check) $((\Xi)\xi, 1) \Vdash_1 \neg\mathscr{C}(pq)$, with $\Xi = \lambda x(\chi^{-}_{\neg\mathscr{C}(pq)})\lambda d(\chi')x$. By the hypothesis on (η, r) then we have $(\eta, r) \star ((\Xi)\xi).(\pi, q) \in \perp\!\!\!\perp_1$, i.e. $(\eta \star (\Xi)\xi.\pi, r(1q)) \in \perp\!\!\!\perp_1$. We can easily verify that we can then take $\theta = (\overline{\alpha})\lambda x\lambda y(x)(\Xi)y$, where $1(r(qr')) \xrightarrow{\alpha} r(1q)$ (for example $\alpha = (\alpha_1)(\alpha_3)(\alpha_1)(\alpha_4)(\alpha_1)(\alpha_0)(\alpha_3)(\alpha_0)(\alpha_1)$).

v): Thanks to lemma 3.62 it is enough to show that $\theta \in QP$ s.t. $(\theta, 1) \star (\xi, p').(\eta, r).(\pi, q) \in \perp\!\!\!\perp_1$, i.e. $(\theta \star \xi.\eta.\pi, 1(p'(rq))) \in \perp\!\!\!\perp_1$, for $(\xi, p') \Vdash_1 J(p)$, $(\eta, r) \Vdash_1 q \sqsubseteq p$ and $(\pi, q) \in \Pi_c \times P$. By remark 4.2 we have $(\tilde{\chi})\xi \Vdash_0 \neg\mathscr{C}(pp')$ for a certain $\tilde{\chi}$, and by theorem 3.57 $(\chi^{+}_{q\sqsubseteq p})\eta \Vdash_0 \mathscr{C}(r) \to q \sqsubseteq p$. Now, let $(qr)p \xrightarrow{\beta_1} r$ and $(qr)p \xrightarrow{\beta_2} pq$ (for example we can take $\beta_1 = (\alpha_0)(\gamma_2)$ and $\beta_2 = (\gamma_4)(\alpha_1)$); if we prove that $h := \lambda u\lambda v\lambda t(\lambda x(((v)(\beta_1)t)x)(\beta_2)t)u \Vdash_0 \neg\mathscr{C}(pp') \to (\mathscr{C}(r) \to q \sqsubseteq p) \to \neg\mathscr{C}((p'r)q)$, we have finished. In fact then, said $\Xi := \lambda a\lambda b((h)(\chi')a)(\chi^{+}_{q\sqsubseteq p})b$, we would have $(\Xi)\xi\eta \Vdash_0 \neg\mathscr{C}((p'r)q)$. Again by remark 4.2 we would have $((\chi)((\Xi)\xi)\eta, p'r) \Vdash_1 J(q)$, and so (easy to check) $((\Xi')\xi\eta, p'r) \Vdash_1 J(q)$, with $\Xi' := \lambda x\lambda y(\chi)((\Xi)x)y$, from where, by definition, $((\Xi')\xi\eta, p'r) \star (\pi, q) \in \perp\!\!\!\perp_1$. It is easy to show that we could then take $\theta = \lambda x\lambda y(\overline{\alpha})((\Xi')x)y$, where $1(p'(rq)) \xrightarrow{\alpha} (p'r)q$.
Let's prove now that $h \Vdash_0 \neg\mathscr{C}(pp') \to (\mathscr{C}(r) \to q \sqsubseteq p) \to \neg\mathscr{C}((p'r)q)$:
we easily have that $h \star u.v.t.\pi \succ v \star (\beta_1)t.u.(\beta_2)t.\pi$ for all $\pi \in \Pi_c$. Then if $u \Vdash_0 \neg\mathscr{C}(pp')$, $v \Vdash_0 (\mathscr{C}(r) \to q \sqsubseteq p)$ and $t \in C((p'r)q)$, we have $(\beta_1)t \in C(r)$ and $(\beta_2)t \in C(qp')$, and so $u.(\beta_2)t.\pi \in \|q \sqsubseteq p\|_0 = \bigcup_{r' \in P} |\neg\mathscr{C}(pr')|_0.C(qr').\Pi_c$, that finally gives us the wanted result.

vi): We can reason as in (v).

□

Remark 3.64. *The previous theorem tells us that, in the realizability models $\mathfrak{M}$, $J_{\mathfrak{M}} \subseteq P$ is an initial segment (point (v)), non trivial (point (i)) and prime (point (iii)), which contains all the non trivial conditions (point (ii)).*

Theorem 3.65 (Density)**.** *We have:*
$\exists \theta \in QP \ s.t. \ \forall r \in P, \forall \phi : P \to P, \ (\theta, r) \Vdash_1 (\forall p(\neg\mathscr{C}(p \cdot \phi(p)) \to J(p))) \to \neg\forall p J(p \cdot \phi(p))$.

Proof. Let $(\xi, p_0) \Vdash_1 \forall p(\neg\mathscr{C}(p \cdot \phi(p)) \to J(p))$, $(\eta, q_0) \Vdash_1 \forall p J(p \cdot \phi(p))$ and $(\pi, r_0) \in \Pi_c$. Thanks to 4.4 it is enough to find a $\theta \in QP$ s.t. $(\theta \star \xi.\eta.\pi, 1(p_0(q_0r_0))) \in \perp\!\!\!\perp_1$. Let's check that $\theta = (\overline{\beta})\lambda x\lambda y(x)(\zeta)y$ works, where $\zeta = (\chi)\lambda d\lambda x\lambda y((\chi')x)(\alpha)y$, where $qr \xrightarrow{\alpha} q(qr)$ and $1(p(qr)) \xrightarrow{\beta} p(1q)$ (for example we can take $\alpha = (\alpha_1)(\alpha_0)(\alpha_3)(\alpha_2)$ and $\beta = (\alpha_1)(\alpha_3)(\alpha_1)(\alpha_4)(\alpha_1)(\alpha_0)(\alpha_3)(\alpha_0)(\alpha_1)$).

If we prove that $((\zeta)\eta, 1) \Vdash_1 \neg\mathscr{C}(q_0 \cdot \phi(q_0))$ we have finished: in fact by the hypothesis on (ξ, p_0) we would have $(\xi, p_0) \Vdash_1 \neg\mathscr{C}(q_0 \cdot \phi(q_0)) \to J(q_0)$ and therefore in particular $\forall \pi \in \Pi_c,\ (\xi, p_0) \star ((\zeta)\eta, 1).(\pi, q_0) \in \perp\!\!\!\perp_1$, i.e. $\forall \pi \in \Pi_c,\ \forall \tau \in C(p_0(1q_0)),\ \xi \star (\zeta)\eta.\pi^\tau \in \perp\!\!\!\perp_0$. But since we can easily check that $\theta \star \xi.\eta.\pi^\tau \succ \xi \star\ (\zeta)\eta.\pi^{(\beta)\tau}$, and since $\tau \in C(1(p_0(q_0r_0))) \Rightarrow (\beta)\tau \in C(p_0(1q_0))$, we would obtain $(\theta \star \xi.\eta.\pi, 1(p_0(q_0r_0))) \in \perp\!\!\!\perp_1$, that is the wanted result.
Let's show now that $((\zeta)\eta, 1) \Vdash_1 \neg\mathscr{C}(q_0 \cdot \phi(q_0))$:
we have to show that $\forall(\varpi, p) \in \Pi_c \times P,\ \forall \tau \in C(q_0 \cdot \phi(q_0)),\ ((\zeta)\eta, 1) \star (\tau, 1).(\varpi, r) \in \perp\!\!\!\perp_1$. But since we easily check that $(\zeta)\eta \star \tau.\varpi^{\tau'} \succ \eta \star \varpi^{(\alpha)\tau}$, it turn out to be same as $\forall \varpi \in \Pi_c$, $\forall \tau \in C(q_0 \cdot \phi(q_0))$, $\eta \star \varpi^{(\alpha)\tau} \in \perp\!\!\!\perp_0$. Now, by the hypothesis on (η, q_0) we have $(\eta, q_0) \Vdash_1 J(q_0 \cdot \phi(q_0))$, i.e. $\forall \varsigma \in C(q_0(q_0 \cdot \phi(q_0))),\ \eta \star \varpi^\varsigma \in \perp\!\!\!\perp_0$, that is exactly the result we're looking for, because $\tau \in C(q_0 \cdot \phi(q_0)) \Rightarrow (\alpha)\tau \in C(q_0(q_0 \cdot \phi(q_0)))$. □

Remark 3.66. *We can see that the formula that we have just realized in the density theorem says that, in the realizability models $\mathfrak{M}$, $P \setminus J_{\mathfrak{M}}$ intersects all the sets of the form $\{p \cdot \phi(p)\}_{p \in P}$ that are dense.*

3.3.4 The Countable Chain Condition

This subsection is purely technical and it serves to obtain some strong properties on some forcing structures. We will use this results in the last subsection.

