The Curry-Howard view of classical logic

a short introduction

Stéphane Graham-Lengrand

Version: 29th March 2016

Table of Contents

Notations and prerequisites

In these notes, we assume the reader to be already familiar with some areas and concepts of logic and computer science. Unless specifically given, the notations and definitions used in these notes are rather standard, and formally follow [Len06]. The areas and concepts are:

- Set theory; see e.g. [Kri71].
  In particular we will use the concepts of, and notations for, subsets, power sets, union, intersection and difference of sets, relations, functions, injectivity, surjectivity, etc. Our notation for the power set of A is $\mathbb{P}(A)$. Our notation for the set of total functions from A to B is denoted $A \to B$; the set of partial functions from A to B is denoted $A \rightharpoonup B$. We also assume the reader to be familiar with natural numbers, lists and trees.
- The standard difference between object-level and meta-level.
  In particular, variables of the meta-level are called meta-variables and (unless otherwise stated) "rules" and "systems" are meta-level devices (i.e. a rule has no existence at the object level, but its instances do -and the collection of them, for example).
- Trees and derivations.
  We use inference rules and systems to define sets of (valid) derivations and derivability of judgements, as well as partial derivations; when we state that a rule is derivable/admissible/invertible (in a system) we actually mean that its instances are derivable/admissible/invertible (in the collection of derivations defined by the system).
- Rewriting (first-order and higher-order); see e.g. [Ter03].
  In particular, the notations $\longrightarrow^n$, $\longrightarrow^+$, $\longrightarrow^*$, $\longleftrightarrow^*$, denote the composition n times of a (binary) relation $\longrightarrow$, the transitive closure, the transitive and reflexive closure, and the transitive, reflexive and symmetric closure, of the relation $\longrightarrow$, respectively.

  We assume that the reader is familiar with the properties of confluence and Church-Rosser, weak normalisation, strong normalisation, and the usual techniques to prove them, in particular the simulation techniques.

  Following [Len06], the notation

  $$(\gamma) \quad M \longrightarrow N$$

  introduces a rewrite rule whose contextual closure (or more precisely, the contextual closure of its instances) is denoted $M \longrightarrow_\gamma N$. We also use this notation when γ is a system of rules.

  Our languages will often be made of terms whose syntax is defined by a BNF-grammar. Some of its syntactic categories may contain *variables*. We assume the reader is familiar with variable binding, α-conversion and *equivariance*; specifying binders and their scopes automatically defines what the *free variables* of a term, denoted $\mathsf{FV}(t)$, are; *capture-avoiding substitution* of u for x in t is denoted

$$\{^{u}/_{x}\}\, t$$

where x is a variable of some syntactic category with variables and u is a term of that syntactic category.

- Basic proof theory; see e.g. [TS00]. In particular, standard proof formalisms such as Natural Deduction and Sequent Calculus, for intuitionistic and classical logic (propositional and first-order).

Chapter 1

Classical proofs as programs

Contents

The Curry-Howard correspondence [CF58, How80] has been one of the most fruitful connections between proofs and computation: As one of the embodiments of constructivism, where mathematical proofs bear computational content, the correspondence naturally emerged in the context of minimal and intuitionistic logic, and gave rise to the field of Type Theory [ML82, ML84].

Despite the non-constructive character of proofs in classical logic, arising from the Law of Excluded Middle, or the Double Negation Elimination etc, it is natural to investigate what part of the Curry-Howard correspondence can still be built for that logic.

In this chapter we review the foundations of the correspondence in the framework of classical logic, along the main lines of investigation that were explored over the past 25 years since Griffin's seminal work [Gri90].

The first step in this programme is to turn a proof format for classical logic into a typing system for a language. For such a language to be of computational nature, an operational semantics and/or a denotational semantics has to be designed.

Section 1.1 reviews the basic concepts of the Curry-Howard correspondence, both in their original framework and at a more abstract level. Section 1.2 present some concepts in programming, namely continuations and control, which will prove useful to understand classical proofs as programs. Section 1.3 presents early formalisations of the above concepts as proof-term calculi for classical logic while Section 1.4 presents in more details one of the most convenient ones, which relates to Gentzen's classical sequent calculus. Section 1.6 uses continuations to describe the evaluation strategies known as Call-by-Name and Call-by-Value while Section 1.7 explains how this can be used to build semantics for classical proofs.

1.1 Curry-Howard correspondence: concepts and instances

The correspondence relates logic to programming languages, and is sometimes taken to involve a third aspect, namely category theory (as it forms a popular framework to build the semantics of programming languages). Table 1.1 gives a high-level view of the correspondence,[1] which operates at several levels: mathematical formulae, or propositions, correspond to the types of a given programming language; proofs of such propositions correspond to programs that can be given the corresponding type; the way proofs can be composed corresponds to the way programs can be composed/applied; finally (and this is where we adopt the view of computation as proof-normalisation), cut-elimination corresponds to program execution.

Logic	**Programming language**	**Categories**
Propositions	Types	Objects
Proofs	Typed programs	Morphisms
Cut/Composition	Program composition	Morphism composition
Cut-elimination	Program execution	Equality of morphisms (commuting diagrams)

Table 1.1: High-level view of the Curry-Howard correspondence

The rest of this section gives a brief overview of the correspondence in the framework of minimal and intuitionistic logic. An in-depth presentation of the correspondence can be found in the book [SU06].

1.1.1 Simply-typed combinators

The original instance of the correspondence was given in the study of combinators [CF58], which yields a simple language made of three basic programs **I**, **K**, **S** and with program application as its only construct:

DEFINITION 1 (The (I, K, S)-combinatoric system)

The syntax is given by the following grammar:

$$M, N, \ldots ::= \mathbf{I} \,|\, \mathbf{K} \,|\, \mathbf{S} \,|\, M\ N$$

The last construct, *program application*, is associative to the left, i.e. $(M\ N)\ P$ can be abbreviated as $M\ N\ P$.

[1] We use the expression (Curry-Howard) "correspondence" rather than the popular (Curry-Howard) "isomorphism", as it is difficult to specify what the isomorphism exactly is before specifying exactly what formal systems we intend to relate.

and its operational semantics is given by the following first-order rewrite system

$$\begin{array}{lcl} \mathbf{I}\ M & \longrightarrow & M \\ \mathbf{K}\ M\ N & \longrightarrow & M \\ \mathbf{S}\ M\ N\ P & \longrightarrow & M\ P\ (N\ P) \end{array}$$

※

Clearly, the operational semantics defines **I** as the identity, while **K** provides erasure and **S** provides duplication. The reduction relation is confluent and defines a model of computation that turns out to be Turing-complete. At the cost of losing that property, the language can be given an intuitive typing system, using *simple types*:

Definition 2 (Simple types) *Simple types* are defined by the following grammar:

$$A, B, \ldots ::= a \mid A \to B$$

where a ranges over a fixed set of elements called *atomic types*. The symbol $\to$ is associative to the right, i.e. $A \to (B \to C)$ can be abbreviated as $A \to B \to C$. ※

The typing system is defined as follows:

Definition 3 (Simple types for the (I, K, S)-combinatoric system)

Typing is a binary relation between terms and simple types, denoted with expressions such as $\vdash M : A$. That relation is defined for the combinators as follows:

$$\begin{array}{l} \vdash \mathbf{I} : A \to A \\ \vdash \mathbf{K} : A \to B \to A \\ \vdash \mathbf{S} : (A \to (B \to C)) \to (A \to B) \to (A \to C) \end{array}$$

and program application is typed by the following rule:

$$\frac{\vdash M : A \to B \qquad \vdash N : A}{\vdash M\ N : B}$$

Derivability of the typing statement $\vdash M : A$, from the above axioms and using the program application rule, is denoted $\vdash_{\mathsf{stC}} M : A$. ※

The reduction defined by the rewrite system preserves types, a property called Subject Reduction:

Theorem 1 (Subject reduction for simply-typed combinatoric system)

If $\vdash M : A$ and $M \longrightarrow N$ then $\vdash N : A$. ※

Proof: By induction on the derivation of $M \longrightarrow N$, with the base cases corresponding to the 3 rewrite rules themselves. □

The essence of the Curry-Howard correspondence, is the simple remark that, viewing the functional type construct $\to$ as the logical symbol for implication, simples types are isomorphic[2] to the syntax of formulae for propositional minimal logic [Joh36] and that typing derivations are isomorphic to proofs in a particular Frege-Hilbert system [Fre79, Hil28] for minimal logic.

[2]The isomorphism with simple types assumes that atomic types are isomorphic to atomic formulae.

DEFINITION 4 (Propositional minimal logic)

Formulae of minimal logic are defined by the following grammar:

$$A, B, \ldots ::= \; a \mid A \Rightarrow B$$

where a ranges over a fixed set of elements called *atomic formulae*.

Proofs are the derivations built with the *Modus Ponens* rule

$$\frac{\vdash A \Rightarrow B \qquad \vdash A}{\vdash B}$$

from the axioms:

$$A \Rightarrow A$$
$$A \Rightarrow B \Rightarrow A$$
$$(A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)$$

The symbol $\Rightarrow$ is associative to the right.

Derivability of $\vdash A$, from the above axioms and using the program application rule, is denoted $\vdash_{\mathsf{FH}\Rightarrow} A$. ※

Via the correspondence, the Subject Reduction property allows the view of the reduction relation as a proof-transforming procedure.

1.1.2 Simply-typed λ-calculus

Curry's view about minimal logic was extended by Howard to intuitionistic first-order arithmetic [How80]. A different format was also proposed, both for proofs and for programs, and became perhaps the most popular setting for the Curry-Howard correspondence: Natural Deduction [Gen35] was the formalism used for proofs, and the λ-calculus [Chu41] was the formalism used for programs. This instance of the Curry-Howard correspondence that we present is a version of natural deduction using *sequents* and a version of the simply-typed λ-calculus using typing contexts.

DEFINITION 5 (λ-calculus) The syntax of the λ-calculus is given by the following grammar:

$$M, N, \ldots ::= \; x \mid \lambda x.M \mid M \; N$$

where x ranges over a denumerable set of *variables*, and the construct $\lambda x.M$ binds x in M.[3]

Standard conventions are used for parentheses [Bar84]: the scopes of binders extend as much as parentheses allow (i.e. $\lambda x.M \; N$ abbreviates $\lambda x.(M \; N)$); program application is associative to the left (i.e. $M \; N \; P$ abbreviates $(M \; N) \; P$); moreover, binder can be grouped, so that $\lambda xy.M$ abbreviates $\lambda x.\lambda y.M$.

The following rewrite rules

$$\begin{array}{lll} (\beta) & (\lambda x.M) \; N & \longrightarrow \; \left\{ {}^{N}\!/_{x} \right\} M \\ (\eta) & \lambda x.M \; x & \longrightarrow \; M \qquad \text{if } x \notin \mathsf{FV}(M) \end{array}$$

define the reduction relations $\longrightarrow_\beta$, $\longrightarrow_\eta$ and $\longrightarrow_{\beta\eta}$. ※

As for the combinatoric system from Section 1.1.1, the reduction relations are confluent:

THEOREM 2 (Confluence) $\longrightarrow_\beta$, $\longrightarrow_\eta$ and $\longrightarrow_{\beta\eta}$ are confluent. ※

[3] As mentioned in the section about notations, specifying binders and their scopes automatically defines free variables, α-conversion, capture-avoiding substitution, etc.

Proof: See for instance [Bar84]. □

DEFINITION 6 (Simply-typed λ-calculus) *Typing contexts* are finite maps from variables to simple types, with () denoting the empty context (sometimes the notation () is completely omitted), Γ, Γ' denoting the union of contexts Γ and Γ' (assuming it is defined), and $x:A$ denoting the singleton context mapping variable x to the simple type A.

The typing rules of the simply-typed λ-calculus are given in Fig. 1.

Derivability of the typing statement $\Gamma \vdash M:A$ in that system is denoted $\Gamma \vdash_{\mathsf{st}\lambda} M:A$. ※

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

Figure 1: Simply-typed λ-calculus

As for the combinatoric system from Section 1.1.1, the reduction relations satisfy Subject Reduction:

THEOREM 3 (Subject reduction for simply-typed λ-calculus)

1. If $\Gamma, x:A \vdash M:B$ and $\Gamma \vdash N:A$ then $\Gamma \vdash \left\{ {}^{N}/_{x} \right\} M:B$.
2. If $\Gamma \vdash M:A$ and $M \longrightarrow_{\beta\eta} N$ then $\Gamma \vdash N:A$.

※

Proof: See for instance [Bar84]. □

Our second instance of the Curry-Howard correspondence relates the simply-typed λ-calculus with the Natural Deduction system $\mathsf{NJ}_{\Rightarrow}$ for minimal logic.

DEFINITION 7 (Natural Deduction for minimal logic - $\mathsf{NJ}_{\Rightarrow}$)

System $\mathsf{NJ}_{\Rightarrow}$ is the inference system given in Fig. 2, where

- A, B range over formulae of minimal logic;
- Γ stands for a "collection" of formulae. By collection we mean either set or multiset,[4] with Γ, Γ' denoting the union of Γ and Γ', A denoting either the formula A itself or the singleton $\{A\}$ (or $\{\!\{A\}\!\}$), while the empty set (or multiset) is sometimes ommitted;
- $\Gamma \vdash A$ is a structure called *sequent*.

Derivations in that system are called *proofs* in $\mathsf{NJ}_{\Rightarrow}$.

Derivability in $\mathsf{NJ}_{\Rightarrow}$ of a sequent $\Gamma \vdash A$ is denoted $\Gamma \vdash_{\mathsf{NJ}_{\Rightarrow}} A$. ※

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

Figure 2: Natural Deduction for minimal logic - $\mathsf{NJ}_{\Rightarrow}$

[4] That choice will change the number of proofs of a given formula.

Comparing Fig. 1 and Fig. 2 reveals our second instance of the Curry-Howard correspondence, although the exact meaning of the word 'correspondence' is in this case more subtle than for our first instance:

Clearly, the bijective aspect that pertains to the word "isomorphism" is jeopardised as $\dfrac{}{x:A, y:A \vdash x:A}$ and $\dfrac{}{x:A, y:A \vdash y:A}$ are clearly two distinct typing derivations which would both 'correspond to' the proof $\dfrac{}{A, A \vdash A}$ (whether we use sets or multisets). Moreover, binding introduces an ambiguity in the way we count typing derivations: is there one or infinitely many derivations of $\vdash \lambda x.x : A \to A$?[5]

For this reason, these notes take the view that the interesting aspects of the Curry-Howard correspondence do not include the bijective aspect of an encoding from one system into another, but rather its compositionality (for trees), and the soundness and completeness properties:

In the present case, the forgetful encoding that maps every typing derivation to a proof is compositional with respect to the tree-structure of derivations; its surjectivity provides completeness of type inhabitation -whether there exists a λ-term of a given type- with respect to the provability of the corresponding formula; soundness is simply the fact that the tree obtained by forgetting variables and terms from a typing derivation is a correct proof.

These are the properties that we will aim at when investigating the variants of the Curry-Howard correspondence.

As for the combinatoric system from Section 1.1.1, the Subject Reduction property allows the view of the reduction relations $\longrightarrow_\beta$, $\longrightarrow_\eta$ and $\longrightarrow_{\beta\eta}$ as proof-transforming procedures.

In summary, the most well-known settings for the Curry-Howard correspondence are:

Frege-Hilbert system	$\leftrightarrow$	Combinators (S,K,I)	[CF58]
Natural Deduction	$\leftrightarrow$	Typed λ-terms	[How80]

1.1.3 The categorical aspect

We now briefly mention what is sometimes considered a third aspect of the Curry-Howard corespondence, in category theory.

Categories can be used to shed a semantical light on the Curry-Howard correspondence. In our case, a particular kind of category provides models of the simply-typed λ-calculus: cartesian closed categories (CCC). In brief, CCC feature a terminal object, products, and exponential objects. We start with a few notational conventions:

NOTATION 8 (Category)

The class of morphisms from object A to object B is denoted $\mathsf{hom}(A, B)$, and the expression $f\colon A \longrightarrow B$ denotes that f is a morphism from A to B. Identity morphisms are denoted Id_A, and the composition of $f\colon A \longrightarrow B$ and $g\colon B \longrightarrow C$ is denoted $f \cdot g\colon A \longrightarrow C$.

[5] Formally, the typing system allows infinitely many premisses for that typing judgement, depending on the variable that we pick to place in the typing context with type A.

In a cartesian closed category (CCC),

- the *terminal object* is denoted 1, with morphisms $1_A : A \longrightarrow 1$
- the *product* of A and B is denoted $A \times B$, with projections denoted $\pi_1 : A \times B \longrightarrow A$ and $\pi_2 : A \times B \longrightarrow B$ (and more generally, the i^{th} projection from n objects is denoted $\pi_{i/n}$) and morphism pairing denoted $\langle f_1, f_2 \rangle : C \longrightarrow A \times B$ for every $f_1 : C \longrightarrow A$ and $f_2 : C \longrightarrow B$.
- the *exponential* of A and B is denoted B^A, with morphisms $\mathsf{eval} : B^A \times A \longrightarrow B$ and a currified morphism $\Lambda g : X \longrightarrow B^A$ for every $g : X \times A \longrightarrow B$.

※

For a formal definition of the above concepts, see Appendix 3.2.3.

Definition 9 (Semantics of the simply-typed λ-calculus in a CCC)

Consider a cartesian closed category.

Consider a mapping that interprets every atomic type a as an object $[\![a]\!]$ of the CCC, and extend it to all simple types by defining $[\![A{\to}B]\!]$ as $[\![B]\!]^{[\![A]\!]}$.

Consider a total order on the λ-calculus variables; when writing a typing context as $x_1 : A_1, \ldots, x_n : A_n$ we now follow the convention that $x_1, \ldots, x_n$ is an increasing sequence; we then define the semantics of any typing context by

$$[\![x_1 : A_1, \ldots, x_n : A_n]\!] := 1 \times [\![A_1]\!] \times \cdots \times [\![A_n]\!]$$

The semantics of a typing derivation π for the typing judgement $\Gamma \vdash M : A$ is defined according to Fig. 3, by induction on π, as a morphism $[\![\pi]\!] : [\![\Gamma]\!] \longrightarrow [\![A]\!]$. ※

$$\left[\!\!\left[\frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i} \right]\!\!\right] := \pi_{i+1/n+1} : 1 \times [\![A_1]\!] \times \cdots \times [\![A_n]\!] \longrightarrow [\![A_i]\!]$$

$$\left[\!\!\left[\frac{\begin{array}{c}\vdots\,\pi \\ \Gamma, x : A \vdash M : B\end{array}}{\Gamma \vdash \lambda x.M : A{\to}B} \right]\!\!\right] := \Lambda g : [\![\Gamma]\!] \longrightarrow [\![B]\!]^{[\![A]\!]}$$

$$\text{where} \quad g = \left[\!\!\left[\begin{array}{c}\vdots\,\pi \\ \Gamma, x : A \vdash M : B\end{array} \right]\!\!\right] : [\![\Gamma]\!] \times [\![A]\!] \longrightarrow [\![B]\!]$$

$$\left[\!\!\left[\frac{\begin{array}{cc}\vdots\,\pi_1 & \vdots\,\pi_2 \\ \Gamma \vdash M : A{\to}B & \Gamma \vdash N : A\end{array}}{\Gamma \vdash M\ N : B} \right]\!\!\right] := \langle g_1, g_2 \rangle \cdot \mathsf{eval} : [\![\Gamma]\!] \longrightarrow [\![B]\!]$$

$$\text{where} \quad g_1 = \left[\!\!\left[\begin{array}{c}\vdots\,\pi_1 \\ \Gamma \vdash M : A{\to}B\end{array} \right]\!\!\right] : [\![\Gamma]\!] \longrightarrow [\![B]\!]^{[\![A]\!]}$$

$$\text{and} \quad g_2 = \left[\!\!\left[\begin{array}{c}\vdots\,\pi_2 \\ \Gamma \vdash N : A\end{array} \right]\!\!\right] : [\![\Gamma]\!] \longrightarrow [\![A]\!]$$

Figure 3: Semantics of the simply-typed λ-calculus in a CCC

Note that in the case of a λ-abstraction, we assume that x is (strictly) greater than any

variable in Γ. Any derivation in which this is not the case can easily be turned into one satisfying this condition: variables can always be renamed[6] so that they are introduced in the typing context in increasing order.[7]

Now as mentioned earlier, we can relate the reductions in the simply-typed λ-calculus to the equality of morphisms in CCC:

THEOREM 4 (Soundness and completeness)

Assume $\Gamma \vdash_{\mathsf{st}\lambda} \overset{\vdots\,\pi}{M : A}$ and $\Gamma \vdash_{\mathsf{st}\lambda} \overset{\vdots\,\pi'}{N : A}$.

$M \longleftrightarrow^*_{\beta\eta} N$ if and only if in every CCC we have $[\![\pi]\!] = [\![\pi']\!]$. ※

The equality theorems that can be derived from the axioms of CCC (and that therefore hold in every CCC) are reflected syntactically in the simply-typed λ-calculus, and the simply-typed λ-calculus is therefore said to form an *internal language* for CCC.

Note that it is easy to define the semantics of the (**I**, **K**, **S**)-combinatoric system (with simple types) in a CCC, for instance by encoding combinators as simply-typed λ-terms:

THEOREM 5 (Semantics of the (I, K, S)-combinatoric system)

The encoding of Fig. 4 satisfies satisfies the following properties:

- If $\vdash_{\mathsf{stC}} M : A$ then $\vdash_{\mathsf{st}\lambda} \overline{M} : A$, with a function $\pi \mapsto \overline{\pi}$ transforming a derivation of the former into a derivation of the latter.
- If $M \longrightarrow M'$ then $\overline{M} \longrightarrow_\beta M'$.

The above properties allow the definition of the semantics $[\![\pi]\!]$ of a typing derivation π of $\vdash_{\mathsf{stC}} M : A$ as the morphism $[\![\overline{\pi}]\!] : 1 \longrightarrow [\![A]\!]$, such that the following holds:
If $\pi \longleftrightarrow^* \pi'$ then $[\![\pi]\!] = [\![\pi']\!]$. ※

$$
\begin{array}{lcl}
\overline{\mathsf{I}} & := & \lambda x.x \\
\overline{\mathsf{K}} & := & \lambda xy.x \\
\overline{\mathsf{S}} & := & \lambda xyz.x\ z\ (y\ z) \\
\overline{M\ N} & := & \overline{M}\ \overline{N}
\end{array}
$$

Figure 4: (**I**, **K**, **S**)-combinators as λ-terms

1.1.4 Applying the methodology to other systems

The approach of the Curry-Howard correspondence can be, and has been, generalised with the following methodology:

- The first step is to decorate proofs with proof-terms: $\Gamma \vdash A$ becomes $\Gamma' \vdash M : A$, with Γ' being a typing context whose co-domain (i.e. the types which have been assigned to variables) is Γ;

[6] Using equivariance of typing derivations.

[7] Alternative presentations of the simply-typed λ-calculus may be more convenient to define its semantics in a CCC: the use of De Bruijn indices (see e.g. [Bar84]) instead of named variables provides a natural way of ordering the objects in the interpretation of a typing environment (without resorting to ordering the set of variables); if variables carry their own type (or when each type comes with its own set of variables), the interpretation can be defined on the terms themselves rather than their typing derivations.

- the second is to express proof transformations in terms of proof-term reduction, denoted $M \longrightarrow_{\mathcal{S}} N$, often given by a rewrite system $\mathcal{S}$.

The desired properties of reduction are

- *Progress*, i.e. any term containing "undesirable structures" can be reduced.
- *Subject reduction* property, i.e. preservation of typing:
  If $\Gamma \vdash M : A$ and $M \longrightarrow_{\mathcal{S}} N$ then $\Gamma \vdash N : A$
- possibly *Confluence*, programs are deterministic.
- possibly *Normalisation*, i.e. the fact that the execution of programs terminates.

The notion of "undesirable structures" is of course one of the concepts to identify in an interesting way; for instance in the simply-typed λ-calculus, a structure of the form $(\lambda x.M)\ N$ corresponds to the introduction of implication followed by the elimination of the introduced implication, a situation which we may consider undesirable from a proof-theoretic point of view.

To illustrate this methodology, we show how the correspondence from Section 1.1.2 can be extended to intuitionistic logic with both the implication connective and the logical constant $\bot$.

First note that by identifying $\bot$ simply as one of the atomic formulae, intuitionistic negation can be defined as follows: $\neg A := A \Rightarrow \bot$. With this definition, the following rules are instances of those of the simply-typed λ-calculus:

$$\frac{\Gamma, x : A \vdash M : \bot}{\Gamma \vdash \lambda x.M : \neg A} \qquad \frac{\Gamma \vdash M : \neg A \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : \bot}$$

and these reflect the usual Natural Deduction rules for negation, but what is missing, to have intuitionistic logic, is the rule named *Ex falso quodlibet* (EFQ):

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}$$

We now see what that rule may become in the Curry-Howard correspondence.

Example 1 (Extension of the Curry-Howard correspondence for $\mathbf{NJ}_{\Rightarrow,\bot}$)

We extend the syntax of the λ-calculus with the following construct:

$$M, N, \ldots ::= \ \ldots \ \big| \ \mathsf{abort}(M)$$

and we add to the simply-typed λ-calculus a typing rule corresponding to EFQ:

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \mathsf{abort}(M) : A}$$

We may then add to the λ-calculus a rule such as

$$(\flat) \quad \mathsf{abort}(M)\ N \longrightarrow \mathsf{abort}(M)$$

to computationally interpret the new construct as a greedy consumer of arguments, and $\longrightarrow_{\flat}$, $\longrightarrow_{\beta\flat}$, $\longrightarrow_{\eta\flat}$, and $\longrightarrow_{\beta\eta\flat}$ are all confluent.

With these definitions, we still have Subject Reduction:
If $\Gamma \vdash M : A$ and $M \longrightarrow_{\beta\eta\flat} N$ then $\Gamma \vdash N : A$.

We can also interpret EFQ in category theory by requiring from a CCC the extra axiom that there is an *initial object* $\bot$, i.e. an object such that, from every object A there is a unique morphism $0_A : \bot \longrightarrow A$ (which is the dual of the terminal object 1 of the CCC). ※

From there, it is natural to try to extend the above correspondence to classical logic, which can be obtained from intuitionistic logic by adding any one of the three axiom schemes:

Elimination of double negation (EDN):	$(\neg\neg A)\Rightarrow A$
Peirce's law (PL):	$((A\Rightarrow B)\Rightarrow A)\Rightarrow A$
Law of excluded middle (LEM):	$A \vee \neg A$

In presence of EFQ (in short, the axiom scheme $\bot\Rightarrow A$), the above schemes are all equivalent in terms of formula provability. Interestingly enough and as noted in [AH03], without EFQ, we only have the following implications between the schemes:

$$\text{EDN}\Rightarrow\text{PL}\Rightarrow\text{LEM}$$
$$\text{EDN}\Rightarrow\text{EFQ}$$

Alternatives to adding axiom schemes is to add inference rules such as

$$\frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A} \text{ for EDN}$$

or even to change the structure of the proof formalism, for instance by using the classical sequent calculus [Gen35] with *right-contraction*.

When thinking about classical logic, we have a tendency to identify a formula A with $\neg\neg A$, as suggested not only by the elimination of double negation but also by models of classical provability in boolean algebras.

Now, attempts to apply the Curry-Howard methodology to, say, the above axiom schemes or inference rule, are limited by the following fact:

A CCC with initial object $\bot$ and such that every object A is naturally isomorphic to $\bot^{\bot^A}$, collapses to a boolean algebra: there is at most 1 morphism between any 2 objects (see the proof in [LS86] or [Str11]).

That means that such a category would not distinguish two proofs of the same theorem, which is rather useless for a theory of proofs, or for the proofs-as-programs paradigm.

At that point, the natural question to ask is whether classical logic has computational content? To that question, and based on the above remarks, the book *Proofs and Types* [GTL89] answers in 1989:

"[The Curry-Howard] interpretation is not possible with classical logic: there is no sensible way of considering proofs as algorithms. In fact, classical logic has no denotational semantics, except the trivial one which identifies all the proofs of the same type."

In the rest of this chapter we explore the alternative answers that have been given to the question since then.

1.2 Continuations and control

For this we start with some concepts which at first sight may seem unrelated: continuations and control.

A freeze-frame shot taken at one point of a program's execution flow could be represented, in a high-level view, as follows:

$\downarrow$	code P that has been executed, producing data v
v	its output
$\downarrow$	code E that remains to be executed, consuming data v

The code E that remains to be executed, and more generally the programming environment or programming context within which some code is executed, is called *continuation*.

The concept is also useful for compiling recursive calls: consider the following pseudo code

```
myfunction(a1,...,an){
  some code;
  x = myfunction(a1',...,an');
  some code possibly using x;
}
```

When executing the recursive call, the code that remains to be executed (i.e. `some code possibly using x`), together with the values of the local variables, needs to be stored in order to resume computation after the recursive call has returned with a value for `x`. But this is not needed in the case of *tail recursion*, in which `some code possibly using x` just returns (the value for) `x`.

The above code can be transformed into a tail-recursive code by modelling the remaining code `some code possibly using x` as a "continuation" function `c'` taking the value of `x` as input, and by *passing that continuation* `c'` as an extra argument to the recursive call, now in charge of executing it once the value of `x` is determined:

```
myfunction(a1,...,an,c){
  some code;
  return myfunction(a1',...,an',c');
}
```

The function `myfunction` now generally takes an extra argument `c` (the continuation function describing what to do with the result) and uses it in e.g. `some code` or in the contruction of `c'`. Originally, the continuation is often the identity function ("do nothing with your result, just return it"), but it usually becomes more and more sophisticated with each recursive call.

Now we see what these concepts become in the case where the programming language is the λ-calculus. An instance of the above program execution flow picture

$\downarrow$ code P that has been executed, producing data v
v its output
$\downarrow$ code E that remains to be executed, consuming data v

can be seen by considering

- P to be a λ-term that is reduced,
- v to be the value to which P reduces,
- E to be the context, in the syntactic sense: a term with a hole $E[\ ]$ (with the original λ-term being $E[P]$, i.e. the context $E[\ ]$ whose hole has been filled with the λ-term P).

This is only a general idea: whether that view accurately describes program execution depends on the evaluation strategy for λ-terms; in particular, whether λ-terms are reduced inside-out, what notion of "value" is considered (is it a normal form?), what grammar for contexts $E[\ ]$ ranges over, etc.

But in pure λ-calculus, it is clear that P has no knowledge of $E[\ ]$ while being evaluated.

Control is about letting a program know and manipulate its evaluation context. Originally, the concept was used to model `goto` instructions, and other features that are not pure functional programming.

In the case of λ-calculus, the evaluation context $E[\ ]$ within which a term is evaluated gives rise to a continuation function $\lambda x.E[x]$ (for a fresh variable x) that could be passed as

an argument.

Reynolds [Rey72], Strachey-Wadsworth [SW00] (re-edition of 74) explored continuations and control along those lines, letting a program capture its evaluation context with a feature known as *call-with-current-continuation* (call-cc): cc. This was added to the programming language `Scheme`.

Felleisen's PhD work [Fel87] on the Syntactic Theory of Control introduced another control operator: $\mathcal{C}$.

The general idea of these control operators is given by the following reduction rules:

$$\begin{array}{lll} E[\mathsf{cc}\ M] & \longrightarrow & E[M\ (\lambda x.E[x])] \\ E[\mathcal{C}\ M] & \longrightarrow & M\ (\lambda x.E[x]) \end{array}$$

In presence of $\mathsf{abort}(\ )$ and its (slightly modified) rule

$$E[\mathsf{abort}(M)] \longrightarrow M$$

cc and $\mathcal{C}$ are interdefinable:

$$\begin{array}{lll} \mathcal{C}\ M & := \mathsf{cc}\ (\lambda k.\mathsf{abort}(M\ k)) & k \notin \mathsf{FV}(M) \\ \mathsf{cc}\ M & := \mathcal{C}\ (\lambda k.k\ (M\ k)) & k \notin \mathsf{FV}(M) \end{array}$$

Indeed,

$$\begin{array}{lcl} E[\mathsf{cc}\ M] & = & E[\mathcal{C}\ (\lambda k.k\ (M\ k))] \\ & \longrightarrow & (\lambda k.k\ (M\ k))\ (\lambda x.E[x]) \\ & \longrightarrow & (\lambda x.E[x])\ (M\ \lambda x.E[x]) \\ & \longrightarrow & E[M\ (\lambda x.E[x])] \\ E[\mathcal{C}\ M] & = & E[\mathsf{cc}\ (\lambda k.\mathsf{abort}(M\ k))] \\ & \longrightarrow & E[(\lambda k.\mathsf{abort}(M\ k))\ (\lambda x.E[x])] \\ & \longrightarrow & E[\mathsf{abort}(M\ \lambda x.E[x])] \\ & \longrightarrow & M\ (\lambda x.E[x]) \end{array}$$

Of course, the above rules are not "standard" rewrite rules (clearly not first-order rewrite rules) and remain informal because we have not specified what $E[\ ]$ exactly stands for or ranges over.

More fundamentally, a central question about control is: what kind of continuation can be captured by a control operator and how? Is the capture delimited? undelimited? etc.

But what is interesting is that the above intuitions are sufficient to start seeing a connection with classical logic, as initiated by Griffin in [Gri90]:

$$\begin{array}{ll} \mathsf{cc} \text{ can be typed by PL:} & ((A{\to}B){\to}A){\to}A \\ \mathcal{C} \text{ can be typed by EDN:} & (\neg\neg A){\to}A \end{array}$$

With these types, the rewrite rules satisfy the following Subject Reduction properties: The following (again, informal) typing tree for $E[\mathsf{cc}\ M]$

$$\dfrac{\dfrac{}{\Gamma \vdash \lambda x.E[x] : A{\to}B} \qquad \dfrac{\dfrac{}{\Gamma \vdash \mathsf{cc} : ((A{\to}B){\to}A){\to}A} \qquad \Gamma \vdash M : (A{\to}B){\to}A}{\Gamma \vdash \mathsf{cc}\ M : A}}{\Gamma \vdash E[\mathsf{cc}\ M] : B}$$

can be transformed into a typing tree for $E[M\ (\lambda x.E[x])]$

$$\dfrac{\dfrac{}{\Gamma \vdash \lambda x.E[x] : A \to B} \qquad \dfrac{\Gamma \vdash M : (A \to B) \to A \qquad \dfrac{}{\Gamma \vdash \lambda x.E[x] : A \to B}}{\Gamma \vdash M\ (\lambda x.E[x]) : A}}{\Gamma \vdash E[M\ (\lambda x.E[x])] : B}$$

while the following (again, informal) typing tree for $E[\mathcal{C}\ M]$

$$\dfrac{\dfrac{}{\Gamma \vdash \lambda x.E[x] : A \to \bot} \qquad \dfrac{\dfrac{}{\Gamma \vdash \mathcal{C} : ((A \to \bot) \to \bot) \to A} \qquad \Gamma \vdash M : (A \to \bot) \to \bot}{\Gamma \vdash \mathcal{C}\ M : A}}{\Gamma \vdash E[\mathcal{C}\ M] : \bot}$$

can be transformed into a typing tree for $E[M\ (\lambda x.E[x])]$

$$\dfrac{\dfrac{}{\Gamma \vdash M : (A \to \bot) \to \bot} \qquad \dfrac{}{\Gamma \vdash \lambda x.E[x] : A \to \bot}}{\Gamma \vdash M\ (\lambda x.E[x]) : \bot}$$

We can already see that, for the Subject Reduction property to hold in the case of $\mathcal{C}$, the context $E[\ ]$ cannot be any context: it has to produce something of type $\bot$. Similarly, for the generalised abort rule to satisfy Subject Reduction, the context $E[\ ]$ also needs to produce something of type $\bot$.