Notation 3.67. *We define the following formulas of $\mathcal{F}_1$: $X \subseteq Y := \forall^{\mathbb{N}} x(Xx \to Yx)$, and the same for $X^+ \subseteq Y$, $X \subseteq Y^+$ and $X^+ \subseteq Y^+$. Finally we set $X \simeq Y := X \subseteq Y \wedge Y \subseteq X$ and the same for the other three combination of X^+ and Y^+.*

Definition 3.68. *Let $(P, C, 1, \cdot)$ be a forcing structure.*
We say that she satisfies the Countable Chain Condition *(from now on "CCC") iff*
$\exists \boldsymbol{ccd}, \boldsymbol{ccd'} \in QP, \exists \boldsymbol{p} : \mathscr{P}(\Pi_c \times P)^{\mathbb{N}} \to P$ s.t. $\forall \mathscr{H} : \mathbb{N} \to \mathscr{P}(\Pi_c \times P)$ we have:

$$\boldsymbol{ccd} \Vdash_0 \forall^{\mathbb{N}} n \forall q (q \not\in \mathscr{H}(n) \to \boldsymbol{p}(\mathscr{H}) \sqsubseteq q)$$

$$\boldsymbol{ccd'} \Vdash_0 \mathrm{AtMost} \to \mathrm{AtLeast} \to \mathrm{NonTriv} \to \mathrm{Decr} \to \mathscr{C}(\boldsymbol{p}(\mathscr{H})),$$

where $\mathrm{AtMost} := \forall^{\mathbb{N}} n \forall q \forall r (q \not\in \mathscr{H}(n) \to r \not\in \mathscr{H}(n) \to q = r)$,
$\mathrm{AtLeast} := \forall^{\mathbb{N}} n \exists q (q \not\in \mathscr{H}(n))$,
$\mathrm{NonTriv} := \forall^{\mathbb{N}} n \forall q (q \not\in \mathscr{H}(n) \to \mathscr{C}(q))$,
$\mathrm{Decr} := \forall^{\mathbb{N}} m \forall^{\mathbb{N}} n \forall q \forall r (q \not\in \mathscr{H}(n) \to r \not\in \mathscr{H}(n+m) \to r \sqsubseteq q)$.

The meaning of those formulas is given by the following remark:

Remark 3.69. *We note that a predicate $\mathscr{H} : \mathbb{N} \to \mathscr{P}(\Pi_c \times P)$ naturally induces a sequence $\{\tilde{\mathscr{H}_n}\}_{n \in \mathbb{N}}$ of subsets of P setting $\tilde{\mathscr{H}_n} := \{p \in P\ t.q\ \mathfrak{M} \models p \not\in \mathscr{H}(n)\} \subseteq P$. Then we have:*
$\mathfrak{M} \models \forall^{\mathbb{N}} n \exists q\ q \not\in \mathscr{H}(n)$ iff $\forall n \in \mathbb{N} \subseteq M, \forall q \in \tilde{\mathscr{H}_n},\ \boldsymbol{p}(\mathscr{H}) \leq q$, and
$\mathfrak{M} \models \mathrm{AtMost} \to \mathrm{AtLeast} \to \mathrm{NonTriv} \to \mathrm{Decr} \to \mathscr{C}(\boldsymbol{p}(\mathscr{H}))$ *iff*
$(\forall n \in \mathbb{N} \subseteq M (\exists! p \in \tilde{\mathscr{H}_n}$ that we name p_n, and $p_n \in \mathscr{C})$ and $\{p_n\}_{n \in \mathbb{N}}$ decreasing$) \Rightarrow \boldsymbol{p}(\mathscr{H}) \in \mathscr{C}$.

So, if a forcing structure satisfies the CCC then, in the realizability models $\mathfrak{M}$, the "usual" set theoretic countable chain condition holds for the sequences that can be written in the form $\{\tilde{\mathscr{H}}_n\}_{n\in\mathbb{N}}$.

Lemma 3.70 (with axiom of choice)**.** $\exists\phi : \mathbb{N}\times P\times\mathscr{P}(\Pi_c\times P)\longrightarrow P$ *s.t.*
$\forall n\in\mathbb{N}$, $\forall\mathscr{H}:\mathbb{N}\longrightarrow\mathscr{P}(\Pi_c\times P)$, *said* $F[p,q,\mathscr{H},n]=\mathscr{C}(pq)\to q\not\in\mathscr{H}(n)$, *we have:*
$\sigma\Vdash_0(\forall^{\mathbb{N}}xF[p,\phi(x,p,\mathscr{H}(n)),\mathscr{H},n])\to\forall qF[p,q,\mathscr{H},n]$ *(σ is the instruction introduced in he notation 3.1).*
Note that by definition $(p\Vdash\mathscr{H}(n))=\forall qF[p,q,\mathscr{H},n]$.

Proof. Let's fix $p\in P$ and $\mathscr{H}\subseteq\mathscr{P}(\Pi_c\times P)$. We can clearly define a function $\mathscr{H}\subseteq\Pi_c\times P\longrightarrow\tilde{\mathscr{H}}\in\mathscr{P}(\Pi_c\times P)^{\mathbb{N}}$ setting for example $\tilde{\mathscr{H}}(0):=\mathscr{H}$ and $\tilde{\mathscr{H}}(n+1):=\varnothing$. Let's consider then the family $\{A_j^{p,\mathscr{H}}\}_{j\in\mathbb{N}}$ of subsets of P defined by:
$A_j^{p,\mathscr{H}}:=\{q\in P\ s.t.\ \pi_j\in\bigcup_{q'\in P}\|F[p,q',\tilde{\mathscr{H}},0]\|_0\Rightarrow\pi_j\in\|F[p,q,\tilde{\mathscr{H}},0]\|_0\}\subseteq P$.
Clearly (by construction!) we have $A_j^{p,\mathscr{H}}\neq\varnothing\ \forall j\in\mathbb{N}$.
But then by the axiom of choice it exist $\phi_{p,\mathscr{H}}:\mathbb{N}\longrightarrow\bigcup_{j\in\mathbb{N}}A_j^{p,\mathscr{H}}$ s.t. $\phi_{p,\mathscr{H}}(i)\in A_i^{p,\mathscr{H}}\ \forall i\in\mathbb{N}$,
i.e. $\pi_j\in\bigcup_{q'\in P}\|F[p,q',\tilde{\mathscr{H}},0]\|_0\Rightarrow\pi_j\in\|F[p,\phi_{p,\mathscr{H}}(i),\tilde{\mathscr{H}},0]\|_0$.
Let's verify that we can then take $\phi(n,p,\mathscr{H}):=\phi_{p,\mathscr{H}}(i)\in P$. Let $\xi\Vdash_0\forall^{\mathbb{N}}xF[p,\phi(x,p,\mathscr{H}(n)),\mathscr{H},n]$ and $\pi\in\|\forall qF[p,q,\mathscr{H},n]\|_0=\bigcup_{q\in P}\|F[p,q,\mathscr{H},n]\|_0$. We note first that by construction ϕ has the following property: $\forall j\in\mathbb{N}$, $\pi_j\in\bigcup_{q'\in P}\|F[p,q',\mathscr{H},n]\|_0\Rightarrow\pi_j\in\|F[p,\phi(j,p,\mathscr{H}(n)),\mathscr{H},n]\|_0$.
In fact, for that we can use the fact that $\|F[p,q',\mathscr{H},n]\|_0=\|F[p,q',\tilde{\mathscr{H}}(n),0]\|_0$ (for which one can use the definition of F and the fact that $\tilde{\mathscr{H}}(n)(0)=\mathscr{H}(n)$) and that $\phi(j,p,\mathscr{H}(n))\in A_j^{p,\mathscr{H}(n)}$.
Now, since $\pi=\pi_j$ for some $j\in\mathbb{N}$ then by the property of ϕ we obtain $\pi\in\|F[p,\phi(j,p,\mathscr{H}(n)),\mathscr{H},n]\|_0$.
We have to show $\sigma\star\xi.\pi\in\perp\!\!\!\perp_0$, that according to the reduction rule of σ becomes $\xi\star(\mathtt{succ})^j\mathtt{0}.\pi\in\perp\!\!\!\perp_0$. But since $(\mathtt{succ})^j\mathtt{0}.\pi\in\|\{(\mathtt{succ})^j\mathtt{0}\}\to F[p,\phi(j,p,\mathscr{H}(n)),\mathscr{H},n]\|_0\subseteq\bigcup_{i\in\mathbb{N}}\|\{(\mathtt{succ})^j\mathtt{0}\}\to F[p,\phi(j,p,\mathscr{H}(n)),\mathscr{H},n]\|_0=\|\forall^{\mathbb{N}}xF[p,\phi(x,p,\mathscr{H}(n)),\mathscr{H},n]\|_0$, by the hypothesis on ξ we have finished. □

Notation 3.71. *We set* $(p\Vdash\pm\mathscr{H}(n)):=\forall q(\mathscr{C}(pq)\to(q\Vdash\mathscr{H}(n))\to(p\Vdash\mathscr{H}(n)))$

Theorem 3.72. *Let $(P,C,1,\cdot)$ be a forcing structure satisfying the CCC. Then:*
there exist an application $(p,\mathscr{H})\in P\times\mathscr{P}(\Pi_c)^{\mathbb{N}\times P}\longrightarrow p^{\mathscr{H}}\in P$ *and there exist* $\delta_0,\delta_1,\delta_2\in QP$ *s.t.*
$\forall p\in P$, $\forall\mathscr{H}:\mathbb{N}\times P\longrightarrow\mathscr{P}(\Pi_c)$, $\delta_0\Vdash_0\mathscr{C}(p)\to\mathscr{C}(p^{\mathscr{H}})$ *and* $\delta_1\Vdash_0 p^{\mathscr{H}}\sqsubseteq p$ *and* $\delta_2\Vdash_0\forall^{\mathbb{N}}n(p^{\mathscr{H}}\Vdash\pm\mathscr{H}(n))$.

Idea of the proof. We define a formula of $\mathcal{F}_0$ by $\Psi[r,p,\mathscr{H}(n)]:=$
$(p\Vdash(\mathscr{H}(n)\to r=p))\wedge\forall^{\mathbb{N}}x(\neg F[p,\phi(x,p,\mathscr{H}(n)),\mathscr{H},n]\to$
$\forall^{\mathbb{N}}y\forall^{\mathbb{N}}z(y+z+1=x\to F[p,\phi(y,p,\mathscr{H}(n)),\mathscr{H},n])\to r=p\cdot\phi(x,p,\mathscr{H}(n)))$.
We use the following lemma, of which we omit the proof (see it in [Kriv08], "lemma 25").
Lemma.
The following formulas are realized by proof-like terms that do not depend on her parameters (that are p, $\mathscr{H}$ and n).

i): $\forall^{\mathbb{N}}n\exists r\Psi[r,p,\mathscr{H}(n)]$

ii): $\forall^{\mathbb{N}}n\forall q\forall r(\Psi(q,p,\mathscr{H}(n))\to\Psi(r,p,\mathscr{H}(n))\to q=r)$

iii): $\forall r(\Psi(r,p,\mathscr{H}(n)) \to r \sqsubseteq p)$

iv): $\forall r(\Psi(r,p,\mathscr{H}(n)) \to \mathscr{C}(p) \to \mathscr{C}(r))$

v): $\forall r(\Psi(r,p,\mathscr{H}(n)) \to (r \Vdash \pm \mathscr{H}(n))$.

We define now an application $\mathscr{Y} : \mathbb{N} \times P \longrightarrow \mathscr{P}(\Pi_c)$ by: $\mathscr{Y}(n,q) := \|\Phi[n,q,p,\mathscr{H}]\|_0$, where $\Phi[n,q,p,\mathscr{H}] = \forall Z((\forall^{\mathbb{N}} j \forall r \forall r'((r \not\Vdash Z^+(j)) \to \Psi(r',r,\mathscr{H}(j)) \to (r' \not\Vdash Z^+(j+1)))) \to (p \not\Vdash Z^+(0)) \to (q \not\Vdash Z^+(n)))$.
We use the following lemma, of which we omit the proof (see it in [Kriv08], "lemma 26").
Lemma.
The following formulas are realized by proof-like terms that do not depend on its parameters (that are p and $\mathscr{H}$).

i): $\forall q(q \not\Vdash \mathscr{Y}^+(0) \leftrightarrow q = p)$

ii): $\forall^{\mathbb{N}} \exists q(q \not\Vdash \mathscr{Y}^+(n))$

iii): $\forall^{\mathbb{N}} n \forall q \forall r((q \not\Vdash \mathscr{Y}^+(n)) \to (r \not\Vdash \mathscr{Y}^+(n)) \to q = r)$

iv): $\forall^{\mathbb{N}} \forall^{\mathbb{N}} \forall q \forall r((q \not\Vdash \mathscr{Y}^+(n)) \to (r \not\Vdash \mathscr{Y}^+(n+m)) \to (r \sqsubseteq q))$

v): $\forall^{\mathbb{N}} \forall q((q \not\Vdash \mathscr{Y}^+(n+1)) \to (q \Vdash \pm \mathscr{H}(n)))$

vi): $\mathscr{C}(p) \to \forall^{\mathbb{N}} \forall q((q \not\Vdash \mathscr{Y}^+(n)) \to \mathscr{C}(q))$

Since the forcing structure satisfies the CCC we can use the application **p**. One can verify that (see again [Kriv08]) that the application defined by $p^{\mathscr{H}} := \mathbf{p}(\mathscr{Y})$ has all the desired properties. □

Theorem 3.73. *Let $(P, C, 1, \cdot)$ be a forcing structure satisfying the CCC. Then:*
there exists an application $(p, \mathscr{H}) \in P \times \mathscr{P}(\Pi_c)^{\mathbb{N} \times P} \longrightarrow K_{p,\mathscr{H}} \in \mathscr{P}(\Pi_c)^{\mathbb{N}}$ and there exist $\theta, \delta \in QP$ s.t. $\forall p \in P$, $\forall \mathscr{H} : \mathbb{N} \longrightarrow \mathscr{P}(\Pi_c \times P)$ we have:
$(\theta, p^{\mathscr{H}}) \Vdash_1 \mathscr{H} \simeq K_{p,\mathscr{H}}$
$\delta \Vdash_0 \neg\mathscr{C}(p \cdot p^{\mathscr{H}}) \to \neg\mathscr{C}(p)$.

Proof. The existence of δ follows from δ_0 and δ_1 of theorem 3.72.
We set now $K_{p,\mathscr{H}}(n) := \|p^{\mathscr{H}} \Vdash \mathscr{H}(n)\|_0 = \|\forall q(\mathscr{C}(p^{\mathscr{H}} q)) \to q \not\Vdash \mathscr{H}^+(n)\|_0$.
We use the following lemma, of which we omit the proof (see it in [Kriv08], "lemma 28").
Lemma.
$\exists \theta_0, \theta_1 \in QP$ s.t. $\theta_0 \Vdash_0 p^{\mathscr{H}} \Vdash \forall^{\mathbb{N}} n(K_{p,\mathscr{H}} \to \mathscr{H}(n))$ and $\theta_1 \Vdash_0 p^{\mathscr{H}} \Vdash \forall^{\mathbb{N}} n(\mathscr{H}(n) \to K_{p,\mathscr{H}})$.
According to that lemma and theorem 3.54 we obtain:
$((\chi_{\forall^{\mathbb{N}} n(Y(n) \to X^+(n))})\theta_0, p^{\mathscr{H}}) \Vdash_1 \forall^{\mathbb{N}} n(K_{p,\mathscr{H}} \to \mathscr{H}(n))$ and
$((\chi_{\forall^{\mathbb{N}} n(X^+(n) \to Y(n))})\theta_1, p^{\mathscr{H}}) \Vdash_1 \forall^{\mathbb{N}} n(\mathscr{H}(n) \to K_{p,\mathscr{H}})$
(we recall that $\chi_{(.)}$ does not depend on the parameters of $(.)$).
Now, since by the adequation lemma the set of formulas realized by some proof-like terms is deductively closed, we have the thesis (taking the conjunction of the two formulas that we have just realized). □

Theorem 3.74. *Let $(P, C, 1, \cdot)$ be a forcing structure satisfying the CCC. Then:*
$\exists \theta_0 \in QP$ *s.t.* $\forall r \in P$, $(\theta_0, r) \Vdash_1 \forall X^+ \exists X (X^+ \simeq X)$.

Proof. We look for a θ_0 s.t. $\forall \mathscr{H} : \mathbb{N} \to \mathscr{P}(\Pi_c \times P)$, $(\theta_0, r) \Vdash_1 \neg \forall X \neg (\mathscr{H} \simeq X)$, i.e. $(\theta_0 \star \xi.\pi, r(qq')) \in \perp\!\!\!\perp_1$, with $(\xi, q) \Vdash_1 \forall X \neg (\mathscr{H} \simeq X)$ and $(\pi, q') \in \Pi_c \times P$. Then, let $\tau \in C(r(qq'))$.
By hypothesis on (ξ, q) we have $(\xi, q) \Vdash_1 \neg (\mathscr{H} \simeq K_{r(qq'), \mathscr{H}})$. Let's set $p := r(qq')$. Thanks to the theorem 3.73 then we get $(\xi \star \theta.\varpi, q(p^{\mathscr{H}} \cdot 1)) = (\xi, q) \star (\theta, p^{\mathscr{H}}).(\varpi, 1) \in \perp\!\!\!\perp_1$ for every $\varpi \in \Pi_c$.
Let $pp' \xrightarrow{\alpha} q(p'1)$ and $\upsilon \in C(pp^{\mathscr{H}})$. Then $(\alpha)\upsilon \in C(q(p^{\mathscr{H}} 1))$ and so $\chi' \star \xi.(\alpha)\upsilon.\theta.\varpi \succ \xi \star \theta.\varpi^{(\alpha)\upsilon} \in \perp\!\!\!\perp_0$.
So we obtain $\lambda y(((\chi')\xi)(\alpha)y)\theta \Vdash_0 \neg \mathscr{C}(pp^{\mathscr{H}})$. But from theorem 3.73 we have therefore $(\delta)\lambda y(((\chi')\xi)(\alpha)y)\theta \Vdash_0 \neg \mathscr{C}(p)$. So, since $\tau \in C(p)$, we have $\delta \star \lambda y(((\chi')\xi)(\alpha)y)\theta.\tau.\pi \in \perp\!\!\!\perp_0$ and so we can take $\theta_0 := \lambda x(\chi)(\delta)\lambda y(((\chi')x)(\alpha)y)\theta$. □

3.4 Ultrafilters and programs

The idea is now to choose a particular language, to prove that we can apply to it all the strong properties that we found in the previous technical subsections, and to finally realize the formula that formally translates the axiom of the ultrafilter. Of course, we will recall what this axiom says.