In fact, $\mathcal{C}$ generalises $\mathsf{abort}(\_)$, since we can define

$$\mathsf{abort}(M) := \mathcal{C}\ (\lambda x.M)$$

where x is a fresh (and therefore dummy) variable.

This reflects what we have already seen in pure logic: EDN $\Leftrightarrow$ (PL $\wedge$ EFQ)

1.3 Contributions in the 90s

One possible formalisation of the above informal concepts was proposed by Parigot [Par92] in the form of the $\lambda\mu$-calculus.

Definition 10 ($\lambda\mu$-calculus) The syntax of terms extends that of λ-calculus as follows:

$$\begin{array}{lrcl} \text{Terms} & M, N, P \ldots & ::= & x \mid \lambda x.M \mid M\ N \mid \mu\alpha.c \\ \text{Commands} & c & ::= & [\alpha]M \end{array}$$

where α ranges over a new set of variables called *continuation variables*, and $\mu\alpha.c$ binds α in c. The scope of this binder, as well as that of the unique *command* construct $[\alpha]M$, extend as much as parentheses allow, so that $\mu\alpha.M\ N$ stands for $\mu\alpha.(M\ N)$ and $[\alpha]M\ N$ stands for $[\alpha](M\ N)$.

The typing rules extend those of λ-calculus as follows:

$$\frac{}{\Gamma, x:A \vdash x:A; \Delta}$$

$$\frac{\Gamma, x:A \vdash M:B; \Delta}{\Gamma \vdash \lambda x.M : A{\to}B; \Delta} \qquad \frac{\Gamma \vdash M:A{\to}B; \Delta \quad \Gamma \vdash N:A; \Delta}{\Gamma \vdash M\ N : B; \Delta}$$

$$\frac{c:(\Gamma \vdash ; \alpha:A, \Delta)}{\Gamma \vdash \mu\alpha.c : A; \Delta} \qquad \frac{\Gamma \vdash M:A; \alpha:A, \Delta}{[\alpha]M : (\Gamma \vdash ; \alpha:A, \Delta)}$$

where Γ is a typing context for term-variables and Δ is a typing context for continuation-variables. Derivability of sequents in this system is respectively denoted $\Gamma \vdash_{\lambda\mu} M:A\ ;\ \Delta$ and $c:(\Gamma \vdash_{\lambda\mu} ;\Delta)$.

The reduction rules extend the β-reduction of λ-calculus as follows:

$$\begin{array}{lcl} (\lambda x.M)\ N & \longrightarrow & \left\{{}^{N}\!/_{x}\right\} M \\ (\mu\alpha.c)\ N & \longrightarrow & \mu\beta.\left\{{}^{[\beta]M\ N}\!/_{[\alpha]M}\right\} c \\ [\beta]\mu\alpha.c & \longrightarrow & \left\{{}^{\beta}\!/_{\alpha}\right\} c \end{array}$$

where $\left\{{}^{[\beta]M\ N}\!/_{[\alpha]M}\right\} c$ is an unconventional substitution operation, consisting in replacing, in c, every subcommand (i.e. subterm that is a command) of the form $[\alpha]M$ by $[\beta]M\ N$, with the usual capture-avoiding conditions pertaining to substitution.

The rules define a reduction relation $\longrightarrow_{\lambda\mu}$ on both terms and commands. ※

A basic intuition of the syntax is that each continuation variable α represents a "place" where various sub-terms of a given type (that of α) can be "stored" with a construct such as $[\alpha]M$. The construct $\mu\alpha.c$ retrieves what is stored under the continuation variable α and presents it as if it was a simple term. The second rewrite rule distributes for instance an argument to every sub-term stored under the variable α.

This calculus provides a computational interpretation of classical logic. Indeed, the typing system, when forgetting variables and terms, turns into the proof system of Fig. 5, where Γ and Δ now stand for sets or multisets of formulae. We see that the system generalises $\mathsf{NJ}_{\Rightarrow}$, in particular with a more general form of sequent: $A_1, \ldots, A_n \vdash A; B_1, \ldots, B_m$, and a new form of sequent $A_1, \ldots, A_n \vdash ; B_1, \ldots, B_m$.

$$\frac{}{\Gamma, A \vdash A; \Delta}$$

$$\frac{\Gamma, A \vdash B; \Delta}{\Gamma \vdash A{\Rightarrow}B; \Delta} \qquad \frac{\Gamma \vdash A{\Rightarrow}B; \Delta \quad \Gamma \vdash A; \Delta}{\Gamma \vdash B; \Delta}$$

$$\frac{\Gamma \vdash ; A, \Delta}{\Gamma \vdash A; \Delta} \qquad \frac{\Gamma \vdash A; A, \Delta}{\Gamma \vdash ; A, \Delta}$$

Figure 5: The proof system corresponding to the simply-typed $\lambda\mu$-calculus

Just like a sequent $A_1, \ldots, A_n \vdash A$ of $\mathsf{NJ}_\Rightarrow$ can be interpreted as the formula $(A_1 \wedge \cdots \wedge A_n) \Rightarrow A$, the two sequent forms above can respectively be interpreted as the formulae $(A_1 \wedge \cdots \wedge A_n) \Rightarrow (A \vee B_1 \vee \cdots \vee B_m)$ and $(A_1 \wedge \cdots \wedge A_n) \Rightarrow (B_1 \vee \cdots \vee B_m)$. With this interpretation, the system of Fig. 5 can be easily checked to be sound with respect to classical logic, and for completeness we can see that

- the rules of $\mathsf{NJ}_\Rightarrow$ are particular instances of the first three rules;
- the system features a right-contraction rule, which allows Peirce's Law to be proved, as we see below.

As with the simply-typed λ-calculus, the rewrite rules satisfy Subject Reduction, which allows the $\lambda\mu$-calculus to describe a proof-transforming procedure for the system of Fig. 5:

Theorem 6 (Subject reduction for the simply-typed $\lambda\mu$-calculus)

1. If $\Gamma \vdash_{\lambda\mu} M : A; \Delta$ and $M \longrightarrow_{\lambda\mu} M'$ then $\Gamma \vdash_{\lambda\mu} M' : A; \Delta$.
2. If $c : (\Gamma \vdash_{\lambda\mu} ; \Delta)$ and $c \longrightarrow_{\lambda\mu} c'$ then $c' : (\Gamma \vdash_{\lambda\mu} ; \Delta)$.

※

Remark 7 This calculus integrates Peirce's law: By defining

$$\mathsf{cc} := \lambda x.\mu\alpha.[\alpha](x\ \lambda y.\mu\beta.[\alpha]y)$$

we can build the following typing tree:

$$\dfrac{\dfrac{}{x:(A\to B)\to A \vdash x:(A\to B)\to A;\alpha:A} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{x:(A\to B)\to A, y:A \vdash y:A;\alpha:A,\beta:B}}{[\alpha]y:(x:(A\to B)\to A, y:A \vdash ;\alpha:A,\beta:B)}}{x:(A\to B)\to A, y:A \vdash \mu\beta.[\alpha]y:B;\alpha:A}}{x:(A\to B)\to A \vdash \lambda y.\mu\beta.[\alpha]y:A\to B;\alpha:A}}{\dfrac{\dfrac{x:(A\to B)\to A \vdash x\ \lambda y.\mu\beta.[\alpha]y:A;\alpha:A}{[\alpha](x\ \lambda y.\mu\beta.[\alpha]y):(x:(A\to B)\to A \vdash ;\alpha:A)}}{\dfrac{x:(A\to B)\to A \vdash \mu\alpha.[\alpha](x\ \lambda y.\mu\beta.[\alpha]y):A;}{\vdash \mathsf{cc}:((A\to B)\to A)\to A;}}}$$

Now, consider that contexts are of the form $E[\ ] = [\gamma]([\ ]\ N_1 \ldots N_n)$. We can perform the following reduction:

$$\begin{array}{rcl} E[\mathsf{cc}\ M] & = & [\gamma](\lambda x.\mu\alpha.[\alpha](x\ \lambda y.\mu\beta.[\alpha]y))\ M\ N_1 \ldots N_n \\ & \longrightarrow & [\gamma](\mu\alpha.[\alpha](M\ \lambda y.\mu\beta.[\alpha]y))\ N_1 \ldots N_n \\ & \longrightarrow & [\gamma](\mu\alpha.[\alpha](M\ \lambda y.\mu\beta.[\alpha]y\ N_1)\ N_1)\ N_2 \ldots N_n \\ & \longrightarrow & \ldots \\ & \longrightarrow & [\gamma]\mu\alpha.[\alpha](M\ \lambda y.\mu\beta.[\alpha]y\ N_1 \ldots N_n)\ N_1 \ldots N_n \\ & \longrightarrow & [\gamma](M\ \lambda y.\mu\beta.[\gamma]y\ N_1 \ldots N_n)\ N_1 \ldots N_n \\ & = & E[M\ (\lambda y.\mu\beta.E[y])] \end{array}$$

※

Notice that what is passed to M as an argument is not exactly $\lambda y.E[y]$, since $E[\ ]$ forms a command and $\lambda y.E[y]$ is not correct syntax, but $\mu\beta.E[y]$ turns the command $E[y]$ into a term (of any type).

Remark 8 If given a top-level continuation variable $\mathsf{top} : \bot$ (Ariola-Herbelin [AH03, AH08]), then the $\lambda\mu$-calculus integrates *Ex falso quodlibet* and the elimination of double negation:

We can build the following typing tree:

$$\dfrac{\dfrac{\dfrac{}{x:\bot \vdash x:\bot;\alpha:A}}{[\mathsf{top}]x:(x:\bot \vdash ;\alpha:A)}}{x:\bot \vdash \mu\alpha.[\mathsf{top}]x:A;}}{\vdash \lambda x.\mu\alpha.[\mathsf{top}]x:\bot\to A;}$$

and perform the following reduction:

$$\begin{array}{rcl} E[(\lambda x.\mu\alpha.[\mathsf{top}]x)\ M] & = & [\gamma](\lambda x.\mu\alpha.[\mathsf{top}]x)\ M\ N_1 \dots N_n \\ & \longrightarrow & [\gamma](\mu\alpha.[\mathsf{top}]M)\ N_1 \dots N_n \\ & \longrightarrow & [\gamma](\mu\alpha.[\mathsf{top}]M)\ N_2 \dots N_n \\ & \longrightarrow & \dots \\ & \longrightarrow & [\gamma](\mu\alpha.[\mathsf{top}]M) \\ & \longrightarrow & [\mathsf{top}]M \end{array}$$

And by defining

$$\mathcal{C} := \ \lambda x.\mu\alpha.[\mathsf{top}](x\ \lambda y.\mu\beta.[\alpha]y)$$

we can build the following typing tree:

$$\dfrac{\dfrac{}{x:\neg\neg A \vdash x:\neg\neg A;\alpha:A} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{x:\neg\neg A, y:A \vdash y:A;\alpha:A,\beta:\bot}}{[\alpha]y:(x:\neg\neg A, y:A \vdash ;\alpha:A,\beta:\bot)}}{x:\neg\neg A, y:A \vdash \mu\beta.[\alpha]y:\bot;\alpha:A}}{x:\neg\neg A \vdash \lambda y.\mu\beta.[\alpha]y:\neg A;\alpha:A}}{\dfrac{\dfrac{\dfrac{x:\neg\neg A \vdash x\ \lambda y.\mu\beta.[\alpha]y:\bot;\alpha:A}{[\mathsf{top}](x\ \lambda y.\mu\beta.[\alpha]y):(x:\neg\neg A \vdash ;\alpha:A)}}{x:\neg\neg A \vdash \mu\alpha.[\mathsf{top}](x\ \lambda y.\mu\beta.[\alpha]y):A;}}{\vdash \lambda x.\mu\alpha.[\mathsf{top}](x\ \lambda y.\mu\beta.[\alpha]y):(\neg\neg A)\to A;}}$$

and we can perform the following reduction:

$$\begin{array}{rcl} E[\mathcal{C}\ M] & = & [\gamma](\lambda x.\mu\alpha.[\mathsf{top}](x\ \lambda y.\mu\beta.[\alpha]y))\ M\ N_1 \dots N_n \\ & \longrightarrow & [\gamma](\mu\alpha.[\mathsf{top}](M\ \lambda y.\mu\beta.[\alpha]y))\ N_1 \dots N_n \\ & \longrightarrow & [\gamma](\mu\alpha.[\mathsf{top}](M\ \lambda y.\mu\beta.[\alpha]y\ N_1))\ N_2 \dots N_n \\ & \longrightarrow & \dots \\ & \longrightarrow & [\gamma]\mu\alpha.[\mathsf{top}]M\ \lambda y.\mu\beta.[\alpha]y\ N_1 \dots N_n \\ & \longrightarrow & [\mathsf{top}]M\ \lambda y.\mu\beta.[\gamma]y\ N_1 \dots N_n \\ & = & [\mathsf{top}]M\ (\lambda y.\mu\beta.E[y]) \end{array}$$

※

Notice that what is eventually produced by the rewrites is not M and $M\ (\lambda y.\mu\beta.E[y])$, respectively, but $[\mathsf{top}]M$ and $[\mathsf{top}]M\ (\lambda y.\mu\beta.E[y])$, since reducing a command has to produce a command. But since M is of type $\bot$ (respectively produces a term of type $\bot$), it can be stored in the top-level continuation top.

Now, when thinking about classical logic, we often have in mind concepts of symmetry or duality:

Inversing the order in a boolean algebra provides another boolean algebra where e.g. the

top and bottom elements have been swapped, the meet and join operations have been swapped.

Very related to this are De Morgan's rules, which show a duality, via negation, between $\wedge$ and $\vee$:

$$\begin{aligned}\neg(A \wedge B) &= \neg A \vee \neg B \\ \neg(A \vee B) &= \neg A \wedge \neg B\end{aligned}$$

In terms of proof formalisms, the classical sequent calculus LK [Gen35] shows a symmetry between the left-hand side and the right-hand of sequents, of the form $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$: whatever can be done on the left-hand side can be done on the right-hand side, and vice versa. For instance, the left-introduction rule for $\wedge$ is symmetric to the right-introduction rule for $\vee$ and vice versa; left-contraction symmetric to right-contraction (and this is very different from intuitionistic logic).

But so far, such symmetries and dualities are not explicitly reflected in our proof-term approach to classical proofs.

However, before even Griffin made the connection between control operators and classical logic, Filinski [Fil89] formalised a duality between

- functions as values
- functions as continuations

in the form of a "symmetric λ-calculus", with explicit conversions from one view of functions to the other. Yet there was no explicit connection with classical logic.

In [BB96], Barbanera and Berardi formalised their own symmetric λ-calculus, with a typing system providing a Curry-Howard interpretation of classical proofs. The classical proof system depicted by their calculus is a one-sided version of the classical sequent calculus [Gen35], with a proof of normalisation for typed terms (we will see such a proof in Chapter 2).

Since then, two calculi emerged to provide Curry-Howard interpretations of the two-sided sequent calculus LK (or variants thereof), with the reduction rules describing the famous proof-transformation procedure known as *cut-elimination*:

- Urban's calculus [Urb00],
- Curien and Herbelin's $\overline{\lambda}\mu\widetilde{\mu}$ [CH00] for $\Rightarrow$,
  later extended by Wadler [Wad03] for $\wedge$ and $\vee$ (explicitly connecting the symmetries of the calculus to De Morgan's duality).

These two independent (sets of) contributions had different aims: Curien and Herbelin's was to expose, as the syntactic symmetry of the classical sequent calculus, a *duality* in computation based on Filinski's ideas about continuations and on the call-by-value and call-by-name evaluation strategies; they gave semantics to their calculus, but with no proof of normalisation. Urban's aim was to have a typing system as close as possible to LK and have a reduction system as close as possible to basic cut-elimination procedures; his Ph.D. adapted Barbanera and Berardi's proof of strong normalisation to his calculus, but gave no (denotational) semantics.

Several papers formalise the links between the various proof calculi for classical logic: in particular, [Len03] relates $\overline{\lambda}\mu\widetilde{\mu}$ and Urban's calculus, [Roc05] relates $\overline{\lambda}\mu\widetilde{\mu}$ and $\lambda\mu$, and [Her05] presents an extensive exploration of the relations between the various calculi.

In the rest of this chapter, we focus on Curien and Herbelin's calculus to explore some more semantical concepts, but many of them can be transposed to other calculi for classical logic (in particular, Parigot's $\lambda\mu$-calculus).