Definition 3.75. *We will consider the set of formulas $\mathcal{F}_0$ and $\mathcal{F}_1$ that we obtain taking as condition variable the second order variables of the form X^1 (with arity 1). We will indicate with $\wedge$ the binary function symbol $\cdot$ on the condition terms, and we add the binary function symbol $\setminus$. The condition terms are therefore given by the following syntax: $t^P := X \mid t^P \setminus t^P \mid t^P \wedge t^P$.*
We have the following languages:
the formulas of $\mathcal{F}_0$ are given by the syntax:

$$t \neq u \mid t^P \neq t^P \mid X(\vec{t}) \mid t^P \not\in X^+(\vec{t}) \mid t^P \not\in J^+(u^P) \mid A \to A \mid \mathscr{C}(t^P) \to A \mid \forall x A \mid \forall^{\mathbb{N}} x A \mid \forall X A \mid \forall X^+ A$$

the formulas of $\mathcal{F}_0^{usu}$ are given by the syntax:

$$t \neq u \mid X(\vec{t}) \mid A \to A \mid \forall x A \mid \forall^{\mathbb{N}} x A \mid \forall X A$$

the formulas of $\mathcal{F}_0^{\mathbb{N}}$ are given by the syntax:

$$t \neq u \mid X(\vec{t}) \mid A \to A \mid \forall^{\mathbb{N}} x A \mid \forall X A$$

the formulas of $\mathcal{F}_1$ are given by the syntax:

$$t \neq u \mid X(\vec{t}) \mid J(t^P) \mid X^+(\vec{t}) \mid \mathscr{C}(t^P) \to A \mid A \to A \mid \forall x A \mid \forall^{\mathbb{N}} x A \mid \forall X A \mid \forall X^+ A$$

the formulas of $\mathcal{F}_1^{\mathbb{N}}$ are given by the syntax:

$$t \neq u \mid X(\vec{t}) \mid J(t^P) \mid X^+(\vec{t}) \mid A \to A \mid \forall^{\mathbb{N}} x A \mid \forall X A \mid \forall X^+ A.$$

Note that this sets of formulas are included in the sets of formulas of the definition 3.12 and extended in the definition 4.1. In particular we can then interpret the formulas of $\mathcal{F}_0$ in the realizability algebra SR_0 and the formulas of $\mathcal{F}_1$ in the algebras $SR_1(P)$ for a certain forcing structure over P. It's precisely want we're going to do.

Remark 3.76. *The notion of models $\mathfrak{M}$for the formulas $\mathcal{F}_1^{\mathbb{N}}$ is that of the "habitual" models for a second order language, i.e. the data of a set M, of an interpretation of terms in M, of an interpretation of predicates of the form X and X^+ as subset of M, and of a family $J_{\mathfrak{M}}$ of subsets of M.*

We will build a forcing structure over the set $\mathscr{P}(\Pi_c)^{\mathbb{N}}$ and we will consider the SR_1-algebra associated.

Notation 3.77. *From now on we fix:*
$P := \mathscr{P}(\Pi_c)^{\mathbb{N}}$,
$1 : n \in \mathbb{N} \to \varnothing \in \mathscr{P}(\Pi_c)$ *(and we use the same name for the function $n \in \mathbb{N} \to \varnothing \in \mathscr{P}(\Pi_c \times P)$),*
$\wedge : \mathscr{P}(\Pi_c)^{\mathbb{N}} \times \mathscr{P}(\Pi_c)^{\mathbb{N}} \to \mathscr{P}(\Pi_c)^{\mathbb{N}}$, $(X \wedge Y)(n) := \neg(X(n) \to \neg Y(n)) := (X(n)^{\perp}.Y(n)^{\perp}.\Pi_c)^{\perp}.\Pi_c$,
$\mathbb{C}(X) := \forall m(Nat\{m\} \to \neg\forall n(Nat\{n\} \to \neg X(n+m)))$.
We interpret the terms $t^P \setminus u^P$ by $[\![t^P]\!] \setminus [\![u^P]\!]$, where $\setminus : P \times P \longrightarrow P$,
$(X \setminus Y)(n) = (X(n)^{\perp}.(Y(n).\Pi_c)^{\perp}.\Pi_c)^{\perp}.\Pi_c$.

Remark 3.78. *On a: $\mathfrak{M} \models \mathbb{C}(X)$ iff $\mathfrak{M} \models \forall^{\mathbb{N}} m \exists^{\mathbb{N}} n X(n+m)$ iff $X_{\mathfrak{M}} \cap \mathbb{N}$ is infinite.*
Indeed, let $\mathfrak{M} \models \forall^{\mathbb{N}} m \exists^{\mathbb{N}} n X(n+m)$ (the other implication are clear); in particular $\exists f : \mathbb{N} \to \mathbb{N}$ s.t. $\forall m \in \mathbb{N}, m + f(m) \in X$. If by absurd $X_{\mathfrak{M}} \cap \mathbb{N}$ is finite then also $\{m + f(m)\}_{m \in \mathbb{N}}$ is finite and so $\exists A \subseteq \mathbb{N}$ infinite, $\exists k \in \mathbb{N}$ s.t. $a + f(a) = k$, $\forall a \in A$. But then $0 \leq f(a) = k - a$, i.e. $k \geq a, \forall a \in A$. Since A is infinite and k finite, this is not possible.

Proposition 3.79. *$(\mathscr{P}(\Pi_c)^{\mathbb{N}}, C, 1, \wedge)$ is a forcing structure,*
where $C : \mathscr{P}(\Pi_c)^{\mathbb{N}} \to \mathscr{P}(\Lambda_c)$, $C(A) := \{\tau \in \Lambda_c \ s.t. \ \tau \Vdash_0 \mathbb{C}(A)\}$.

Proof. We have to find the forcing combinators $\alpha_0, \ldots, \alpha_4$, i.e. some proof-like term s.t. $\forall \tau \in \Lambda_c, \forall A, B, C \in \mathscr{P}(\Pi_c)^{\mathbb{N}}$ we have:
$\tau \Vdash_0 \mathbb{C}(A \wedge B) \Rightarrow (\alpha_0)\tau \Vdash_0 \mathbb{C}(A)$; $\tau \Vdash_0 \mathbb{C}(A \wedge B) \Rightarrow (\alpha_1)\tau \Vdash_0 \mathbb{C}(B \wedge A)$
$\tau \Vdash_0 \mathbb{C}(A) \Rightarrow (\alpha_2)\tau \Vdash_0 \mathbb{C}(A \wedge A)$; $\tau \Vdash_0 \mathbb{C}(A \wedge (B \wedge C)) \Rightarrow (\alpha_3)\tau \Vdash_0 \mathbb{C}((A \wedge B) \wedge C)$
$\tau \Vdash_0 \mathbb{C}(A) \Rightarrow (\alpha_4)\tau \Vdash_0 \mathbb{C}(A \wedge 1)$.
So it is enough to find some α_i s.t.
$\alpha_0 \Vdash_0 \forall X \forall Y \ \mathbb{C}(X \wedge Y) \to \mathbb{C}(X)$; $\alpha_1 \Vdash_0 \forall X \forall Y \ \mathbb{C}(X \wedge Y) \to \mathbb{C}(Y \wedge X)$
$\alpha_2 \Vdash_0 \forall X \ \mathbb{C}(X) \to \mathbb{C}(X \wedge X)$; $\alpha_3 \Vdash_0 \forall X \forall Y \forall Z \ \mathbb{C}(X \wedge (Y \wedge Z)) \to \mathbb{C}((X \wedge Y) \wedge Z)$
$\alpha_4 \Vdash_0 \forall X \ \mathbb{C}(X) \to \mathbb{C}(X \wedge 1)$.
Now, one can easily check that
$H := \lambda f^{\forall x(Xx \to X'x)} \lambda u^{\mathbb{C}(X)} \lambda m^{Nat\{m\}} \lambda h^{\forall n(Nat\{n\} \to X'(m+n) \to \perp)}((u)m)\lambda n^{Nat\{n\}} \lambda x^{X(m+n)}((h)n)(f)x$
is a derivation of $\forall X \forall X'((\forall x(Xx \to X'x)) \to (\mathbb{C}(X) \to \mathbb{C}(X')))$, and so

$H \Vdash_0 \forall X \forall X'((\forall x(Xx \to X'x)) \to (\mathbb{C}(X) \to \mathbb{C}(X')))$;
therefore we can find $\beta_i \in QP,\ i = 0, \dots, 4$ s.t.
$\beta_0 \Vdash_0 \forall X \forall Y \forall x((X \wedge Y)x \to Xx)$; $\beta_1 \Vdash_0 \forall X \forall Y \forall x((X \wedge Y)x \to (Y \wedge Y)x)$
$\beta_2 \Vdash_0 \forall X \forall x((X)x \to (X \wedge X)x)$; $\beta_3 \Vdash_0 \forall X \forall Y \forall Z \forall x((X \wedge (Y \wedge Y))x \to ((X \wedge Y) \wedge Z)x)$
$\beta_4 \Vdash_0 \forall X \forall x(Xx \to 1(x))$.
One can easily verify that the following β_i works: $\beta_0 = \texttt{callcc}$, $\beta_1 = \lambda t \lambda u(t) \lambda a \lambda b(u)ba$, $\beta_2 = \lambda x \lambda f(f)xx$, $\beta_3 =$, $\beta_4 = \beta_2$. □

Theorem 3.80. *The forcing structure $(\mathscr{P}(\Pi_c)^{\mathbb{N}}, C, 1, \wedge)$ satisfies the Countable Chain Condition.*

Idea of the proof. We need to build an application $\mathbf{p} : \mathscr{P}(\Pi_c \times \mathscr{P}(\Pi_c)^{\mathbb{N}})^{\mathbb{N}} \to \mathscr{P}(\Pi_c)^{\mathbb{N}}$ and two proof-like terms **ccd** and **ccd'** s.t. $\forall \mathscr{H} : \mathbb{N} \to \mathscr{P}(\Pi_c \times \mathscr{P}(\Pi_c)^{\mathbb{N}})$ on a:
ccd $\Vdash_0 \forall^{\mathbb{N}} j \forall Y(Y \not\in \mathscr{H}(j) \to \mathbf{p}(\mathscr{H}) \sqsubseteq Y)$
ccd' $\Vdash_0 AtMost \to AtLeast \to NonTriv \to Decr \to \mathbb{C}(\mathbf{p}(\mathscr{H}))$,
where $AtMost := \forall^{\mathbb{N}} j \forall Y \forall Z(Y \not\in \mathscr{H}(n) \to Z \not\in \mathscr{H}(n) \to Y = Z)$,
$AtLeast := \forall^{\mathbb{N}} j \exists Y(Y \not\in \mathscr{H}(n))$,
$NonTriv := \forall^{\mathbb{N}} j \forall Y(Y \not\in \mathscr{H}(n) \to \mathbb{C}(Y)$,
$Decr := \forall^{\mathbb{N}} m \forall^{\mathbb{N}} j \forall Y \forall Z(Y \not\in \mathscr{H}(j) \to Z \not\in \mathscr{H}(j+m) \to Z \sqsubseteq Y)$.
The idea beyond is the following:
the hypotheses of the formula that has to be realized by **ccd'** say that the sets $\tilde{\mathcal{H}}_n \subseteq \mathscr{P}(\Pi_c)^{\mathbb{N}}$ induced by $\mathscr{H}$ as explained in remark 4.11 are of the form $\tilde{\mathcal{H}}_n = \{p_n\}$, and the sequence $\{p_n\}_{n \in \mathbb{N}}$ is decreasing in the order $\sqsubseteq$. That means that, if we call A_i the interpretation in the model $\mathfrak{M}$ of the predicate p_i we get that: $(A_{i+1} \setminus A_i)_{\mathscr{M}} \cap \mathbb{N}$ finite $\forall i \in \mathbb{N}$, and its elements are non trivial, i.e. $(A_i)_{\mathscr{M}} \cap \mathbb{N}$ infinite $\forall i \in \mathbb{N}$; we define a function (à priori partial) $f : \mathbb{N} \longrightarrow \mathbb{N}$ by $f(j) := min\{n \in \mathbb{N}\ s.t.\ n > j\ and\ n \in \bigcap_{i \le j} A_i\}$. Since the $(A_i)_{\mathscr{M}} \cap \mathbb{N}$ are infinite, then also the image $f\mathbb{N}$ of f is infinite, and f is total. We set $\mathbf{p}(\mathscr{H}) := f\mathbb{N}$.
In order to do that, we shall write the relation $n = f(j)$ as a formula $\phi(j, n, \mathscr{H})$ and we shall set $\mathbf{p}(\mathscr{H})(n) = \|\exists^{\mathbb{N}} j \phi(j, n, \mathscr{H})\|_0$. The formula ϕ is defined by:
$\phi(j, n, \mathscr{H}) := j < n \wedge B(j, n, \mathscr{H}) \wedge \forall^{\mathbb{N}} k(j < k < n \to \neg B(j, k, \mathscr{H}))$,
where $B(j, n, \mathscr{H}) = \forall^{\mathbb{N}} i(i \le j \to A(i, n, \mathscr{H}))$ and $A(i, n, \mathscr{H}) = \forall Y(Y \not\in \mathscr{H}(i) \to Yn)$.
Now, for the adequation lemma, to prove that a certain formula is realized by a proof-like term, it is enough to prove the formula under some hypotheses already realized by proof-like terms. It's what it's done in [Kriv08], pages 28-29, in order to realize the formulas $\forall^{\mathbb{N}} j \forall Y(Y \not\in \mathscr{H}(j) \to \mathbf{p}(\mathscr{H}) \sqsubseteq Y)$ and $\mathbb{C}(\mathbf{p}(\mathscr{H}))$, which ends the proof. □

Notation 3.81. *We define the following $\mathcal{F}_1$ formulas:*

- $(X^+ \sqsubseteq Y^+) := \exists^{\mathbb{N}} l \forall^{\mathbb{N}} m(l \le m \to X^+(m) \to Y^+(m))$, *for X, Y second order variables of arity 1 (and $(l \le m) := \exists^{\mathbb{N}} k(l + k = m)$).*
- $\mathscr{J}\{X^+\} := \forall Y^1(Y \simeq X^+ \to J(Y)) = \forall Y^1((\forall^{\mathbb{N}} n(Yn \leftrightarrow X^+ n)) \to J(Y))$.

We observe that we can show that $\sqsubseteq$ is a preorder on M, i.e.
$\vdash A \sqsubseteq A$ and $\vdash A \sqsubseteq B \to B \sqsubseteq C \to A \sqsubseteq C$. Indeed for the transitivity (the reflexivity is clear) we can follow the idea that, for three subsets A, B, C of M, if $\exists l \in \mathbb{N}$ s.t. $\forall m \ge l$,

$m \in A \cap \mathbb{N} \Rightarrow m \in B \cap \mathbb{N}$ *and if* $\exists l' \in \mathbb{N}$ *s.t.* $\forall m \geq l'$, $m \in B \cap \mathbb{N} \Rightarrow m \in C \cap \mathbb{N}$, *then* $\exists l'' = max\{l, l'\} \in \mathbb{N}$ *s.t.* $\forall m \geq l''$, $m \in A \cap \mathbb{N} \Rightarrow m \in C \cap \mathbb{N}$.
By the adequation lemma the formula " $\sqsubseteq$ preorder on M*" is realized.*

The formula $X \sqsubseteq Y$ *is equivalent to* $\exists^{\mathbb{N}} l \forall^{\mathbb{N}} m(l \leq m \to X(m) \to Y(m))$ *and also to* $\neg\mathbb{C}(X \setminus Y)$.

Theorem 3.82. *i):* $\exists \theta \in QP\ s.t.\ (\theta, 1) \Vdash_1 \forall X^+ \forall Y^+ (X^+ \simeq Y^+ \to \mathscr{J}\{X^+\} \to \mathscr{J}\{Y^+)\}$

ii): $\exists \theta \in QP\ s.t.\ (\theta, 1) \Vdash_1 \forall X \forall X^+ (X \simeq X^+ \to (J(X) \leftrightarrow \mathscr{J}\{X^+\}))$

Proof. i): We note that by definition of $\mathscr{J}\{X^+\}$ we can easily show that $\vdash \forall X^+ \forall Y^+ \forall Y (X^+ \simeq Y^+ \to \mathscr{J}\{X^+\} \to y \simeq Y^+ \to J(Y))$. So clearly $\forall X^+ \forall Y^+ (X^+ \simeq Y^+ \to \mathscr{J}\{X^+\} \to \mathscr{J}\{Y^+)\}$ and so by the adequation lemma we get the thesis.

ii): The implication $\forall X \forall X^+ (X \simeq X^+ \to \mathscr{J}\{X^+\} \to J(X)$ is clear by definition of $\mathscr{J}\{X^+\}$; for the implication $\forall X \forall X^+ (X \simeq X^+ \to J(X) \to \mathscr{J}\{X^+\})$ we can reason as follows: by theorem 3.63 (v) we have that $\forall X \forall Y (J(X) \to Y \sqsubseteq X \to J(Y))$ is realized by a proof-like term. So, since the relation $\subseteq$ is stronger than $\sqsubseteq$, (and again by adequation lemma) we obtain also that $\forall X \forall Y (J(X) \to X \simeq Y \to J(Y))$ is realized by a proof-like term. But fixed X^+, we have:
$\forall X \forall Y (J(X) \to X \simeq Y \to J(Y)) \vdash \forall X (X \simeq X^+ \to J(X) \to \forall Y (Y \simeq X^+ \to J(Y)))$.
Since $\forall Y (Y \simeq X^+ \to J(Y)) = \mathscr{J}(X^+)$, by adequation lemma we get the thesis.

□

Lemma 3.83. $\exists \theta \in QP\ s.t.\ (\theta, 1) \Vdash_1 \forall X (J(1 \setminus X) \leftrightarrow \neg J(X))$.

Proof. Easy, see [Kriv08], "lemma 37". □

Theorem 3.84. *i):* $\exists \theta \in QP\ s.t.\ (\theta, 1) \Vdash_1 \forall X^+ \forall Y^+ (Y^+ \sqsubseteq X^+ \to \mathscr{J}\{X^+\} \to \mathscr{J}\{Y^+\})$.

ii): $\exists \theta \in QP\ s.t.\ (\theta, 1) \Vdash_1 \neg \mathscr{J}[1]$

iii): $\exists \theta \in QP\ s.t.\ (\theta, 1) \Vdash_1 \forall X^+ (\mathscr{J}[1 \setminus X^+] \leftrightarrow \neg \mathscr{J}\{X^+\})$

iv): $\exists \theta \in QP\ s.t.\ (\theta, 1) \Vdash_1 \forall X^+ \forall Y^+ (\neg \mathscr{J}\{X^+\} \to \mathscr{J}\{X^+ \wedge Y^+\} \to \mathscr{J}\{Y^+\})$.