1.4 System L

System L is the new name of Curien and Herbelin's $\overline{\lambda}\mu\widetilde{\mu}$, extended with other connectives. We start with its syntax:

DEFINITION 11 (Syntax) The syntax of L is made of three syntactic categories:

$$\begin{array}{lcl} \text{terms} & t & ::= x \mid \mu\beta.c \mid \lambda x.t \mid (t_1, t_2) \mid \mathsf{inj}_i(t) \\ \text{continuations} & e & ::= \alpha \mid \mu x.c \mid t :: e \mid (e_1, e_2) \mid \mathsf{inj}_i(e) \\ \text{commands} & c & ::= \langle t \mid e \rangle \end{array}$$

where i ranges over $\{1, 2\}$, x and α respectively range over term variables and continuation variables,[8] $\mu\beta.c$ binds β in c, $\mu x.c$ binds x in c, and $\lambda x.t$ binds x in t. The scope of binders extends as much as parentheses allow. ※

A (somewhat shallow) intuition of the syntax can be given as follows:

Term variables x, y, ...	denote inputs	
Continuation variables α, β, ...	denote outputs	
A *term* has	one main output	
	some inputs	(free term variables)
	some "alternative" outputs	(free continuation variables)
A *continuation* has	one main input	
	some "additional" inputs	(free term variables)
	some possible outputs	(free continuation variables)
A *command* is	a term facing a continuation	(the interaction is computation)

DEFINITION 12 (Typing)

We consider the following grammar for *types* (extending that of simple types):

$$A, B, \ldots ::= a \mid A \to B \mid A \wedge B \mid A \vee B$$

where a ranges over a fixed set of elements called *atomic types*. The symbol $\to$ is associative to the right, i.e. $A \to (B \to C)$ can be abbreviated as $A \to B \to C$.

The typing system for System L is given for three kinds of sequents corresponding to the three syntactic categories of the syntax:

$$\Gamma \vdash t : A \mid \Delta \qquad \Gamma \mid e : A \vdash \Delta \qquad c : (\Gamma \vdash \Delta)$$

where Γ is a typing context for term-variables and Δ is a typing context for continuation-variables.

The system is presented in Fig. 6. Derivability of sequents in this system is respectively denoted $\Gamma \vdash_{\mathsf{L}} t : A \mid \Delta$, $\Gamma \mid e : A \vdash_{\mathsf{L}} \Delta$, and $c : (\Gamma \vdash_{\mathsf{L}} \Delta)$. ※

As we can see, forgetting about variables and proof-terms does not give the sequent calculus LK exactly as we know it from [Gen35] or as the popular variants described in [TS00] (for this one can look at Urban's Ph.D. [Urb00]), if only because there are three types of sequents. However, it is a variant with a bit more structure, which defines the same notion of provability as LK, and which will prove useful for the computational interpretation of classical logic.

An intuition about this interpretation can be given as follows: similarly to the Curry-Howard correspondence in intuitionistic logic, each connective in the syntax of formulae corresponds to a type construct in programming; term constructs offer basic ways in which such

[8] As in Parigot's $\lambda\mu$-calculus [Par92].

$$\frac{}{\Gamma, x{:}A \vdash x{:}A \;;\; \Delta} \qquad \frac{}{\Gamma \;;\; \alpha{:}A \vdash \alpha{:}A, \Delta}$$

$$\frac{\Gamma, x{:}A \vdash t{:}B \;;\; \Delta}{\Gamma \vdash \lambda x.t{:}A{\to}B \;;\; \Delta} \qquad \frac{\Gamma \vdash t{:}A \;;\; \Delta \quad \Gamma \;;\; e{:}B \vdash \Delta}{\Gamma \;;\; t{::}e{:}A{\to}B \vdash \Delta}$$

$$\frac{\Gamma \vdash t_1{:}A_1 \;;\; \Delta \quad \Gamma \vdash t_2{:}A_2 \;;\; \Delta}{\Gamma \vdash (t_1, t_2){:}A_1 \wedge A_2 \;;\; \Delta} \qquad \frac{\Gamma \;;\; e{:}A_i \vdash \Delta}{\Gamma \;;\; \mathsf{inj}_i(e){:}A_1 \wedge A_2 \vdash \Delta}$$

$$\frac{\Gamma \vdash t{:}A_i \;;\; \Delta}{\Gamma \vdash \mathsf{inj}_i(t){:}A_1 \vee A_2 \;;\; \Delta} \qquad \frac{\Gamma \;;\; e_1{:}A_1 \vdash \Delta \quad \Gamma \;;\; e_2{:}A_2 \vdash \Delta}{\Gamma \;;\; (e_1, e_2){:}A_1 \vee A_2 \vdash \Delta}$$

$$\frac{c{:}(\Gamma \vdash \alpha{:}A, \Delta)}{\Gamma \vdash \mu\alpha.c{:}A \;;\; \Delta} \qquad \frac{c{:}(\Gamma, x{:}A \vdash \Delta)}{\Gamma \;;\; \mu x.c{:}A \vdash \Delta}$$

$$\frac{\Gamma \vdash t{:}A \;;\; \Delta \quad \Gamma \;;\; e{:}A \vdash \Delta}{\langle t \mid e \rangle \,{:}(\Gamma \vdash \Delta)}$$

Figure 6: Typing system for L

types can be inhabited, while continuation constructs offer basic ways in which inhabitants of such types are consumed:

- A conjunction $A_1 \wedge A_2$ corresponds to a product type, so basic inhabitants are pairs (t_1, t_2) of terms (with the first component inhabiting A_1 and the second inhabiting A_2); basic continuations that consume such a pair start by extracting either the first or the second component (in other words, they start with one of the two projections), which corresponds to the continuation constructs $\mathsf{inj}_1(e)$ and $\mathsf{inj}_2(e)$.
- A disjunction $A_1 \vee A_2$ corresponds to a sum type, so basic inhabitants are the injections $\mathsf{inj}_1(t)$ and $\mathsf{inj}_2(t)$ (with t inhabiting A_1 or inhabiting A_2, respectively); basic continuations that consume such an injection must handle both cases, so the case analysis leads to providing a pair $\langle e_1, e_2 \rangle$ of two continuations: the former can consume inhabitants of A_1 and the latter can consume inhabitants of A_2.
- An implication $A_1 \Rightarrow A_2$ corresponds to a function type, with the basic inhabitants being constructed with λ-abstractions just like in the λ-calculus; we do not have the construct that directly applies a function to an argument, but a basic way in which a continuation consumes a function is to offer an argument t as the input of the function, together with a continuation e that can consume the output of the function; hence the continuation construct $t{::}e$ (which is simply the usual stacking construct that can be found in abstract machines to implement computation in the λ-calculus).

This intuition will be strengthened by the reduction rules for System L, but we first start with an example.

The following story is borrowed from Phil Wadler [Wad03] (who might have borrowed it from Peter Selinger), and illustrates the computational contents of classical proofs:

EXAMPLE 2 (The devil, the fool, and the $1,000,000)

> The Devil meets a man and says:
> "- I have an offer for you! I promise you that
> *either I offer you $1,000,000 or, if you give me $1,000,000, then I will grant you any wish.*
> Actually, I choose to offer you the latter."
> The man then goes back home and, motivated by the Devil's promise, strives to gather $1,000,000. Ten years later, he finally succeeds; he goes back to the Devil and, handing him the money, says:
> "- Here's $1,000,000! I want immortality."
> The Devil takes the money and says:
> "- Well done and thank you!
> Actually, I've changed my mind. I've now decided to fulfil my promise by offering you $1,000,000. Here is your money back!" ※

The reason why that short story illustrates the computational contents of classical logic is that the Devil behaves as a proof of the Law of Excluded Middle: Imagine that

- the money ($1,000,000) can be seen as an atomic proposition a
- the part of the promise "If you give me $1,000,000, I'll grant you any wish" can be seen as the formula $a \Rightarrow \bot$, i.e. $\neg a$;

the Devil is then the proof of $a \vee \neg a$ shown in Fig. 7. Indeed, following the bottom-up

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y:a \vdash y:a \;;\; \alpha:a\vee\neg a, \beta:\bot}}{y:a \vdash \mathsf{inj}_1(y):a\vee\neg a \;;\; \alpha:a\vee\neg a, \beta:\bot} \quad \dfrac{}{y:a \;;\; \alpha:a\vee\neg a \vdash \alpha:a\vee\neg a, \beta:\bot}}{\langle \mathsf{inj}_1(y) \mid \alpha\rangle : (y:a \vdash \alpha:a\vee\neg a, \beta:\bot)}}{y:a \vdash \mu\beta.\langle \mathsf{inj}_1(y) \mid \alpha\rangle : \bot \;;\; \alpha:a\vee\neg a}}{\vdash \lambda y.\mu\beta.\langle \mathsf{inj}_1(y) \mid \alpha\rangle : \neg a \;;\; \alpha:a\vee\neg a}}{\vdash \mathsf{inj}_2(\lambda y.\mu\beta.\langle \mathsf{inj}_1(y) \mid \alpha\rangle) : a\vee\neg a \;;\; \alpha:a\vee\neg a} \quad \dfrac{}{;\; \alpha:a\vee\neg a \vdash \alpha:a\vee\neg a}}{\dfrac{\langle \mathsf{inj}_2(\lambda y.\mu\beta.\langle \mathsf{inj}_1(y) \mid \alpha\rangle) \mid \alpha\rangle : (\vdash \alpha:a\vee\neg a)}{\vdash \mu\alpha.\langle \mathsf{inj}_2(\lambda y.\mu\beta.\langle \mathsf{inj}_1(y) \mid \alpha\rangle) \mid \alpha\rangle : a\vee\neg a \;;}}$$

Figure 7: A proof of LEM

construction of the left-hand branch, we see that

- the proof (the Devil) starts by choosing to prove $\neg a$, as reflected by the $\mathsf{inj}_2(\_)$ construct;
- that requires an input of type a (the $\$1,000,000$ earned by the fool), namely y, as reflected by the $\lambda y.\_$ construct;
- given the impossibility to prove $\bot$ directly (the immortality wish, or for that matter, any wish), the proof re-attacks the original formula to prove, namely $a\vee\neg a$ (the Devil returns to his original promise), but this time with the input $y:A$ (the $\$1,000,000$ that the fool gave him);
- this time, the proof chooses to prove a, which is trivially done by returning y (the Devil chooses to give $\$1,000,000$, by returning the money that the man earned).

  We see here that the proof works because of the possibility to construct an inhabitant of

$a \vee \neg a$, twice along the same branch (we inhabit it the first time with the second injection, then with the first one), which is technically allowed by the *right-contraction* implicitly featured in the bottom two steps of the proof. While in intuitionistic logic it is possible to contract on the left but not contract on the right, classical logic allows both symmetrically.

This allows to also build a proof-term of type PL and, allowing again (as in Parigot's $\lambda\mu$) a top-level continuation variable $\mathtt{top}$ of type $\bot$, we can build proof-terms for *Ex falso quodlibet* and the elimination of double negation.

In summary, we have seen that it is easy enough to introduce proof-terms to represent classical proofs, such that the symmetry of classical logic reflects the symmetry between programs and continuations.

The use of classical reasoning corresponds to the use of control features allowing programs to capture their continuation, as we now see by looking at reductions:

Definition 13 (Reductions) The reductions are given by the following rewrite system:

$$\begin{array}{lccc}
(\to) & \langle \lambda x.t_1 \mid t_2 :: e\rangle & \longrightarrow & \langle t_2 \mid \mu x.\langle t_1 \mid e\rangle\rangle \\
(\wedge) & \langle (t_1,t_2) \mid \mathsf{inj}_i(e)\rangle & \longrightarrow & \langle t_i \mid e\rangle \\
(\vee) & \langle \mathsf{inj}_i(t) \mid (e_1,e_2)\rangle & \longrightarrow & \langle t \mid e_i\rangle \\
(\overleftarrow{\mu}) & \langle \mu\beta.c \mid e\rangle & \longrightarrow & \{^{e}/_{\beta}\}\, c \\
(\overrightarrow{\mu}) & \langle t \mid \mu x.c\rangle & \longrightarrow & \{^{t}/_{x}\}\, c
\end{array}$$

※

Now, while it was very clear that Parigot's $\lambda\mu$ forms an extension of λ-calculus, we should emphasise the fact that the λ-calculus can be encoded in System L:

Definition 14 (Encoding of λ-calculus)

We encode λ-terms as terms of System L by first encoding values, then all terms:

$$\begin{array}{lcl}
\overline{x}^{\mathrm{V}} & := & x \\
\overline{\lambda x.M}^{\mathrm{V}} & := & \lambda x.\overline{M}
\end{array}
\qquad
\begin{array}{l}
\overline{V\ M_1\ \ldots\ M_n} := \mu\alpha.\left\langle \overline{V}^{\mathrm{V}} \mid \overline{M_1} :: \ldots \overline{M_n} :: \alpha \right\rangle \\
\qquad \text{where } V \text{ is not an application and } n \geq 0
\end{array}$$

※

Lemma 9 (Simulation of λ-calculus)

1. $\mu\alpha.\left\langle \overline{M} \mid \overline{M_1} :: \ldots \overline{M_n} :: \alpha \right\rangle \longrightarrow \overline{M\ M_1\ \ldots\ M_n}$.
2. $\left\{^{\overline{N}}/_{x}\right\} \overline{M} \longrightarrow^{*} \overline{\{^{N}/_{x}\}\, M}$.
3. If $M \longrightarrow_{\beta} N$ then $\overline{M} \longrightarrow^{*} \overline{N}$.

※

Proof: The first point is a simple $\overleftarrow{\mu}$-reduction, the second point is by induction on M, the third point is by induction on the rewrite derivation. □

Lemma 10 (Preservation of simple types)

1. If $\Gamma \vdash_{\mathsf{st}\lambda} V : A$ then $\Gamma \vdash_{\mathsf{L}} \overline{M}^{\mathrm{V}} : A$;
2. If $\Gamma \vdash_{\mathsf{st}\lambda} M : A$ then $\Gamma \vdash_{\mathsf{L}} \overline{M} : A$;

※

Since System L contains cuts, a proof of LEM, and it can encode simply-typed λ-terms, it is clearly complete for classical logic (in the same sense as for the $\lambda\mu$-calculus: EDN and EFQ require the presence of a top-level continuation variable $\mathsf{top} : \bot$). Soundness can be trivially

checked by checking that all the inference rules are sound when forgetting about variables and proof-terms.

Substitution behaves well with respect to typing:

THEOREM 11 (Substitution Lemma)

1. If $c:(\Gamma, x:A \vdash_{\mathsf{L}} \Delta)$ and $\Gamma \vdash_{\mathsf{L}} t:A \ ; \Delta$ then $\{^{t}/_{x}\}\, c:(\Gamma \vdash_{\mathsf{L}} \Delta)$.
2. If $c:(\Gamma \vdash_{\mathsf{L}} \alpha:A, \Delta)$ and $\Gamma \ ; e:A \vdash_{\mathsf{L}} \Delta$ then $\{^{e}/_{\alpha}\}\, c:(\Gamma \vdash_{\mathsf{L}} \Delta)$.

※

Proof: By induction on c, simultaneously proving the two analogous properties for both terms and continuations. □

And the reduction relation satisfies Subject Reduction:

THEOREM 12 (Subject reduction for System L)

1. If $c:(\Gamma \vdash_{\mathsf{L}} \Delta)$ and $c \longrightarrow c'$ then $c':(\Gamma \vdash_{\mathsf{L}} \Delta)$.
2. If $\Gamma \vdash_{\mathsf{L}} t:A \ ; \Delta$ and $t \longrightarrow t'$ then $\Gamma \vdash_{\mathsf{L}} t':A \ ; \Delta$.
3. If $\Gamma \ ; e:A \vdash_{\mathsf{L}} \Delta$ and $e \longrightarrow e'$ then $\Gamma \ ; e':A \vdash_{\mathsf{L}} \Delta$.

※

Proof: Straightforward induction on the rewrite derivations. □

Again, Subject Reduction allows the rewrite system to describe a proof transformation procedure in the classical sequent calculus, and in this case it is *cut-elimination* [Gen35].

Let us see the other properties we mentioned when introducing the Curry-Howard methodology:

Progress depends of course on what we consider an "undesirable structure". In the case of sequent calculus, the natural concept of undesirable structure is the cut, which in the typing system of L is (at least at first sight) represented as the bottom-most rule of Fig. 6. And at this point we notice that some cuts cannot be reduced, as no rewrite rule applies to their proof-terms, namely those of the form $\langle x \mid e \rangle$ and $\langle t \mid \alpha \rangle$ when e is not of the form $\mu x.c$ and t is not of the form $\mu\alpha.c$. We may think progress fails (in terms of cut-elimination), but we should also notice that cuts of that form are very peculiar: they do nothing but respectively implement a left-contraction or a right-contraction, two rules that the extra structure of the system requires for completeness (compared to e.g. G3ii [TS00]):

$$\frac{\Gamma, x:A \vdash x:A \ ; \Delta \qquad \Gamma, x:A \ ; e:A \vdash \Delta}{\langle x \mid e \rangle : (\Gamma, x:A \vdash \Delta)} \qquad \frac{\Gamma \vdash t:A \ ; \alpha:A, \Delta \qquad \Gamma \ ; \alpha:A \vdash \alpha:A, \Delta}{\langle t \mid \alpha \rangle : (\Gamma \vdash \alpha:A, \Delta)}$$

We actually used two of these special "cuts" in the proof of LEM showed in Fig. 7, and we would not expect to eliminate them (unless we had specific constructs for contractions and for the axiom represented as $\langle x \mid \alpha \rangle$).

Concerning normalisation, it can be proved that typed commands (resp. terms, continuations) are strongly normalising. This was inferred from Urban's calculus in [Len03], but can be more simply obtained as the direct application of Barbanera and Berardi's technique, as shown in [Pol04] for a variant of System L with explicit substitutions. This will be the topic of Chapter 2.

Finally, we look at the confluence property.

1.5 Non-confluence of cut-elimination in classical logic

As the reduction relation of System L specifies a cut-elimination procedure, we should note that cut-elimination in classical logic, at a purely logical level, can easily be defined as a non-confluent transformation procedure. A typical example of this is Lafont's example, which we first express in the original sequent calculus LK, with explicit rules for weakenings and contractions and "context-splitting" rules (see e.g. [TS00]):

Example 3 (Lafont's example for non-confluence)

Consider the following cut that we would like to eliminate

$$\dfrac{\dfrac{\vdots\,\pi}{\Gamma \vdash \Delta}}{\dfrac{\Gamma \vdash \Delta, A}{\Gamma, \Gamma' \vdash \Delta, \Delta'}} \qquad \dfrac{\dfrac{\vdots\,\pi'}{\Gamma' \vdash \Delta'}}{\Gamma', A \vdash \Delta'}$$

There are two ways to eliminate the cut:

$$\dfrac{\vdots\,\pi}{\Gamma \vdash \Delta}{\Gamma, \Gamma' \vdash \Delta, \Delta'} \quad \text{or} \quad \dfrac{\vdots\,\pi'}{\Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

※

This obviously leads to non-confluence as soon as π and π' are two distinct proofs (say, cut-free). Note that we could, somewhat artificially, avoid the choice between π and π' by considering the following *mix* rule [FR94]:

$$\dfrac{\Gamma \vdash \Delta \quad \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

which would allow the symmetric combination of π and π' into a single proof (and what would be the semantics of this combination?). The question is whether we accept this derivation as a normal proof. Let us look at the same example in a sequent calculus (such as G3ii [TS00]) where rules are "context-sharing":

Example 4 (Lafont's example in a context-sharing sequent calculus)

The following cut:

$$\dfrac{\dfrac{\vdots\,\pi}{\Gamma \vdash \Delta}}{\dfrac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta}} \qquad \dfrac{\dfrac{\vdots\,\pi'}{\Gamma \vdash \Delta}}{\Gamma, A \vdash \Delta}$$

can be reduced to:

$$\dfrac{\vdots\,\pi}{\Gamma \vdash \Delta} \quad \text{or} \quad \dfrac{\vdots\,\pi'}{\Gamma \vdash \Delta}$$

※

What is even more striking in this example is that π and π' are two proofs of the same sequent, which we probably do not want to consider denotationally equal and whose combination via the following rule

$$\frac{\Gamma \vdash \Delta \quad \Gamma \vdash \Delta}{\Gamma \vdash \Delta}$$

looks even more artificial than with the context-splitting mix. Again, do we want this derivation as a normal proof?

Unsatisfying though the mix may seem, it does technically solve Lafont's non-confluence problem based on two weakenings. Unfortunately, it cannot solve the even more problematic example obtained with contractions instead of weakenings:

EXAMPLE 5 (Example with contractions) The following cut

$$\dfrac{\dfrac{\vdots\,\pi}{\Gamma \vdash \Delta, A, A}}{\Gamma \vdash \Delta, A} \qquad \dfrac{\vdots\,\pi'}{\Gamma, A, A \vdash \Delta}}{\Gamma \vdash \Delta}$$

can be reduced to

$$\begin{array}{c}\vdots\,\pi' \quad \vdots\,\pi' \\ \bullet \quad\quad \bullet \\ \vdots\,\pi(\simeq) \\ \Gamma \vdash \Delta\end{array} \quad \text{or} \quad \begin{array}{c}\vdots\,\pi \quad \vdots\,\pi \\ \bullet \quad\quad \bullet \\ \vdots\,\pi'(\simeq) \\ \Gamma \vdash \Delta\end{array}$$

where $\pi(\simeq)$ (resp. $\pi'(\simeq)$) denotes the proof π (resp. π') modified by the propagation of π' (resp. π) into its structure.

We can give a concrete instance of the above:

$$\frac{(A \to B) \to A \vdash A \qquad A, A \to C, A \to D \vdash C \wedge D}{(A \to B) \to A, A \to C, A \to D \vdash C \wedge D}$$

Peirce's Law requires a right-contraction on the cut-formula A while the right-hand side proof requires a left-contraction on the cut-formula A. ※

Coming back to the proof-term side, both examples would appear in System L as instances of the following scheme:

$$\frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \;;\; \Delta} \qquad \frac{c' : (\Gamma, x : A \vdash \Delta)}{\Gamma \;;\; \mu x.c' : A \vdash \Delta}}{\langle \mu\alpha.c \mid \mu x.c' \rangle : (\Gamma \vdash \Delta)}$$

α (resp. x) could be used 0 (weakening), 1, or several (contraction) times in c (resp. c').

That cut could be reduced to

$$\dfrac{c : (\Gamma \vdash \alpha : A, \Delta) \qquad \dfrac{c' : (\Gamma, x : A \vdash \Delta)}{\Gamma \;;\; \mu x.c' : A \vdash \Delta}}{\left\{ {}^{\mu x.c'}\!/_{\alpha} \right\} c : (\Gamma \vdash \Delta)} \quad \text{or} \quad \dfrac{\dfrac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \;;\; \Delta} \qquad c' : (\Gamma, x : A \vdash \Delta)}{\{ {}^{\mu\alpha.c}\!/_{x} \}\, c' : (\Gamma \vdash \Delta)}$$

where dotted lines do not represent primitive inference rules, but inference rules that have been shown admissible in the typing system (Lemma 11).

In the case of weakening, and reflecting Example 4, $\alpha \notin \mathsf{FV}(c)$ and $x \notin \mathsf{FV}(c')$ and we can reduce $\langle \mu\alpha.c \mid \mu x.c' \rangle$ to two arbitrary commands c and c' with the same type.

This makes it hard to give a denotational semantics of classical proofs or of typed proof-terms: if we require $\langle \mu\alpha.c \mid \mu x.c' \rangle$, $\left\{ {}^{\mu x.c'}\!/_{\alpha} \right\} c$, and $\{ {}^{\mu\alpha.c}\!/_{x} \}\, c'$, to have the same denotation,

Example 4 leads to giving the same denotation to every proof of the same sequent.

This of course relates to the fact that we have already mentioned: a CCC with initial object where every object A naturally isomorphic to $\neg\neg A$ collapses to a boolean algebra. As identified in [Str11], there are three natural ways to resolve this:

1. Break the symmetry between $\wedge$ and $\vee$
2. Break the cartesian product (as studied for instance in [FP06, LS05, Lam07, Str11])
3. Break curryfication (as studied for instance in [DP04, CS09])

In these notes, we break the symmetry between $\wedge$ and $\vee$, since out of the three solutions it is the one for which the Curry-Howard correspondence with programs is best understood.

One way of breaking the non-confluence problem

$$\dfrac{\begin{array}{cc}\vdots\,\pi & \vdots\,\pi' \\ \Gamma \vdash \Delta, A & \Gamma, A \vdash \Delta\end{array}}{\Gamma \vdash \Delta}$$

is simply to give systematic priority to

- the right (push π into π')
- or to the left (push π' into π)

Almost by definition, both solutions make the calculus confluent.

They also break the $\wedge\vee$ symmetry: Giving systematic priority to the right, say, makes a term t of type $A\wedge B$ have the same behaviour as $(\mathsf{inj}_1(t), \mathsf{inj}_2(t))$, whereas a continuation e of type $A\vee B$ will not necessarily have the same behaviour as $(\mathsf{inj}_1(e), \mathsf{inj}_2(e))$.

More details on this will be given in Chapter 3, but we shall also see by semantical means that the $\wedge\vee$ symmetry is broken. The two reduction strategies suggest to construct two denotational semantics $[\![c]\!]_{\mathsf{N}}$ and $[\![c]\!]_{\mathsf{V}}$ with the hope that:

$[\![c_0]\!]_{\mathsf{N}} = [\![c_1]\!]_{\mathsf{N}}$ iff "$c_0 \longleftrightarrow^* c_1$ with systematic priority to the right"
$[\![c_0]\!]_{\mathsf{V}} = [\![c_1]\!]_{\mathsf{V}}$ iff "$c_0 \longleftrightarrow^* c_1$ with systematic priority to the left"

The use of the letters N and V reflects the fact that the strategies relate to the notions of Call-by-name and Call-by-value, as investigated for instance by Plotkin [Plo75], Moggi [Mog89], and others.

In conclusion of this section, we have seen that it is easy enough to give a rewrite system on proof-terms to represent cut-elimination (and the system follows the intuitions of continuations and control), but it gives a non-confluent calculus because cut-elimination is non-confluent in classical logic (via the Curry-Howard correspondence, because programs and continuations fight for the control of computation).

The rest of this chapter is devoted to the construction of the above CBN and CBV semantics.

1.6 Continuations, Call-by-Name and Call-by-Value

Call-by-name and call-by-value are two strategies for evaluating programs. Imagine the definition of a function (in pseudo-code):

```
MyFavoriteFunction(x){
  ... x ...
}
```
and later a call to that function with argument `A`:
```
MyFavoriteFunction(A)
```

The main question is whether `A` should be evaluated before entering the (code of the) function (CBV) or when it is actually used (CBN)? This is a question of interpretation or compilation of programs and, especially in presence of side-effects, knowing which of the two the compiler implements, is vital for the determinism of evaluation.

In general, we call *values* what evaluation should produce (e.g. booleans true, false). In functional programming, functions are particular values and can be passed as arguments. In general, functions are therefore not reduced.

The λ-calculus is both a core functional language and a theory of functions.

As a core functional language, it is equipped with an operational semantics, close to implementation, which can be expressed by an evaluation strategy that selects a unique β-redex to reduce:

- Never reduce a λ-abstraction, as it is a "value" (this is called *weak reduction*)
- Always reduce M first in an application $M\ N$. Then:
  - If M is an abstraction: reduce the β-redex first (CBN)
    reduce N first (CBV)
  - Otherwise, reduce N (never happens with closed terms)

We denote those strategies $\longrightarrow_{\mathsf{CBN}}$ and $\longrightarrow_{\mathsf{CBV}}$.[9]

As a theory of functions, the λ-calculus is equipped with a denotational semantics close to the mathematical notion of functions: in particular, equalities are congruences (e.g. if $M = N$ then $\lambda x.M = \lambda x.N$) and reductions are congruences (this is called *strong reduction*). In [Plo75], Plotkin investigated the concepts of call-by-name and call-by-value by identifying particular λ-terms as values:

DEFINITION 15 (Value, β_v, Call-by-Name and Call-by-Value)

λ-terms of the form $\lambda x.M$ and x are called *values*, and denoted V, V', etc, while λ-terms of the form MN are not values.[10]

- "Call-by-name" evaluation is given by general β-reduction
$$(\beta) \qquad (\lambda x.M)\ N \longrightarrow \left\{{}^{N}\!/_{x}\right\} M$$
- "Call-by-value" evaluation is given by the restriction of β-reduction where the argument is a value
$$(\beta_v) \qquad (\lambda x.M)\ V \longrightarrow \left\{{}^{V}\!/_{x}\right\} M$$

※

Now, natural questions to raise are

CBN: whether there is a relation between $\longrightarrow_{\mathsf{CBN}}$ and $\longrightarrow_{\beta}$;

CBV: whether there is a relation between $\longrightarrow_{\mathsf{CBV}}$ and $\longrightarrow_{\beta_v}$?

[9] For instance, Haskell implements CBN while OCaml implements CBV.

[10] The intuition is that, by evaluating MN, you may get a λ-term of a completely different shape.

Clearly, $\longrightarrow_{\mathsf{CBN}} \subseteq \longrightarrow_\beta$ and $\longrightarrow_{\mathsf{CBV}} \subseteq \longrightarrow_{\beta_v}$, but what about the converse? A bridge between weak and strong reductions was given by Plotkin [Plo75]:

Theorem 13 (CBN and CBV: weak and strong reductions)

CBN: $\longrightarrow_\beta^*$ is the closure of $\longrightarrow_{\mathsf{CBN}}^*$ under the rules

$$\frac{M_1 \longrightarrow_{\mathsf{CBN}}^* \lambda x.M_2 \quad M_2 \longrightarrow M_2'}{M_1 \longrightarrow \lambda x.M_2'} \qquad \frac{M_1 \longrightarrow_{\mathsf{CBN}}^* M_2\ M_3 \quad M_2 \longrightarrow M_2' \quad M_3 \longrightarrow M_3'}{M_1 \longrightarrow M_2'\ M_3'}$$

CBV: $\longrightarrow_{\beta_v}^*$ is the closure of $\longrightarrow_{\mathsf{CBV}}^*$ under the rules

$$\frac{M_1 \longrightarrow_{\mathsf{CBV}}^* \lambda x.M_2 \quad M_2 \longrightarrow M_2'}{M_1 \longrightarrow \lambda x.M_2'} \qquad \frac{M_1 \longrightarrow_{\mathsf{CBV}}^* M_2\ M_3 \quad M_2 \longrightarrow M_2' \quad M_3 \longrightarrow M_3'}{M_1 \longrightarrow M_2'\ M_3'}$$

※

The point of this result is that we shall now call CBN and CBV, not some operational semantics of some functional programming language, but some rewriting theories in λ-calculus.

As we have already mentioned compilation in reference to CBN/CBV, it is interesting to see that λ-calculus can be compiled into (a fragment of) itself: this is based on the idea of the program transformation presented in Section 1.2 using continuations. As continuations are passed as an extra argument to every call, such transformations are known as *Continuation Passing Style* (CPS)-translations.

Definition 16 (CPS-translations)

Two important CPS-translations were defined for CBN and CBV:

CBN-translation (Plotkin [Plo75])			CBV-translation (Reynolds [Rey72])		
$\underline{x}$	$:=$	$\lambda k.x\ k$	$\overline{x}$	$:=$	$\lambda k.k\ x$
$\underline{\lambda x.M}$	$:=$	$\lambda k.k\ (\lambda x.\underline{M})$	$\overline{\lambda x.M}$	$:=$	$\lambda k.k\ (\lambda x.\lambda k'.\overline{M}\ k')$
$\underline{M\ N}$	$:=$	$\lambda k.\underline{M}\ (\lambda y.y\ \underline{N}\ k)$	$\overline{M\ N}$	$:=$	$\lambda k.\overline{M}\ (\lambda y.\overline{N}\ (\lambda z.y\ z\ k))$

where the variables k and k' are always chosen to be fresh. ※

One main feature of these translations is their target fragment of the λ-calculus: in this fragment, arguments are always values! This fragment is stable under $\longrightarrow_\beta$ and $\longrightarrow_{\beta_v}$, which actually coincide. The evaluation of a CPS-translated term is *strategy-indifferent*. How this evaluation relates to the evaluation of the original term is given by the following simulation properties:

Theorem 14 (CPS-translations preserve reductions)

Soundness:

CBN If $M \longrightarrow_\beta N$ then $\underline{M} \longrightarrow_\beta^* \underline{N}$

CBV If $M \longrightarrow_{\beta_v} N$ then $\overline{M} \longrightarrow_\beta^* \overline{N}$

Completeness:

CBN If $\underline{M} \longleftrightarrow_\beta^* \underline{N}$ then $M \longleftrightarrow_\beta^* N$

CBV It is **not** the case for CBV, that if $\overline{M} \longleftrightarrow_\beta^* \overline{N}$ then $M \longleftrightarrow_{\beta_v}^* N$.

※

Proof: It is interesting to look at soundness to see how or why the CPS-translations make sense; for complete proofs, see [Plo75].

$$\begin{array}{rcl}
\underline{(\lambda x.M)\ N} & = & \lambda k.(\lambda k'.(k'\ (\lambda x.\underline{M})))\ (\lambda y.y\ \underline{N}\ k) \\
& \longrightarrow & \lambda k.(\lambda y.y\ \underline{N}\ k)\ (\lambda x.\underline{M}) \\
& \longrightarrow & \lambda k.(\lambda x.\underline{M})\ \underline{N}\ k \\
& \longrightarrow & \lambda k.(\underline{\left\{{}^{N}\!/_{x}\right\}\underline{M}})\ k \\
& = & \lambda k.(\underline{\left\{{}^{N}\!/_{x}\right\}M})\ k \\
& \longrightarrow & \underline{\left\{{}^{N}\!/_{x}\right\}M}
\end{array}$$

The second equality of course relies on the property that the CPS-translation behaves well with substitution: $(\underline{\left\{{}^{N}\!/_{x}\right\}\underline{M}} = \underline{\left\{{}^{N}\!/_{x}\right\}M})$. The last rewrite is an instance of β-reduction because $\underline{\left\{{}^{N}\!/_{x}\right\}M}$ necessarily starts with a λ-abstraction (i.e. we are not using η-reduction).

$$\begin{array}{rcll}
\overline{(\lambda x.M)\ V} & = & \lambda k.(\lambda k'.(k'\ (\lambda x k''.\overline{M}\ k'')))\ (\lambda y.\overline{V}\ (\lambda z.y\ z\ k)) & \\
& \longrightarrow & \lambda k.(\lambda y.\overline{V}\ (\lambda z.y\ z\ k))\ (\lambda x k''.\overline{M}\ k'') & \\
& \longrightarrow & \lambda k.\overline{V}\ (\lambda z.(\lambda x k''.\overline{M}\ k'')\ z\ k) & \\
& \longrightarrow & \lambda k.\overline{V}\ (\lambda z.(\lambda k''.\{{}^{z}\!/_{x}\}\,\overline{M}\ k'')\ k) & \\
& = & \lambda k.\overline{V}\ (\lambda x.(\lambda k''.\overline{M}\ k'')\ k) & \\
& \longrightarrow & \lambda k.\overline{V}\ (\lambda x.\overline{M}\ k) & \\
& \longrightarrow & \lambda k.(\lambda x.\overline{M}\ k)\ H & \text{where } \overline{V} = \lambda k.k\ H \\
& \longrightarrow & \lambda k.(\underline{\left\{{}^{H}\!/_{x}\right\}\overline{M}})\ k & \\
& = & \underline{\lambda k.(\{{}^{V}\!/_{x}\}\,M)\ k} & \\
& \longrightarrow & \underline{\{{}^{V}\!/_{x}\}\,M} &
\end{array}$$

The second equality is simply the renaming of z into x; the third one relies again on the property that the CPS-translation behaves well with substitution by values $(\underline{\left\{{}^{H}\!/_{x}\right\}\underline{M}} = \underline{\left\{{}^{V}\!/_{x}\right\}M}$ if $\underline{V} = \lambda k.kH)$. The last rewrite is again an instance of β-reduction, not η-reduction.