Proof. i): We can verify that $x : F_0, y : F_1, z : F_2 \vdash t\{x_0, x_1, x_2\} : F$,
where F is the formula of the theorem that we want to realize,
$F_0 := \forall X \forall Y (J(X) \to Y \sqsubseteq X \to J(Y))$, $F_1 := \forall X^+ \exists X (X^+ \simeq X)$ and
$F_2 := \forall X \forall X^+ (X \simeq X^+ \to (J(X) \leftrightarrow \mathscr{J}\{X^+\}))$. Now, theorem 3.63(v) says that F_0 is realized by a $(\theta_0, 1)$, where $\theta_0 \in QP$; theorem 3.72 says that F_1 is realized by a $(\theta_1, 1)$, where $\theta_1 \in QP$; theorem 3.82 (ii) says that F_2 is realized by a $(\theta_2, 1)$, where $\theta_2 \in QP$. But then by adequation lemma we have: $QP \ni t[(\theta_0, 1)/x_0, (\theta_1, 1)/x_1, (\theta_2, 1)/x_2] = (t^*[\theta_0/x_0, \theta_1/x_1, \theta_2/x_2], 1) \Vdash_0 F$, that is the wanted result.

ii): by theorem 3.63 (i) we have that $\neg J[1]$ is realized by a proof-like term. But then theorem 3.82 (ii) tells us that also $\neg \mathscr{J}[1]$ is realized by a proof-like term.

iii): by theorem 3.72 the formula $\exists X \simeq X^+$ is realized by a proof-like term, and so $1 \setminus X \simeq 1 \setminus X^+$ too. But then if $\vdash \mathscr{J}[1 \setminus X^+]$, by theorem 3.82 (ii) we obtain $\vdash J[1 \setminus X]$, from where we get $\vdash \neg J(X)$. Then again by theorem 3.82 (ii) we have $\vdash \neg \mathscr{J}(X^+)$; so we have showed that $\mathscr{J}[1 \setminus X^+] \to \neg \mathscr{J}\{X^+\}$ is derivable from certain realized hypotheses, and so thanks to the adequation lemma she is realized too. The other implication is similar.

iv): we can easily reason in the same way using the point (iii) of theorem 3.63.

□

Definition 3.85. *Let N be a Boolean algebra with order $\leq$ and infima $\inf\{.,.\}$.*

A set $\mathcal{U} \subseteq \mathscr{P}(N)$ is called non trivial ultrafilter on N *iff* $\forall A, B \subseteq N$,

1. $\varnothing \notin \mathcal{U}$
2. $\mathcal{U} \ni A \leq B \Rightarrow B \in \mathcal{U}$
3. $A, B \in \mathcal{U} \Rightarrow \inf\{A, B\} \in \mathcal{U}$
4. $A \in \mathcal{U} \Leftrightarrow N \setminus A \notin \mathcal{U}$.

A set $\mathcal{I} \subseteq \mathscr{P}(N)$ is called non trivial maximal ideal on N *iff $\mathcal{I} = \mathscr{P}(N) \setminus \mathcal{U}$, for an ultrafilter $\mathcal{U}$, that amounts to the same as $\mathcal{I} = \{N \setminus A\}_{A \in \mathcal{U}}$, that is again the same as: $\forall A, B \subseteq N$,*

1. $N \notin \mathcal{I}$
2. $A \leq B \in \mathcal{I} \Rightarrow A \in \mathcal{I}$
3. $\inf\{A, B\} \in \mathcal{I} \Rightarrow$ ou $A \in \mathcal{I}$ ou $B \in \mathcal{I}$
4. $A \in \mathcal{I} \Leftrightarrow N \setminus A \notin \mathcal{I}$.

We call ultrafilter axiom *the statement: "there exist a non trivial ultrafilter on $\mathbb{N}$",that is therefore the same as "there exist a non trivial maximal ideal $\mathcal{I}$ on $\mathbb{N}$".*

We will (give the idea of how to) write a formula that expresses in our formal language that axiom and we will show that she is realized in a realizability interpretation for SR_1.

Proposition 3.86. *In a realizability model $\mathfrak{M}$ (that satisfies, therefore, the formulas (i)-(iv) of the theorem 3.84) let's call $\mathscr{J} := \{D \subseteq M \text{ s.t. } \mathfrak{M} \models \mathscr{J}[D]\}$ (we use the same symbol $\mathscr{J}$ but there is no confusion). We have that:*
$\mathcal{I} := \{A \cap \mathbb{N}\}_{A \in \mathscr{J}} \subseteq \mathscr{P}(\mathbb{N})$ is a non trivial maximal ideal on $\mathbb{N}$ (it satisfies the properties 1-4 of the definition 5.11 with order the inclusion $\subseteq$, that is the formula of the notation 4.9).
Note that so we have also that $\neg\mathcal{I} := \{\mathbb{N} \setminus A\}_{A \in \mathcal{I}}$ is a non trivial ultrafilter on $\mathbb{N}$.

Proof. We first observe that for $A, B \subseteq M$ we have: $A \wedge B \sqsubseteq A \cap B$. In fact since $A \wedge B = \inf_{\sqsubseteq}\{A, B\}$ we have $A \wedge B \sqsubseteq A$ and $A \wedge B \sqsubseteq B$. That is to say $(A \wedge B) \cap \mathbb{N}_{\geq l} \subseteq A$ and

$(A\wedge B)\cap\mathbb{N}_{\geq l'}\subseteq B$, for certain $l,l'\in\mathbb{N}$. But then for $t\geq\max\{l,l'\}$ we have $(A\wedge B)\cap\mathbb{N}_{\geq t}\subseteq A\cap B$, i.e $A\wedge B\sqsubseteq A\cap B$.
Now:

1): if $M\cap\mathbb{N}=\mathbb{N}\in\mathcal{I}$ then $M\in\mathscr{J}$ (by definition of $\mathcal{I}$), which is in contradiction with the formula (ii) of theorem 3.84 (since we can realize the formula $\forall x1(x)$, that says "$1=M$"). So $\mathbb{N}\notin\mathcal{I}$.

2): if $A\subseteq B\cap\mathbb{N}\in\mathcal{I}$ then $A\sqsubseteq B$ (because $\subseteq$ is stronger than $\sqsubseteq$) and $B\in\mathscr{J}$ (by definition of $\mathcal{I}$). So by the formula (i) of the theorem 3.84 $A\in\mathscr{J}$, from where by definition of $\mathcal{I}$ (and since $A\subseteq\mathbb{N}$) $A=A\cap\mathbb{N}\in\mathcal{I}$.

3): if $A\notin\mathcal{I}$ and $A\cap B\in\mathcal{I}$ for $A,B\subseteq\mathbb{N}$, then, since $A=A\cap\mathbb{N}$ and $A\cap B=(A\cap B)\cap\mathbb{N}$, we have (by definition of $\mathcal{I}$) $A\notin\mathscr{J}$ and $A\cap B\in\mathscr{J}$. Now, since $A\wedge B\sqsubseteq A\cap B$, by the formula (i) of theorem 3.84 we have $A\wedge B\in\mathscr{J}$. Then, by the formula (iv) of theorem 3.84, $B\in\mathscr{J}$ and so $B=B\cap\mathbb{N}\in\mathcal{I}$.

4): if $A\cap\mathbb{N}\in\mathcal{I}$ then $A\in\mathscr{J}$, from where by the formula (iii) of theorem 3.84, $M\setminus A\notin\mathscr{J}$. Then if $\mathcal{I}\ni\mathbb{N}\setminus A=\mathbb{N}\cap(M\setminus A)$, we would have $M\setminus A\in\mathscr{J}$, contradiction. So $\mathbb{N}\setminus A\notin\mathcal{I}$. Reciprocally if for $A\subseteq\mathbb{N}$ we have $(M\setminus A)\cap\mathbb{N}=\mathbb{N}\setminus A\notin\mathcal{I}$, then (by definition of $\mathcal{I}$) $M\setminus A\notin\mathscr{J}$ and so by the formula (iii) of theorem 3.84 $A\in\mathscr{J}$. Therefore $A=A\cap\mathbb{N}\in\mathcal{I}$.

□

Notation 3.87. *Let* $\mathrm{Ide}_{\mathcal{I},\subseteq}$ *the formula of* $\mathcal{F}_1^{\mathbb{N}}$ *that translates, for a certain predicate* $\mathcal{I}$*, the conditions 1-4 of the definition 5.11 of non trivial maximal ideal (where we indicate* N *with* 1, $\inf$ *with* $\cap$ *and* $\leq$ *with* $\subseteq$*), i.e. the conjunction of the formula of the theorem 3.84 (with* $\mathcal{I}$ *instead of* $\mathscr{J}$*). Then we can express the fact that* $\mathcal{I}$ *is a non trivial maximal ideal on* $\mathbb{N}$ *by the formula* AU *of* $\mathcal{F}_1^{\mathbb{N}}$ *given by the conjunction of the following formulas:* $\forall^{\mathbb{N}}x\ 1(x)$ *(that says* $1=\mathbb{N}$*),*
"$\setminus$ is the set's subtruction", "$\subseteq$ is an order", "$\cap$ is an inf for $\subseteq$", $\mathrm{Ide}_{\mathcal{I},\subseteq}$.
We will realize that formula.

Corollary 3.88. AU *is realized in* SR_1.

Proof. We can easily realize the formulas $\forall^{\mathbb{N}}x\ 1(x)$, "$\setminus$ is the sets' subtraction", "$\subseteq$ is an order" and "$\cap$ is an inf for $\subseteq$". And more, writing formally the proof of the proposition 5.12 we see that $\mathrm{Ide}_{\mathcal{I},\subseteq}$ is a consequence of $\forall x\ 1(x)$ (which says $1=M$), "$\setminus$ is the sets' subtraction", $\mathrm{Ide}_{\mathscr{J},\sqsubseteq}$, "$\sqsubseteq$ is a preorder" and "$\wedge$ is an inf for $\sqsubseteq$". Now, we can easily realize those formulas, and so by the adequation lemma we realize also the formula $\mathrm{Ide}_{\mathcal{I},\subseteq}$, that ends the proof. □

Now, in a sense[51], all the work of the previous 30 and more pages has been done to achieve the following result. We will comment it in a moment.