The point here is to realise that if V had not been a value, then $\overline{V}$ would not be of the form $\lambda k.kH$, and the simulation of this specific β-reduction would be stuck. □

The above simulations give some intuition about the encodings: the translation of any term M starts with a λ-abstract on a fresh variable k that is used exactly once. The variable k stands for the current continuation (hence the expression *continuation-passing style*). In the encoding of an abstraction $\lambda x.M$ (which is a value), the current continuation is applied to the encoding of the body M under a λ-abstraction on x. In case of an application $M\ N$, the current continuation is not directly applied, but wrapped in a bigger continuation that is passed as an argument to the encoding of M; what this wrapping exactly is depends on whether we do CBN or CBV and will determine whether we reflect the evaluation of N as a value V before we reflect the reduction of $M\ N$.

Now, the fact that $\overline{M} \longleftrightarrow^*_\beta \overline{N}$ does not imply $M \longleftrightarrow^*_{\beta_v} N$ is slightly disappointing: one way to look at it is to consider that $M \longleftrightarrow^*_{\beta_v} N$ is too weak, or incomplete, for Call-by-Value. Indeed, the inspiration from monads, and Moggi's monadic λ-calculus [Mog89], has allowed the extension of the Call-by-Value λ-calculus into a sound and complete calculus with respect to the CBV CPS-translation (see for instance [Len06]).

We now turn to the behaviour of the CPS-translations with respect to typing: Assume we have $\Gamma \vdash M : A$. Do we have: $\Gamma' \vdash \underline{M} : A'$ (for some Γ', A') and $\Gamma'' \vdash \overline{M} : A''$ (for some Γ'', A'')?

The CPS-translations reveal two classes of terms in the target: *values* & *continuations* (like k). The types of values and continuations in the translated terms depend on CBN or CBV:

Definition 17 (CPS-translations of simple types)

We choose or we add a particular atomic type R, called the *response type*, then we define

CBN			CBV		
$\underline{a}$	$:=$	a	$\overline{a}$	$:=$	a
$\underline{A \to B}$	$:=$	$((\underline{A}\to R)\to R)\to(\underline{B}\to R)\to R$	$\overline{A \to B}$	$:=$	$\overline{A}\to(\overline{B}\to R)\to R$

※

Intuitively, a type A in the original calculus will give rise to a type $\underline{A}$ (resp. $\overline{A}$) of "A-values"; continuations are functions consuming those and returning something in the response type R (which is abstract in the sense that we will never need to know what it is), so continuations will therefore be of type $\underline{A}\to R$ (resp. $\overline{A}\to R$).

The encoding $\underline{M}$ (resp. $\overline{M}$) of a term M of type A will take the current continuation, of type $\underline{A}\to R$ (resp. $\overline{A}\to R$), and using that continuation, it will eventually output a response in the response type (as we have seen, the encoding starts with $\lambda k.\ldots$). It will therefore be of type $(\underline{A}\to R)\to R$ (resp. $(\overline{A}\to R)\to R$).

This is formalised as the following theorem:

Theorem 15 (CPS-translations preserve types)

If $\Gamma \vdash M : A$ then $(\underline{\Gamma}\to R)\to R \vdash \underline{M} : (\underline{A}\to R)\to R$ and $\overline{\Gamma} \vdash \overline{M} : (\overline{A}\to R)\to R$,
where $\overline{x_1 : A_1, \ldots, x_n : A_n}$ stands for $x_1 : \overline{A_1}, \ldots, x_n : \overline{A_n}$
and $((\underline{x_1 : A_1, \ldots, x_n : A_n})\to R)\to R$ stands for $x_1 : (\underline{A_1}\to R)\to R, \ldots, x_n : (\underline{A_n}\to R)\to R$. ※

Proof: Straightforward induction on M. □

Variants of CPS-translations exist, of which we mention two that are related to CBN and CBV:

Definition 18 (Variants)

- Fischer's translation for CBV [Fis72]

$\overline{x}$	$:=$	$\lambda k.k\ x$	$\overline{a}$	$:=$	a
$\overline{\lambda x.M}$	$:=$	$\lambda k.(k\ (\lambda k'.\lambda x.\overline{M}\ k'))$	$\overline{A \to B}$	$:=$	$(\overline{B}\to R) \to \overline{A}\to R$
$\overline{M\ N}$	$:=$	$\lambda k.\overline{M}\ (\lambda y.\overline{N}\ (\lambda z.y\ k\ z))$			

- Hofmann & Streicher's translation for CBN [HS97], using product types

$\underline{x}$	$:=$	$\lambda k.x\ k$	$\underline{a}$	$:=$	$a\to R$
$\underline{\lambda x.M}$	$:=$	$\lambda(x,k).\underline{M}\ k$	$\underline{A \to B}$	$:=$	$(\underline{A}\to R)\times\underline{B}$
$\underline{M\ N}$	$:=$	$\lambda k.\underline{M}\ (\underline{N},k)$			

※

Fischer's CBV-translation is very similar to Reynolds's: they only differ in the order in which arguments are passed in the encoding of λ-abstractions and applications (e.g. for the abstraction, Reynolds's translation binds x first, then binds the continuation variable k', whereas Fischer's binds k' first, then x). This is reflected in the encoding of the function type $A\to B$: the two arguments are swapped.

Hofmann & Streicher's CBN-translation differs more importantly from Plotkin's, as fewer "continuation wrappings" are introduced, reflected in the number of $\cdots\to R$ in the encoding

of types: that encoding works "negatively", as $\underline{\underline{A}}$ is directly the type of A-continuations which are not necessarily functions consuming an A-value and returning in the response type.

Again, these translations allow the same simulations as Plotkin's and Reynolds's, and of course preserve typing, with a slightly different formulation in the case of CBN:

THEOREM 16 (Hofmann & Streicher's CBN-translation preserves types)
If $\Gamma \vdash M : A$ then $\underline{\Gamma}\to R \vdash \underline{M} : \underline{A}\to R$
where $(\underline{\underline{x_1 : A_1, \ldots, x_n : A_n}})\to R$ stands for $x_1 : \underline{\underline{A_1}}\to R, \ldots, x_n : \underline{\underline{A_n}}\to R$. ※

Let us now look at CPS-translations with respect to denotational semantics: Remember that simply-typed λ-terms have a semantics in a Cartesian Closed Category. CPS-translations compile the simply-typed λ-calculus into itself (preserving types in the sense of Theorem 15), so we can now assign to a simply-typed λ-term M, the semantics (in a CCC) of $\underline{M}$ or $\overline{M}$ (so that semantics now depends on CBN/CBV). By the simulation theorem (Theorem 14), reductions are sound w.r.t. their corresponding semantics.

More interestingly, notice that we do not need the whole structure of a CCC to build those two semantics, as $\underline{M}$ or $\overline{M}$ live in a fragment of the simply-typed λ-calculus (the CPS-fragment), where in particular the types of $\underline{M}$ or $\overline{M}$ are functional types. More than this, every functional type that we ever need for that fragment is of the form $A\to R$.[11] Therefore, in order to build the categorical semantics of the CPS-fragment, we do not need as strong axioms as those of a CCC: on top of asking for cartesian products we only require the existence of exponentials objects of the form R^A. This is called a *response category*.

Now given a response category, the sub-category made of the objects of the form R^A is called a *continuation category*, a.k.a. *control category* (Selinger [Sel01]). Such a category turns out to have a rich structure that proves very useful for classical logic: not only it is a CCC (with exponential objects $(R^A)^{(R^B)}$ defined as $R^{A\times(R^B)}$) but objects of the form $R^{A\times B}$, denoted $R^A \parr R^B$, will play an important role.

1.7 Classical logic and CBN/CBV

We now relate classical logic to the above notions. We first review known translations from classical logic into intuitionistic logic: The intuition is that we can always turn P into P' by adding (enough) double negations, to get the property that

$$\text{If } \vdash_c P \text{ then } \vdash_i P'.$$

where $\vdash_c$ denotes classical provability and $\vdash_i$ denotes intuitionistic provability. Obviously, $\vdash_c P \leftrightarrow P'$, since the two formulae only differ by some double negations.

A potential question is then: If it suffices to add double negations in a classically provable formula to make it intuitionistically provable, are the two logics *really* different? Well, they differ at least in the sense that double negations break some of the nice properties of intuitionistic logic:

$$\text{If } \vdash_i A_1 \vee A_2 \text{ then either } \vdash_i A_1 \text{ or } \vdash_i A_2.$$
$$\text{If } \vdash_i \exists x A \text{ then there is } t \text{ such that } \vdash_i \{t/x\}\, A$$

[11] To be precise, we did use types such as $A_1\to\cdots\to A_n\to R$, but if we have products we can consider this to be the type $(A_1 \times \cdots \times A_n)\to R$.

Getting t from the proof of $\vdash_i \exists x A$ is called *witness extraction*. This can also be done in some theories, like (Heyting) arithmetics:

$$\text{If } HA \vdash_i \exists x A \text{ then there is } t \text{ such that } HA \vdash_i \{t/x\}\, A.$$

But in the most general case we cannot have the same properties when $\vdash_i \neg\neg(A_1 \vee A_2)$ or $\vdash_i \neg\neg\exists x A$. So what to do with a classical proof of $\vdash \exists x A$ is unclear. However, it is known that if A satisfies some specific property, a witness may be obtained from a classical proof of $\vdash \exists x A$; this is called classical witness extraction, and we will see this in Chapter 2.

The principle of inserting double negations gives rise to double negation translations (or $\neg\neg$-translations), of which we present two, remembering that $\neg A$ is $A \Rightarrow \bot$:

Definition 19 (Double negation translations)

$$\begin{array}{lcllcl} a^\bullet & := & a & a^\star & := & a \\ (A \Rightarrow B)^\bullet & := & ((A^\bullet \Rightarrow \bot) \Rightarrow \bot) \Rightarrow (B^\bullet \Rightarrow \bot) \Rightarrow \bot & (A \Rightarrow B)^\star & := & A^\star \Rightarrow (B^\star \Rightarrow \bot) \Rightarrow \bot \end{array}$$

※

We realise here that these translations, via the Curry-Howard correspondence, are exactly the translations of types from Definition 17 that make Plotkin's and Reynolds's translations "preserve types": The response type previously denoted R corresponds to the formula $\bot$, and a continuation is a proof of negation.

1.7.1 Identifying CBN and CBV in System L

The fact that double negation translations allow the construction of an intuitionistic proof of $A^\bullet$ (resp. $A^\star$) from a classical proof of A, suggests that we can adapt the CPS-translations of Definitions 16 and 18 to encode classical proof-terms, say of System L, into the simply-typed λ-calculus. If this encoding not only preserves types but also reductions (as in e.g. Theorem 14), then we could assign to a classical proof-term the categorical semantics of its CPS-encoding (which, as a simply-typed λ-term, is well-understood).

It remains to identify which reductions of System L will be reflected in the CPS-encoding.

Inspired by Theorem 14, we remark that the β-reductions that can be reflected by the CBV-encoding are of the form β_v, i.e. those reductions where every substitution that is computed substitute a variable by a value. In System L, we can impose similar restrictions: a CBV-reduction should only allow a substitution $\{t/x\}\, c$ to be computed if t is a "value"; and by symmetry, we could expect CBN-reduction to only allow a substitution $\{e/\alpha\}\, c$ to be computed if e is a "continuation value". But we still need to identify what the notions of values and continuation values are for System L. Considering the non-confluence situation $\langle \mu\alpha.c \mid \mu x.c' \rangle$ described in Section 1.4 (which causes so much difficulty for building semantics for System L), ruling out $\mu\alpha.c$ as value and ruling out $\mu x.c'$ as continuation value solves the problem: CBN-reduction would allow the reduction $\langle \mu\alpha.c \mid \mu x.c' \rangle \longrightarrow \{\mu\alpha.c/x\}\, c'$ and disallow $\langle \mu\alpha.c \mid \mu x.c' \rangle \longrightarrow \left\{\mu x.c'/\alpha\right\} c$, while CBV-reduction would allow the reduction $\langle \mu\alpha.c \mid \mu x.c' \rangle \longrightarrow \left\{\mu x.c'/\alpha\right\} c$ and disallow $\langle \mu\alpha.c \mid \mu x.c' \rangle \longrightarrow \{\mu\alpha.c/x\}\, c'$. In other words, CBV-reduction gives priority to the right while CBV-reduction gives priority to the left.

We can formalise this as the following definition:

DEFINITION 20 (CBN and CBV for System L -first attempt)

We identify the following notions of term values and continuation values:

$$\begin{array}{lrcl} \text{Term values} & V & ::= & x \,|\, \lambda x.t \,|\, (t_1, t_2) \,|\, \mathsf{inj}_i(t) \\ \text{Continuation values} & E & ::= & \alpha \,|\, t::e \,|\, (e_1, e_2) \,|\, \mathsf{inj}_i(e) \end{array}$$

The reduction relations $\longrightarrow_{\mathsf{CBN}}$ and $\longrightarrow_{\mathsf{CBV}}$ are defined as the contextual closures of the (groups of) rules in Fig. 9. ※

$$\begin{array}{lrcl} (\to) & \langle \lambda x.t_1 \mid t_2::e \rangle & \longrightarrow & \langle t_2 \mid \mu x.\langle t_1 \mid e \rangle \rangle \\ (\wedge) & \langle (t_1, t_2) \mid \mathsf{inj}_i(e) \rangle & \longrightarrow & \langle t_i \mid e \rangle \\ (\vee) & \langle \mathsf{inj}_i(t) \mid (e_1, e_2) \rangle & \longrightarrow & \langle t \mid e_i \rangle \end{array}$$

$$\begin{array}{llll} (\overleftarrow{\mu}_{\mathsf{N}}) & \langle \mu\beta.c \mid E \rangle \longrightarrow \left\{ {}^{E}\!/_{\beta} \right\} c & (\overleftarrow{\mu}) & \langle \mu\beta.c \mid e \rangle \longrightarrow \{ {}^{e}\!/_{\beta} \}\, c \\ (\overrightarrow{\mu}) & \langle t \mid \mu x.c \rangle \longrightarrow \{ {}^{t}\!/_{x} \}\, c & (\overrightarrow{\mu}_{\mathsf{V}}) & \langle V \mid \mu x.c \rangle \longrightarrow \left\{ {}^{V}\!/_{x} \right\} c \end{array}$$

CBN CBV

Figure 8: CBN and CBV reduction in System L (first attempt)

The fact that CBV-reduction keeps $(\overleftarrow{\mu})$ and restricts $(\overrightarrow{\mu})$ into $(\overrightarrow{\mu}_{\mathsf{V}})$ (and vice versa for CBN), is the formalisation of what we described before.

Doing this "works" in the sense that both CBN and CBV reductions are confluent systems (as higher-order orthogonal rewrite systems).

Unfortunately, these restrictions are not sufficient to build the denotational semantics of those systems according to methodology of CPS-translations, at least if we are to re-use the CPS-translations of types that we have seen in Section 1.6: Indeed, the simulation property that would be, for System L, the equivalent of Theorem 14, fails.

This was noticed in an erratum of [CH00], which also notices that the simulation does work on two specific fragments of System L: Concentrating on the implicational fragment,

- CBN-reduction can be simulated in the λ-calculus (according to Hofmann and Streicher's translation of types [HS97]) when every continuation of the form $t::e$ is such that t is a term value;
- CBV-reduction can be simulated in the λ-calculus (according to Reynold's or Fischer's translation of types [Rey72, Fis72]) when every continuation of the form $t::e$ is such that e is a continuation value.

Note that this makes sense because the former and the latter fragments are stable under CBN and CBV reduction, respectively.

This also suggests how to refine the notions of term values and continuation values as follows: instead of these notions concerning only the top-level construct of a term or a continuation, our new and more appropriate notions of values will be recursively defined.

This *strong* notion of value is taken primarily from [Wad03]:[12]

[12] Inspired by [MM09], we make a change about implication in the case of CBV, for which we *also* restrict continuation values, since this will make CBV normal forms correspond to proofs in LKQ [DJS95, DJS97], as we shall see in Chapter 3.