[51] In a sense, but not in all the senses: even if the aim of our journey in realizability was, from the beginning, to quickly present this theory for the case of $PA2$ and then to give as an example the realization of the ultrafilter axiom, introducing realizability for the forcing and studying them is interesting and has an important value by itself. It simply was not the central aim of our work.

Theorem 3.89. *Let* $ACD :=$ axiom of dependent choice, *i.e.*
$ACD = \forall X \exists Y (H\{X,Y\} \to \exists Z \forall k (Nat\{k\} \to H[Z(k,y)/Xy, Z(k+1,y)/Yy]))$
and let $A \in$ PA2. *We have:*
$x : AU, y : ACD \vdash t\{x,y\} : A \quad \Rightarrow \quad \exists \theta \in QP$ *s.t.* $\theta \Vdash_0 A$.

Proof. From $x : AU, y : ACD \vdash t\{x,y\} : A$ we have $x : AU \vdash \lambda yt : ACD \to A$, and since by corollary 3.88 we have a $\theta \in QP$ s.t. $(\theta, 1) \Vdash_1 AU$, by the adequation lemma we get $((\lambda yt)^*[\theta/x], 1) = sub_1(\lambda yt, ((\theta,1),x)) \Vdash_1 ACD \to A$; so by theorem 3.57 (applicable because $ACD, A \in \text{PA2} \subseteq \mathcal{F}_0^{usu}$), we have $(\chi^+_{ACD \to A})(\lambda yt)^*[\theta/x] \Vdash_0 \mathbb{C}(1) \to ACD \to A$, from where, as there are some $\xi, \eta \in QP$ s.t. $\xi \Vdash_0 \mathbb{C}[1]$ (for example $\xi = \lambda a \lambda b(b)aa$) and $\eta \Vdash_0 ACD$ (see [Kriv04], pag. 20,21 for the proof), we finally obtain $(((\chi^+_{ACD \to A})(\lambda yt)^*[\theta/x])\xi)\eta \Vdash_0 A$. □

We can also prove that the ultrafilter $\neg\mathcal{I}$ is selective:

Definition 3.90. *A ultrafilter* $\mathcal{U} \subseteq \mathscr{P}(\mathbb{N})$ *on* $\mathbb{N}$ *is said to be* selective *iff*
for every partition $\{\mathcal{P}_i\}_{i \in I}$ *(*I *is just a set of indexes) of* $\mathbb{N}$ *s.t.* $\forall i \in I$, $\mathcal{P}_i \notin \mathcal{U}$, *we have:*
$\exists U \in \mathcal{U}$ *s.t.* $\forall i \in I$, $U \cap P_i$ *is a singleton.*

Notation 3.91. *We fix the following formulas (here* Z *will be a binary predicate symbol and* X *and* Y *unary):*
$Part\{Z\} := \forall^{\mathbb{N}} j \exists^{\mathbb{N}}! m Z(m,j)$
$Prem\{x,X,Z\} := \exists^{\mathbb{N}}(Xj \wedge Z(m,j) \wedge \forall^{\mathbb{N}} i < j, \neg(Xi \wedge Z(m,j)))$
$R\{X,Z\} := \forall^{\mathbb{N}} \exists^{\mathbb{N}}(Xj \wedge \forall^{\mathbb{N}} m \leq l, \neg Z(m,j))$
$Part^+\{Z^+\} := Part\{Z^+\} \wedge \forall Y^+(\forall^{\mathbb{N}} m((\forall^{\mathbb{N}} j(Y^+ \leftrightarrow Z^+(m,j))) \to \mathscr{J}(Y^+)))$.

Remark 3.92. *The meaning of those formula is the following:*

- *if* $\mathfrak{M} \models Part\{Z\}$ *then it is naturally induces a partition* $\mathcal{P}_Z = \{f_Z^{-1}(m)\}_{m \in \mathbb{M}}$ *of* $\mathbb{N}$, *where* $f_Z : \mathbb{N} \longrightarrow \mathbb{N}$ *is the function naturally induced by* $Z_{\mathfrak{M}}$,
  i.e. $f(j) =$ *the unique m s.t.* $(m,j) \in Z_{\mathfrak{M}}$.
  And reciprocally if $\mathcal{P}$ *is a partition of* $\mathbb{N}$ *then, said* $\mathcal{S}$ *a system of representatives for* $\mathcal{P}$, *we have* $\mathcal{P} = \mathcal{P}_Z$, *where* $Z_{\mathfrak{M}} = \{(m,j)\ s.t.\ m \in \mathcal{S}\ and\ m \sim_{\mathcal{P}} j\}$, *where* $\sim_{\mathcal{P}}$ *is the equivalence relation induced by* $\mathcal{P}$.
- *if* $\mathfrak{M} \models Prem\{j,X,Z\}$ *then* $j = min\{X_{\mathfrak{M}} \cap [j]_Z\}$, *where* $[j]_Z$ *is the equivalence class of* j *induced by* f_Z, *and the same for the reciprocate.*
- *if* $\mathfrak{M} \models R\{X,Z\}$ *then* $X_{\mathfrak{M}}$ *meets an infinity of elements of* $\mathcal{P}_Z$, *and the same for the reciprocate.*
- *if* $\mathfrak{M} \models Part^+\{Z^+\}$ *then* $\mathcal{P}_{Z^+}$ *is a partition for which no element is in the ultrafilter* $\mathscr{P}(\mathbb{N}) \setminus \mathscr{J}_{\mathfrak{M}}$.

Theorem 3.93. $\exists \theta \in QP$ *s.t.*
$(\theta, 1) \Vdash_1 \forall Z^+(Part^+\{Z^+\} \to \neg\forall X^+ \forall Y^+(\forall^{\mathbb{N}} j(Y^+ j \leftrightarrow Prem[j, X^+, Z^+]) \to \mathscr{J}\{Y^+\}))$.

Proof. The method is always to derive this formula from some formulas that are realized by proof-like terms. See the details in [Kriv08] pages 32-33. □

Remark 3.94. *If we explicit the negation in the formula, we see that in the theorem we have just realized a formula that says that the ultrafilter (associated to $\mathscr{J}$ as we saw before) is selective. We note that then in the theorem 3.89 we can also suppose that AU is a formula that express the fact that there exist a non trivial selective ultrafilter on $\mathbb{N}$.*

As we already mentioned, λ_c-calculus allows the extension of the typing system to all classical logic's proofs. But classical logic is not enough to encode "all" mathematics, which uses also axioms in its various theories. Typing systems can't help us in that direction, since axioms have no proof.
The force of Krivine's realizability is then that it makes it possible to investigate the computational content of external axioms, and then, by typing, making proofs induce programs even when those proofs use axioms.
For instance, in [Kriv04] it is shown how every theorem in analysis (ZF + dependent choice) is justified (read: realized) by a program. Or also, thanks to what we just did we are able to justify the axiom of the ultrafilter by a program. And all becomes even more astonishing when one realizes that nowadays it is possible (of course only in a theoretical way!) to associate a program with all set theory ZF + dependent axiom of choice: and we recall that almost all mathematics can be encoded in ZF + dependent axiom of choice[52] This is clearly interesting by itself, particularly because of the beautiful theory which is classical realizability, and has clearly a significant value from a foundational and from a philosophical point of view[53]
The specification problem is nevertheless still far from having a general satisfactory solution: for example we haven't spoken about what the program that exists for the ultrafilter axiom actually does. And, more important, it is not yet clear what does the program for the axiom of dependent choice do. For example the behaviour of the axiom of countable choice (which is weaker then dependent choice, in turn weaker than the general axiom of choice) is to perform a sort of file updating (see [Kriv04]).

We cannot conclude this chapter without this quote of J.-L. Krivine:

> *My conclusion comes straightforward, after all that I told you: I think that with the developments of the proof/programs correspondence, proof theory has become the most interesting domain in mathematical logic; well, not only of mathematical logic, but in fact of all mathematics. But it's not precisely what I wanted to say: I meant the most interesting scientific domain among all disciplines.*
> J.-L. Krivine, "À propos de la théorie des démonstrations (du programme de Hilbert aux programmes tout court)".
> Colloque en l'honneur de René Cori, 2014
> (my translation from french).

[52] In many cases when in current mathematics we use the axiom of choice, we could have instead used some weaker form of it

[53] It seems that Krivine thinks that if one wrote all the programs that a mathematician - via realizability - produces, one would obtain something like the core of an operating system...

4 Going beyond the proof/programs paradigm

We see that in Tarski semantic, which is the level of truth, negation has the status of *refutation*: $\mathfrak{M} \models \neg A \Leftrightarrow \mathfrak{M} \not\models A$. In particular, $\exists \pi : (\vdash A) \Leftrightarrow \nexists \mathfrak{M}$ s.t. $\mathfrak{M} \models \neg A$. Proofs and models are different *kinds* of objects (the former finite and syntactic, the latter infinite and semantic), and there is no interaction at all between them, since the existence of one excludes the existence of the other. That's it.

As soon as we leave the level of truth and we place ourselves in the deeper level of the proof/program level, we find that negation has a richer significance. In fact, let's take the Brower-Heyting-Kolmogorov interpretation of proofs, which is the informal[54] definition of proofs behind intuitionistic logic; the idea is the following:
$\pi : (\vdash A_1 \wedge A_2)$ means $\pi = (\pi_1, \pi_2)$ with $\pi_i : (\vdash A_i)$, $i = 1, 2$,
$\pi : (\vdash A_1 \vee A_2)$ means $\pi = (\pi', i)$ with $\pi' : (\vdash A_i)$, for some $i = 1, 2$,
$\pi : (\vdash B \to A)$ means π is an algorithm taking a proof $\rho : (\vdash B)$ and returning a proof $\pi(\rho) : (\vdash A)$.
Now let's say we want to check that a given proof π is really a proof of a given formula, by doing multiple tests on π. If it passes all of them then it is ok, if one of them fails then we can conclude that it was not a proof of the formula.

1. if the formula is $A_1 \wedge A_2$ a test consists in either a test that π is a proof of A_1, either a test that π is a proof of A_2
2. if the formula is $A_1 \vee A_2$ a test consists in a couple of tests, one for testing that π is a proof of A_1, the other for testing that π is a proof of A_2
3. if the formula is $B \to A$ a test consists in a couple of a proof $\rho : (\vdash B)$ and a test that $\pi(\rho)$ is a proof of A.

In particular, setting Proofs(A) the set of all the proofs of A and Tests(A) the set of all the tests for A, we have:

$$\mathrm{Tests}(\mathrm{A}_1 \wedge \mathrm{A}_2) = \mathrm{Tests}(\mathrm{A}_1) \cup \mathrm{Tests}(\mathrm{A}_2)$$

$$\mathrm{Tests}(\mathrm{A}_1 \vee \mathrm{A}_2) = \mathrm{Tests}(\mathrm{A}_1) \times \mathrm{Tests}(\mathrm{A}_2)$$

$$\mathrm{Tests}(\mathrm{B} \to \mathrm{A}) = \mathrm{Proofs}(\mathrm{B}) \times \mathrm{Tests}(\mathrm{A}).$$

This intuitive heuristic suggests the *moral identification* of behaviours:

$$\mathrm{Tests}(A) = \mathrm{Proofs}(\neg A)$$

in the sense that with this identifications the last equations are the de Morgan lows.
And let's remark that point 3. says that somewhere there must be the *interaction* "$\pi(\rho)$" between proofs and wannabe proofs, which acts as a *cut* on B.

From all that, we have the following ideas and intuitions:

[54] Supported by corollary 2.41.

i. it has to be possible that a proof for A and a test for A coexist

ii. tests must be of the same kind of proofs

iii. testing a wannabe proof means interact with a proof by means of a sort of cut.

We clearly notice that the natural candidate for tests provided by Tarski semantic, which is Tests(A) = models of $\neg A$, is not of our help since: a proof of A and a model of $\neg A$ cannot exist, they are not object of the same kind, and there is no interaction, as we already remarked. This point of view - that comes out when dealing with the structure of proofs (which becomes handleable in an intuitionistic framework) - escapes to Tarski semantic and requires a finer analysis.

In order for tests to be homogeneous objects with proofs, coexist and interact with them, we shall thus define as "test" a laxer notion than that of "proof" - let's call it "paraproof". And it comes natural to make interact paraproofs between them too.

Let's notice that in realizability the set of stacks $\|.\|$ for an implication is treated exactly as such a notion of test (the other connectives being hidden in the coding). In fact a stack for $A \to B$ is $u.\rho$ where u is a strategy (a wannabe *winning* strategy) for A and ρ a stack for B. And testing a strategy t for $A \to B$ corresponds to executing the process $t \star u.\rho$ in the KAM, a processus passing the test when belonging to the pole.
But at the end of the day realizability is "only" a beautiful machine to extract meaningful computational content from mathematics, and even if close, it is not the "develop this heuristic rigorously" theory.

So we can think to define a space of paraproofs that interact by means of a cut in a "sequent calculus way", and for which the interaction acts as a test that defines the subset of well-behaved objects, which we call proofs.
It turns out that the natural framework to develop some theory out of those ideas is not classical nor intuitionistic logic, nor Krivine's realizability. The natural framework is linear logic.
In fact this ideas have been taken seriously first by J.-Y. Girard and developed by him at the end of the XXth century (independently from classical realizability), as a natural technical and conceptual development of his work in linear logic.
A first point of beginning is given interpreting formulas of linear logic as "games" with some rules, and proofs as winning strategies. This refers to a research area called "game semantics" that is done, essentially, in a category theory setting.
But the point is that we can do better, i.e. go deeper and develop a space where the rules of the games themselves (i.e. the meaning of logical rules) are not presupposed but *emerge* from the interaction. And interaction is governed by the linear negation $(.)^{\perp}$, that is now no more a simple "refutation", but acquires a *normative* role, i.e. A is determined by means of $A^{\perp}$ and vice versa, at the same time, $A^{\perp}$ is determined by means of $A^{\perp\perp} = A$, in a sort of dialectic *à la Hegel*.
The two theories introduced by Girard in which this *interactive* and completely new conception of the foundations of logic is developed, are called "Ludics" ([Gir01]) and "Geometry of Interaction" (developed in a series of papers, starting from [Gir89] and going on). Ludics - which (astonishingly?) found applications in linguistic - is more close to the heuristic we presented here and stresses the idea of "location", while GoI is a general reformulation of cut-elimination[55] in terms of functional analysis or in terms of deformations of paths in some graphs - and has

[55] That we at this point understand to be synonyms of interaction, communication, use, execution, computation, ...

then been found being also linked with the theory of "optimality" for λ-calculus reduction. Those theories provide the main technical ground for what Girard has called the "transcendental syntax program", a program aiming for a finer reformulation of logic (and thus of mathematics) with an *à la Kant* perspective "implicit/explicit" and "typed/untyped".

To conclude and sum up from all we have seen, we understand that proofs, and in particular their interaction, is the real central object of logic, form both a mathematical and a philosophical point of view; and not truth, which is clearly useful in mathematics, but not strong enough to provide a solid foundational ground. Indeed, the idea itself of a "foundation" changes, from the traditional issues (generally coming from the tradition of "analytical philosophy") to being instead the shift from - see the index of this thesis - the level "-1" of truth, to the level "-2" of programs/morphisms, to the level "-3" of interaction with rules presupposed, and finally to level "-4" of "no prejudices" (using Girard's terminology). This gives a deep analysis of the mathematics of Logic.

But in the descent into the other levels, the technical and conceptual matter change and the stress isn't really more on the *computational content of proofs* - see the title of this thesis. We shall thus stop and end this manuscript here.

Bibliography

[AbramTze10] S. Abramsky, N. Tzevelekos,
"Introduction to Categories and Categorical Logic". In: Coecke B. (eds) New Structures for Physics. Lecture Notes in Physics, vol 813. Springer, Berlin, Heidelberg, 2010

[AbrTdf12] V. M. Abrusci, L. Tortora de Falco,
"Logica, vol 1 - Dimostrazioni e modelli al primo ordine". Springer, 2012

[AbrTdf18] V. M. Abrusci, L. Tortora de Falco,
"Logica, vol 2 - Incompletezza, teoria assiomatica degli insiemi". Springer, 2018

[Bar84] H. P. Barendregt,
"The lambda calculus: its syntax and semantics". Studies in logic and the foundations of mathematics. North-Holland, 1984

[Deh93] P. Dehornoy,
"Complexité et décidabilité". Collection Mathématiques et applications. Birkhauser, 2000

[Gir72] J.-Y. Girard,
"Interprétation fonctionnelle et élimination des coupures de l'arithmetique d'ordre supérieur", PhD thesis in mathematics at Université Paris VII, 1972.

[Gir87] J.-Y. Girard,
"Linear logic". Theoretical Computer Science-50, 1987

[Gir89] J.-Y. Girard,
"Geometry of interaction I: interpretation of system F", Ferro, Bonotto, Valentini, and Zanardo editors, Logic colloquium, 1989

[Gir01] J.-Y. Girard,
"Locus solum: from the rules of logic to the logic of rules", Mathematical Structures in Computer Science-11, 2001.

[GirLafTay89] J.-Y. Girard, Y. Lafont, P. Taylor,
"Proofs and Types". Cambridge tracts in theoretical computer science. Cambridge University Press, 1989

[Griff90] T. G. Griffin,
"A Formulae-as-Types Notion of Control". Proceedings of the Seventeenth ACM/SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 1990

[Kriv93] J.-L. Krivine,
"Lambda-calculus, types and models". Ellis Horwood series in computers and their applications. Ellis Horwood, 1993

[Kriv98] J.-L. Krivine,
"Théorie des ensembles". Cassini, Nouvelle bibliothèque mathématique, 1998

[Kriv01] J.-L. Krivine,
"Typed lambda-calculus in classical Zermelo-Fraenkel set theory". Archive for Mathematical Logic, 2001

[Kriv04] J.-L. Krivine,
"Realizability in classical logic". Lessons for PhDs at Aix-Marseille Université, 2004

[Kriv08] J.-L. Krivine,
"Structures de réalisabilité, RAM et ultrafiltre sur N". Septembre 2008

[Miq12] A. Miquel,
"Forcing as a program transformation" (long version, 56 p.). Mathematical Structures in Computer Science, 2012

[Rieg14] L. Rieg,
"On Forcing and Classical Realizability". PhD thesis in Computer Science at Ecole Normale Supérieure of Lyon, 2014

[TdF06] L. Tortora de Falco,
"Sulla struttura logica del calcolo". Rendiconti di Matematica e delle sue applicazioni, Serie VII, 2006.