Definition 21 (CBN and CBV for System L)

In CBN, we identify the following notion of *continuation values*:

$$\text{CBN continuation values} \quad E \ ::= \ \alpha \,|\, t{::}E \,|\, (E_1, E_2) \,|\, \mathsf{inj}_i(E)$$

In CBV, we identify the following notion of *term values* and *continuation values*:

$$\begin{array}{lrcl} \text{CBV term values} & V & ::= & x \,|\, \lambda x.t \,|\, (V_1, V_2) \,|\, \mathsf{inj}_i(V) \\ \text{CBV continuation values} & F & ::= & \alpha \,|\, V{::}F \,|\, (F_1, F_2) \,|\, \mathsf{inj}_i(F) \,|\, \mu x.c \end{array}$$

The reduction relations $\longrightarrow_{\mathsf{CBN}}$ and $\longrightarrow_{\mathsf{CBV}}$ are the contextual closures of the rules in Fig. 9. ※

$(\overleftarrow{\mu}_{\mathsf{N}})$	$\langle \mu\beta.c \mid E\rangle$	$\longrightarrow \left\{{}^{E}\!/_{\beta}\right\} c$	$(\overleftarrow{\mu})$	$\langle \mu\beta.c \mid e\rangle$	$\longrightarrow \{{}^{e}\!/_{\beta}\}\, c$
$(\overrightarrow{\mu})$	$\langle t \mid \mu x.c\rangle$	$\longrightarrow \{{}^{t}\!/_{x}\}\, c$	$(\overrightarrow{\mu}_{\mathsf{V}})$	$\langle V \mid \mu x.c\rangle$	$\longrightarrow \left\{{}^{V}\!/_{x}\right\} c$
(ζ_{N})	$R[e]$	$\longrightarrow \langle \mu\alpha.R[\alpha] \mid e\rangle$	(ζ_{V})	$S[t]$	$\longrightarrow \langle t \mid \mu x.S[x]\rangle$
			(ζ_{V})	$T[e]$	$\longrightarrow T[\mu x.\langle x \mid e\rangle]$
$(\to_{\mathsf{N}})$	$\langle \lambda x.t_1 \mid t_2{::}E\rangle$	$\longrightarrow \langle \{{}^{t_2}\!/_{x}\}\, t_1 \mid E\rangle$	$(\to_{\mathsf{V}})$	$\langle \lambda x.t_1 \mid V{::}F\rangle$	$\longrightarrow \left\langle \left\{{}^{V}\!/_{x}\right\} t_1 \mid F\right\rangle$
$(\wedge_{\mathsf{N}})$	$\langle (t_1, t_2) \mid \mathsf{inj}_i(E)\rangle$	$\longrightarrow \langle t_i \mid E\rangle$	$(\wedge_{\mathsf{V}})$	$\langle (V_1, V_2) \mid \mathsf{inj}_i(F)\rangle$	$\longrightarrow \langle V_i \mid F\rangle$
$(\vee_{\mathsf{N}})$	$\langle \mathsf{inj}_i(t) \mid (E_1, E_2)\rangle$	$\longrightarrow \langle t \mid E_i\rangle$	$(\vee_{\mathsf{V}})$	$\langle \mathsf{inj}_i(V) \mid (F_1, F_2)\rangle$	$\longrightarrow \langle V \mid F_i\rangle$
	CBN			CBV	

where R, S, and T range over contexts of the following grammar:

$$\begin{array}{lrcl} \text{CBN continuation contexts} & R & ::= & \langle t \mid t'{::}[\,]\rangle \ \big|\ \langle t \mid ([\,], e)\rangle \ \big|\ \langle t \mid (E, [\,])\rangle \ \big|\ \langle t \mid \mathsf{inj}_i([\,])\rangle \\ \text{CBV term contexts} & S & ::= & \langle V \mid [\,]{::}e\rangle \ \big|\ \langle ([\,], t) \mid e\rangle \ \big|\ \langle (V, [\,]) \mid e\rangle \ \big|\ \langle \mathsf{inj}_i([\,]) \mid e\rangle \\ \text{CBV continuation contexts} & T & ::= & \langle V \mid V'{::}[\,]\rangle \mid \langle V \mid ([\,], e)\rangle \mid \langle V \mid (F, [\,])\rangle \mid \langle V \mid \mathsf{inj}_i([\,])\rangle \end{array}$$

the (ζ_{N}) only applies under the condition that e is not a (CBN-) continuation value,
the (ζ_{V})-rules only apply under the condition that t is not a (CBV-) term value and e is not a (CBV-) continuation value.

Figure 9: CBN and CBV reduction in System L

In this version, we kept the CBV-rules $(\overleftarrow{\mu})$ and $(\overrightarrow{\mu}_{\mathsf{V}})$, and the CBN-rules $(\overleftarrow{\mu}_{\mathsf{N}})$ and $(\overrightarrow{\mu})$.

The (ζ_{V})-rules (resp. the (ζ_{N})-rule) are new: they were introduced in a slightly more general version in [Wad03], while the version we take here more closely follows [MM09]. These rules are due to our strong restriction on term values (resp. continuation values): the fact that a term is a value is not just the fact that it is not of the form $\mu\alpha.c$, as term values are recursively defined. Therefore if a term t is not of the form $\mu\alpha.c$ but one of its (say direct) subterms is, then t is not a value and there is no CBV-rule to reduce $\langle t \mid \mu x.c\rangle$. Progress then fails if we do not add the (ζ_{V})-rules to pull the first subterm of t that is not a value to the top-level.

These ζ rules also impact rules $(\to)$, $(\wedge)$, and $(\vee)$, which now have to be restricted in order to preserve confluence: for instance in CBV, the fact that $(\mu\alpha.c, t)$ is not a term value means that, when facing a continuation e, $\mu\alpha.c$ will be extracted from the pair and will have the control of computation

$$\langle (\mu\alpha.c, t) \mid e\rangle \longrightarrow_{\zeta_{\mathsf{V}}} \langle \mu\alpha.c \mid \mu x.\langle (x, t) \mid e\rangle\rangle \longrightarrow_{\overleftarrow{\mu}} \left\{{}^{\mu x.\langle (x,t)\mid e\rangle}\!/_{\alpha}\right\} c$$

and therefore it is clear that, should e be of the form $\mathsf{inj}_2(e')$, the original application of rule $(\wedge)$ would have a totally different semantics:

$$\langle (\mu\alpha.c, t) \mid e \rangle \longrightarrow_{\wedge} \langle t \mid e' \rangle$$

Hence the restriction of $(\to)$, $(\wedge)$, and $(\vee)$ to $(\to_{\mathsf{N}})$, $(\wedge_{\mathsf{N}})$, and $(\vee_{\mathsf{N}})$ in the CBN case, and to $(\to_{\mathsf{V}})$, $(\wedge_{\mathsf{V}})$, and $(\vee_{\mathsf{V}})$ in the CBV case.

Note that in CBN, we decided to make $(\to_{\mathsf{N}})$ collapse the two reduction steps

$$\langle \lambda x.t_1 \mid t_2 :: E \rangle \longrightarrow_{\to} \langle t_2 \mid \mu x.\langle t_1 \mid E \rangle \rangle \longrightarrow_{\vec{\mu}} \langle \{t_2/x\}\, t_1 \mid E \rangle$$

into one step, because $(\vec{\mu})$ has priority anyway.[13] The rule $(\to_{\mathsf{V}})$ is designed by symmetry, collapsing the two steps

$$\langle \lambda x.t_1 \mid V :: F \rangle \longrightarrow_{\to} \langle V \mid \mu x.\langle t_1 \mid F \rangle \rangle \longrightarrow_{\vec{\mu}} \left\langle \{V/x\}\, t_1 \mid F \right\rangle$$

and noticing that if t_2 is not a term value, then the application of the original rule $(\to)$ is recovered as follows:

$$\langle \lambda x.t_1 \mid t_2 :: F \rangle \longrightarrow_{\zeta_{\mathsf{V}}} \langle t_2 \mid \mu y.\langle \lambda x.t_1 \mid y :: F \rangle \rangle \longrightarrow_{\to_{\mathsf{V}}} \langle t_2 \mid \mu x.\langle t_1 \mid F \rangle \rangle$$

Of course the extra rules satisfy Subject Reduction, so that we have:

THEOREM 17 (Subject reduction for System L: CBN & CBV)
If $c : (\Gamma \vdash \Delta)$ and either $c \longrightarrow_{\mathsf{CBN}} c'$ or $c \longrightarrow_{\mathsf{CBV}} c'$ then $c' : (\Gamma \vdash \Delta)$.
And similarly for terms and continuations. ※

Proof: Straightforward induction on the rewrite derivation. □

THEOREM 18 (Confluence) $\longrightarrow_{\mathsf{CBN}}$ and $\longrightarrow_{\mathsf{CBV}}$ are confluent. ※

Proof: They are orthogonal higher-order rewrite systems.[14] □

Finally, one could be puzzled by what seems like an asymmetry between CBN and CBV, the latter having more rules and requiring a notion of continuation value while the former does not need a notion of term value. This asymmetry is not due to CBN vs. CBV, but is due to the implication: its main continuation construct $t :: e$ has a term as a direct sub-term, while no term contruct has a continuation as a direct sub-term. It would be the case if we considered the De Morgan dual of implication, namely *subtraction* (see for instance [Cro04]), which would make CBN completely symmetric to CBV.

1.7.2 Two stable fragments

Now it is easy to connect the $\longrightarrow_{\mathsf{CBN}}$ and $\longrightarrow_{\mathsf{CBV}}$ reduction relations of Definition 21 to those of our first attempt in Definition 20, if we concentrate on the two fragments:[15]

DEFINITION 22 (LK^{N} and LK^{V}) Let LK^{N} and LK^{V} be the fragments of System L consisting of $\longrightarrow_{\zeta_{\mathsf{N}}}$ -normal forms and $\longrightarrow_{\zeta_{\mathsf{V}}}$ -normal forms, respectively. ※

[13] We shall see that it makes the $\mu x.\_$ construct superfluous (in the sense that the fragment without this construct is logically complete, and stable under reduction).

[14] The rewrite system presented in Fig. 9 is a standard (higher-order) rewrite system: we did use a non-standard formulation for the ζ rules based on a grammar for continuation contexts and term contexts as well as on side-conditions ("t is not a term value and e is not a continuation value"), but we could equally have formulated all the cases as standard (but numerous!) rewrite rules.

[15] Namely, those fragments where the reductions of Definition 20 are actually simulated by (the adaptation to System L of) the CPS-translations.

Remark 19

1. Concerning implication only, LK^{N} is the fragment where every continuation of the form $t::e$ is such that e is a continuation value.
2. Concerning implication only, LK^{V} is the fragment where every continuation of the form $t::e$ is such that t is a term value;
3. Also, $\longrightarrow_{\zeta_{\mathsf{N}}}$ and $\longrightarrow_{\zeta_{\mathsf{V}}}$ are terminating reduction relations, so it is easy to normalise a command into one of these fragments, using cuts.
4. Moreover, LK^{N} and LK^{V} are respectively stable under $\longrightarrow_{\mathsf{CBN}}$ and $\longrightarrow_{\mathsf{CBV}}$, so the cuts can be reduced while staying in the fragments.
5. Furthermore, in LK^{N} and LK^{V}, $\longrightarrow_{\mathsf{CBN}}$ and $\longrightarrow_{\mathsf{CBV}}$ of Definition 21 respectively coincide with those of Definition 20.[16]
6. Notice that the encoding of λ-calculus in System L in Definition 14, actually only reaches the fragment LK^{N}, and the simulation lemma (Lemma 9) only involves CBN-reduction.

※

1.7.3 Denotational semantics of CBN and CBV

As anticipated, it is now possible to define CPS-translations of terms, continuations, and commands, respectively denoted $\underline{t}$, $\underline{e}$, $\underline{c}$ for CBN, and $\overline{t}$, $\overline{e}$, $\overline{c}$ for CBV, in a way that preserves reductions:

Theorem 20 (Preservation of reduction)

CBN: If $c_1 \longrightarrow_{\mathsf{CBN}} c_2$ then $\underline{c_1} \longrightarrow_{\beta}^{*} \underline{c_2}$

CBV: If $c_1 \longrightarrow_{\mathsf{CBV}} c_2$ then $\overline{c_1} \longrightarrow_{\beta}^{*} \overline{c_2}$

※

We do not give the details here, which are just technical, but they can be found in e.g. [Wad03].

And these encodings also preserve typing, if Hofmann and Streicher's encoding of types for CBN, and Fischer's encoding of types for CBV, are considered not just for $\to$ (i.e. $\Rightarrow$) but also $\times$ (i.e. $\wedge$) and $+$ (i.e. $\vee$):

Theorem 21 (Preservation of typing)

Assume

$$\Gamma \vdash t : A \;;\; \Delta$$
$$\Gamma \;;\; e : A \vdash \Delta$$
$$c : (\Gamma \vdash \Delta)$$

[16] Almost, since in Definition 21 the rule $(\to_{\mathsf{N}})$ (resp. $(\to_{\mathsf{V}})$) collapses the two steps $\longrightarrow_{\to} \cdot \longrightarrow_{\mu}^{\to}$ (resp. $\longrightarrow_{\to} \cdot \longrightarrow_{\mu_{\mathsf{V}}}^{\to}$) of $\longrightarrow_{\mathsf{CBN}}$ (resp. $\longrightarrow_{\mathsf{CBV}}$) from Definition 20: but in LK^{N} (resp. LK^{V}), there is no other choice than $\longrightarrow_{\mu}^{\to}$ (resp. $\longrightarrow_{\mu_{\mathsf{V}}}^{\to}$) for a top-level reduction that can follow $\longrightarrow_{\to}$ in CBN-reduction (resp. CBV-reduction).

Then

$$\underline{\underline{\Gamma}}\to R, \underline{\underline{\Delta}} \vdash \underline{t}:\underline{\underline{A}}\to R \qquad \overline{\Gamma}, \overline{\Delta}\to R \vdash \bar{t}:(\overline{A}\to R)\to R$$
$$\underline{\underline{\Gamma}}\to R, \underline{\underline{\Delta}} \vdash \underline{e}:(\underline{\underline{A}}\to R)\to R \qquad \overline{\Gamma}, \overline{\Delta}\to R \vdash \bar{e}:\overline{A}\to R$$
$$\underline{\underline{\Gamma}}\to R, \underline{\underline{\Delta}} \vdash \underline{c}:R \qquad \overline{\Gamma}, \overline{\Delta}\to R \vdash \bar{c}:R$$

Using Hofmann & Streicher's translation of types [HS97]

Using Fischer's translation of types [Fis72]

※

As already mentioned, we can now use these CPS-translations to define categorical semantics for classical proofs:

DEFINITION 23 (Semantics of System L in a response category)

Assume $c:(x_1:A_1,\ldots,x_n:A_n \vdash \alpha_1:B_1,\ldots,\alpha_m:B_m)$.

Define the semantics $[\![c]\!]^r_{\mathsf{N}} := [\![\underline{c}]\!]$ and $[\![c]\!]^r_{\mathsf{V}} := [\![\bar{c}]\!]$, where $[\![t]\!]$ is the semantics, in a response category, of a λ-term t in the CPS-fragment, as defined by the rules of Fig. 3.

Writing K_A for the object corresponding to $\underline{\underline{A}}$, and C_A for R^{K_A}, we have

$$[\![c]\!]^r_{\mathsf{N}} : (C_{A_1} \times \ldots \times C_{A_n} \times K_{B_1} \times \ldots \times K_{B_m}) \longrightarrow R$$

Writing V_A for the object corresponding to $\overline{A}$, K_A and for R^{V_A}, we have

$$[\![c]\!]^r_{\mathsf{V}} : (V_{A_1} \times \ldots \times V_{A_n} \times K_{B_1} \times \ldots \times K_{B_m}) \longrightarrow R$$

※

Now, remember that a *control category* is the sub-category of a response category $\mathcal{C}$ whose objects are in $\{R^A | A \in \mathcal{C}\}$, and that $R^A \parr R^B$ denotes $R^{A\times B}$.

DEFINITION 24 (Semantics of System L in control and co-control categories)

Assume $c:(x_1:A_1,\ldots,x_n:A_n \vdash \alpha_1:B_1,\ldots,\alpha_m:B_m)$.

CBN The semantics $[\![c]\!]^r_{\mathsf{N}}$ in a response category gives rise, by curryfication, to a morphism

$$[\![c]\!]_{\mathsf{N}} : C_{A_1} \times \ldots \times C_{A_n} \longrightarrow C_{B_1} \parr \ldots \parr C_{B_m}$$

in a control category.

CBV The semantics $[\![c]\!]^r_{\mathsf{V}}$ in a response category gives rise, by curryfication, to a morphism

$$[\![c]\!]_{\mathsf{V}} : K_{B_1} \times \ldots \times K_{B_m} \longrightarrow K_{A_1} \parr \ldots \parr K_{A_n}$$

in a control category.

As the arrow looks "reversed", from the original typing of c, it is more natural to interpret c as the corresponding morphism

$$[\![c]\!]_{\mathsf{V}} : K_{A_1} \otimes \ldots \otimes K_{A_n} \longrightarrow K_{B_1} + \ldots + K_{B_m}$$

in a *co-control category*, the dual of a control category where $\otimes$ denotes the dual of $\parr$ (and the co-product $+$ denotes the dual of the product $\times$).

※

This formalises the idea that CBN-reduction corresponds to a denotational semantics in control categories, while CBV-reduction corresponds to a denotational semantics in co-control categories.

Indeed, from Theorem 20 we get that the semantics validate the reductions:

THEOREM 22 (Soundness of CBN and CBV in control and co-control categories)
If $c \longrightarrow_{\mathsf{CBN}} c'$ then $[\![c]\!]_{\mathsf{N}} = [\![c']\!]_{\mathsf{N}}$
If $c \longrightarrow_{\mathsf{CBV}} c'$ then $[\![c]\!]_{\mathsf{V}} = [\![c']\!]_{\mathsf{V}}$ ※

And we did this by breaking the symmetry between $\wedge$ and $\vee$: Indeed in a control category, $\parr$ is not the dual of $\times$ (equivalently in a co-control category, $+$ is not the dual of $\otimes$).

Conclusion

The work on control and co-control categories is due to Selinger [Sel01] for Parigot's $\lambda\mu$-calculus, showing a duality between CBN and CBV in the categorical sense of duality, and even before a syntactic duality between CBN and CBV was displayed by System L and its variants. It follows preliminary works by Hofmann, Streicher, Reus [HS97, SR98], etc, on the semantics of continuations (where the question of duality between CBV and CBN -in $\lambda\mu$- is conjectured).

In conclusion, many variants of classical proof calculi have been studied; in particular,

- the De Morgan dual of implication in classical logic, namely *subtraction*, can also be given a computational interpretation, as shown for instance by [Cro04];
- variants of Parigot's $\lambda\mu$ have different properties with respect to observational equivalence, separation and η-conversion, etc... [Sau05, Sau08, HZ09, Sau10c, Sau10b, Sau12];
- *control delimiters* can be used to limit the scope of the context that can be captured by a term via control operators, allowing for instance the capture of the *shift* and *reset* operators of [DF89, DF90]; these give rise to *delimited continuations* and can be given a proof-theoretic interpretation [AHS09, HG08, Sau10a];
- other reduction strategies than CBV and CBN have been investigated under the light of the duality of computation, such as Call-by-Need [AF97, AHS11, ADH+12].

Chapter 2

Orthogonality: models for normalisation and witness extraction

Contents

In this chapter we present the concept of *orthogonality* and two applications of it that are useful for classical proof-term calculi: strong normalisation and witness extraction.

Orthogonality was used by Girard in the context of *linear logic* [Gir87] to prove normalisation of cut-elimination and it lies at the heart of its proof semantics based on *coherent spaces*.

The concept of orthogonality has also proved a key concept in the proof theory of classical logic, as it features, just like linear logic does, a duality that is most immediately seen in the form of De Morgan's laws. Indeed, orthogonality is the basis of *classical realisability* [DK00, Kri01], which can be seen as a semantical approach to the Curry-Howard correspondence for classical logic. It also provided a new tool for models of classical proofs, and for proving properties of programs [Par97, MV05, LM08], most notoriously the strong normalisation property. Furthermore, orthogonality shed a new light on the theory of *polarisation* and *focussing* for classical logic, as revealed for instance in [MM09] and explored in Chapter 3.

In this chapter we start by illustrating how proofs of strong normalisation relate to orthogonality. Summarising [BL11], Section 2.1 rephrases and modularises, with the notion of *orthogonality models*, the well-known techniques by Tait [Tai67, Tai75], Reynolds [Rey72] and Girard [Gir72] for proving the strong normalisation of the simply-typed λ-calculus and

System F. Section 2.2 shows how such models allow the adaptation, to classical proof-term calculi, of strong normalisation proofs, both in the case of a confluent calculus and non-confluent calculus. Section 2.3 then shows how orthogonality models can be used for classical witness extraction, using a technique due to Miquel [Miq09, Miq11].

2.1 Revisiting Proofs of Strong Normalisation for System F

This section presents concepts and a methodology developed in [BL11]: in particular, we approach the strong normalisation of System F with the notion of *orthogonality model*, adapting the Tait-Girard methodology [Tai75, Gir72]. Although System F is an intuitionistic system, this approach will form the starting point from which the adaptation to classical logic will be explored.

DEFINITION 25 (System F)

The types of System F are given by the following grammar:

$$A, B, \ldots ::= \alpha \mid A \to B \mid \forall \alpha A$$

where α ranges over a denumerable set of elements called *type variables*, and the construct $\forall \alpha A$ binds α in A.

Typing contexts are defined as in Definition 6, using System F types instead of simple types; they will be denoted Γ, Δ, etc.

The *free type variables* of a type A (resp. a typing context Γ) will be denoted $ftv(A)$ (resp. $ftv(\Gamma)$).

The typing system of System F is given in Fig. 10. Derivability of a sequent in System F is denoted $\Gamma \vdash_{\mathsf{F}} M : A$. ※

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$$

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M : \forall \alpha A}\ \alpha \notin ftv(\Gamma) \qquad \frac{\Gamma \vdash M : \forall \alpha A}{\Gamma \vdash M : \left\{ {}^{B}/_{\alpha} \right\} A}$$

Figure 10: System F

The method to prove strong normalisation is to build a model with

- an interpretation for terms as elements of a set $\mathcal{E}$
- an interpretation for types as (interesting) subsets of $\mathcal{E}$
- such that
  - the interpretation of a term of type A is in the interpretation of A
  - if the interpretation of a term is in there, the term is strongly normalising.

2.1.1 Orthogonality models and the Adequacy Lemma

Definition 26 (Orthogonality models)

An *orthogonality model* is a 4-tuple $(\mathcal{D}, \bot, \mathcal{E}, [\![\_]\!]\_)$ where

- $\mathcal{D}$ is a set of elements called *values*;
- $\bot$ is a relation between values and lists of values called the *orthogonality relation*;
- $\mathcal{E}$ is a superset of $\mathcal{D}$;
- $[\![\_]\!]\_$ is a function mapping every λ-term M (typed or untyped) to an element $[\![M]\!]_\rho$ of $\mathcal{E}$, where ρ is a parameter called *semantic context* mapping term variables to values.
- the following axioms are satisfied:

  (A1) For all ρ, $\vec{v}$, x, if $\rho(x) \bot \vec{v}$ then $[\![x]\!]_\rho \bot \vec{v}$.

  (A2) For all ρ, $\vec{v}$, M_1, M_2, if $[\![M_1]\!]_\rho \bot ([\![M_2]\!]_\rho :: \vec{v})$ then $[\![M_1\ M_2]\!]_\rho \bot \vec{v}$.

  (A3) For all ρ, $\vec{v}$, x, M and for all values u, if $[\![M]\!]_{\rho, x\mapsto u} \bot \vec{v}$ then $[\![\lambda x.M]\!]_\rho \bot (u :: \vec{v})$.

※

In fact, $\mathcal{D}$ and $\bot$ are already sufficient to interpret any System F type A as a set $[\![A]\!]^+$ of values (see Definition 28 below): if types are seen as logical formulae, we can see this construction as a way of building some of their realisability / set-theoretical models.

There is no notion of computation pertaining to values, but the interplay between the interpretation of terms and the orthogonality relation is imposed by the axioms so that the Adequacy Lemma (which relates typing to semantics) holds:

$$\text{If } \vdash_{\mathsf{F}} M : A \text{ then } [\![M]\!] \in [\![A]\!]^+$$

We now assume that we are given an orthogonality model $(\mathcal{D}, \bot, \mathcal{E}, [\![\_]\!]\_)$.

Notation 27 By $\mathcal{D}^*$ we denote the set of lists of values.

If $X \subseteq \mathcal{D}$ and $Y \subseteq \mathcal{D}^*$ let

$$\begin{array}{lcl} X :: Y & := & \{u :: \vec{v} \mid u \in X, \vec{v} \in Y\} \\ X^\bot & := & \{\vec{v} \in \mathcal{D}^* \mid \forall u \in X, u \bot \vec{v}\} \\ Y^\bot & := & \{u \in \mathcal{D} \mid \forall \vec{v} \in Y, u \bot \vec{v}\} \end{array}$$

※

Remark 23 The usual properties of orthogonality hold:

$X \subseteq X^{\bot\bot}$, $X^{\bot\bot\bot} = X^\bot$, and if $X \subseteq X'$ then $X'^\bot \subseteq X^\bot$ ※

We now define the interpretation of types. The intuition is the same as that of Krivine's classical realisability [DK00, Kri01]:

- we first interpret a formula A as a set of "counter-proofs", with the basic constructs that we expect to use in order to refute the formula: for instance the basic way to refute $A_1 \to A_2$ is to provide a "proof" of A_1 and a "counter-proof" of A_2; similarly, the basic way to refute $\forall \alpha A_1$ is to find a suitable interpretation of α and produce a "counter-proof" of A_1 under this interpretation; the set of counter-proofs for atomic formulae is then naturally given by a *valuation*;
- we then define the "proofs" ("realisers" would be a better term) of a formula A as any value that is able to "face all counter-proofs", this latter concept being what the orthogonality relation is precisely there to specify.

DEFINITION 28 (Interpretation of types)

A *valuation* is a function, denoted σ, σ',..., from type variables to subsets of $\mathcal{D}^*$.

Two interpretation of types are defined by simultaneous induction of types, a *positive interpretation* and a *negative interpretation*:

$$\llbracket A \rrbracket^+_\sigma := [A]^{-\perp}_\sigma \qquad \begin{array}{lll} [\alpha]^-_\sigma & := & \sigma(\alpha) \\ [A{\to}B]^-_\sigma & := & \llbracket A \rrbracket^+_\sigma :: [B]^-_\sigma \\ [\forall\alpha A]^-_\sigma & := & \bigcup_{Y\subseteq\mathcal{D}^*} [A]^-_{\sigma,\alpha\mapsto Y} \end{array}$$

We then define the interpretation of typing contexts:

$$\llbracket \Gamma \rrbracket_\sigma := \{\rho \mid \forall (x\!:\!A) \in \Gamma, \rho(x) \in \llbracket A \rrbracket^+_\sigma\}$$

※

This approach begs the question: why is it the case that counter-proofs are defined more primitively than proofs? As described for instance in [MM09] (and in the rest of this thesis), this is simply a coincidence about the two type constructs we use in System F: they both have a "negative polarity", if we see them in the more general context of *polarised logic*, as we shall discuss in Chapter 3. Second-order quantification is often used in the field of realisability (and elsewhere) to encode other logical connectives, which made the negative approach prevalent in that field, with counter-proofs being defined first and proofs being defined by orthogonality. With primitive connectives that have a "positive polarity", such as intuitionistic disjunction, proofs would be defined first and counter-proofs would be defined by orthogonality. We shall come back to that discussion in the next chapters.

REMARK 24 We have the usual properties of substitutions:

$$\left[\left\{{}^{B}\!/_{\alpha}\right\} A\right]^-_\sigma = [A]^-_{\sigma,\alpha\mapsto[B]^-_\sigma} \quad \text{and} \quad \left[\!\left[\left\{{}^{B}\!/_{\alpha}\right\} A\right]\!\right]^+_\sigma = \llbracket A \rrbracket^+_{\sigma,\alpha\mapsto[B]^-_\sigma}$$

Also notice that the *for all* quantifier is interpreted as an intersection:

$$\llbracket \forall\alpha A \rrbracket^+_\sigma = \bigcap_{Y\subseteq\mathcal{D}^*} \llbracket A \rrbracket^+_{\sigma,\alpha\mapsto Y}$$

※

With these definitions we can prove the Adequacy Lemma:

LEMMA 25 (Adequacy Lemma)

If $\Gamma \vdash_{\mathsf{F}} M\!:\!A$, then for all valuations σ and for all mappings $\rho \in \llbracket \Gamma \rrbracket_\sigma$ we have $\llbracket M \rrbracket_\rho \in \llbracket A \rrbracket^+_\sigma$.

※

Proof: By induction on the derivation of $\Gamma \vdash M\!:\!A$, using axioms (A1), (A2) and (A3). See [BL11]. □

2.1.2 Applicative orthogonality models and Strong Normalisation

DEFINITION 29 (Applicative orthogonality model)

An *applicative orthogonality model* is a 4-tuple $(\mathcal{D}, \mathcal{E}, @, \llbracket \_ \rrbracket\_)$ where:

- $\mathcal{D}$ is a set, $\mathcal{E}$ is a superset of $\mathcal{D}$, $@$ is a (total) function from $\mathcal{E}\times\mathcal{E}$ to $\mathcal{E}$, and $\llbracket \_ \rrbracket\_$ is a function (parameterised by a semantic context) from λ-terms to $\mathcal{E}$.
- $(\mathcal{E}, \mathcal{D}, \perp, \llbracket \_ \rrbracket\_)$ is an orthogonality model,
  where the relation $u \perp \vec{v}$ is defined as $(u@\vec{v}) \in \mathcal{D}$
  (writing $u@\vec{v}$ for $(\ldots(u@v_1)@\ldots@v_n)$ if $\vec{v} = v_1 :: \ldots v_n :: [\,]$).

※

Remark 26 Axioms (A1) and (A2) are ensured provided that $[\![M\ N]\!]_\rho = [\![M]\!]_\rho @ [\![N]\!]_\rho$ and $[\![x]\!]_\rho = \rho(x)$. These conditions can hold by definition (as in the following example), or can be proved. ※

We now give an applicative orthogonality model to conclude strong normalisation of System F; this will capture, in essence, the Tait-Girard proof methodology [Tai75, Gir72]. The model is here a *term model*, in that $\mathcal{E}$ is the set of all λ-terms and a λ-term is interpreted as itself.

Example 6 (A term-model for Strong Normalisation)

Let $\mathcal{D}$ be the set of strongly-normalising λ-terms, and let $\mathcal{E}$ be set of all λ-terms. We define $u \perp \vec{v}$ as $(u@\vec{v}) \in \mathsf{SN}$, and the interpretation of terms as follows:

$$\begin{array}{lcl} [\![x]\!]_\rho & := & \rho(x) \\ [\![M_1\ M_2]\!]_\rho & := & [\![M_1]\!]_\rho\ [\![M_2]\!]_\rho \\ [\![\lambda x.M]\!]_\rho & := & \lambda x.[\![M]\!]_{\rho, x\mapsto x} \end{array}$$

Requirement 3 is a consequence of anti-reduction:
If $\left\{{}^{P}\!/_{x}\right\} M\ \vec{N} \in \mathsf{SN}$ and $P \in \mathsf{SN}$ then $(\lambda x.M)\ P\ \vec{N} \in \mathsf{SN}$.
Note that for all $\vec{N} \in \mathsf{SN}^*$ and all term variables x, $x \perp \vec{N}$.
Hence, for all valuations σ and all types A, $x \in [\![A]\!]^+_\sigma$.
We apply the Adequacy Lemma (Lemma 25):
If $\Gamma \vdash M : A$, then for all valuation σ and all mapping $\rho \in [\![\Gamma]\!]_\sigma$ we have $[\![M]\!]_\rho \in \mathsf{SN}$.
Hence, $M \in \mathsf{SN}$. ※

In summary, we have defined a family of models for the (polymorphically) typed λ-calculus, and presented one instance with which strong normalisation could be inferred. In [BL11] we presented other instances of orthogonality models, based for instance on intersection types. Unlike usual models (e.g. CCC), orthogonality models do not necessarily equate terms up to β-reduction (if $M \longrightarrow_\beta N$, we do not necessarily have $[\![M]\!] = [\![N]\!]$). This allows us to build a model where $[\![M]\!] = M$, from which we can infer strong normalisation of typed terms (an instance of CCC would be useless for this).

2.2 Adapting the approach to classical calculi

Orthogonality was used by Parigot to prove strong normalisation of CBN $\lambda\mu$-calculus [Par97]. For their non-confluent calculus, Barbanera & Berardi [BB96] adapted the Tait-Girard reducibility technique with "symmetric reducibility candidates". The key idea in both cases is still that a type A is interpreted as a pair of two orthogonal sets:[1]

- a set $[\![A]\!]^+$ of proof(-terms)
- a set $[\![A]\!]^-$ of counter-proof(-terms)

...satisfying some saturation property (like reducibility candidates do).

In the proof of strong normalisation of System F that we presented in the previous section, the notion of saturation that holds for $[\![A]\!]^+$ is that it is closed under bi-orthogonal ($[\![A]\!]^{+\perp\perp} = [\![A]\!]^+$). In particular, in an applicative term model, the fact that $[\![A]\!]^+$ is closed under bi-orthogonal allows to derive, from axiom (A3) of the orthogonality relation, the property that

[1] Two sets $\mathcal{U}$ and $\mathcal{V}$ are *orthogonal* if $\forall t \in \mathcal{U}, \forall u \in \mathcal{V}, t \perp u$.

if $(\{N/x\}\, M)\ N_1 \dots N_n \in [\![A]\!]^+$ and $N, N_1, \dots, N_n \in \mathcal{D}$, then $(\lambda x.M)N\ N_1 \dots N_n \in [\![A]\!]^+$. Although not explicitly used in our proof of strong normalisation (Example 6), this property lies in the background and is often explicitly used in more traditional presentations of the reducibility technique [Tai75, Gir72]. In brief, the technique works because the interpretation of a type is closed under "head-anti-reduction".

This is also the approach for classical proof-term calculi, in particular for a confluent calculus such as the Parigot's $\lambda\mu$ [Par97].

2.2.1 The case of a confluent calculus

In this section we take the example of the LK^{N} fragment of System L (Definition 22), with $\Rightarrow$ as the only connective, and considering the reduction relation CBN. We can also prove strong normalisation by building a term model based on orthogonality:

Three rewrite rules apply:

$$\begin{array}{lccl} (\to) & \langle \lambda x.t_1 \mid t_2 :: E \rangle & \longrightarrow & \langle \{t_2/x\}\, t_1 \mid E \rangle \\ (\overleftarrow{\mu}_{\mathsf{N}}) & \langle \mu\beta.c \mid E \rangle & \longrightarrow & \{E/\beta\}\, c \\ (\overrightarrow{\mu}) & \langle t \mid \mu x.c \rangle & \longrightarrow & \{t/x\}\, c \end{array}$$

Therefore we can adapt the axiom (A3) of Definition 26 as follows:

DEFINITION 30 (Orthogonality model for System LK^{N})

An *orthogonality model* for System LK^{N} is given by $(\mathcal{D}_t, \mathcal{D}_e, \bot)$ where $\mathcal{D}_t$ is a set of terms, $\mathcal{D}_e$ is a set of continuations, and $\bot$ is a relation between $\mathcal{D}_t$ and $\mathcal{D}_e$, which can be seen as a set of commands, and satisfying the following *saturation requirements*:

If $\langle t_1 \mid e \rangle\, (\rho, x \mapsto t_2) \in \bot$ and $t_2 \in \mathcal{D}_t$ then $\langle \lambda x.t_1 \mid t_2 :: e \rangle\, \rho \in \bot$
If $c(\rho, \beta \mapsto E) \in \bot$ and $E \in \mathcal{D}_e$ then $\langle \mu\beta.c \mid E \rangle\, \rho \in \bot$
If $c(\rho, x \mapsto t) \in \bot$ and $t \in \mathcal{D}_t$ then $\langle t \mid \mu x.c \rangle\, \rho \in \bot$

where ρ is a *semantic context*, i.e. a substitution mapping term variables to terms in $\mathcal{D}_t$ and continuation variables to continuations in $\mathcal{D}_e$, and $c\rho$ denotes the capture-avoiding application, to c, of the substitution ρ. ※

DEFINITION 31 (Interpretation of types for System LK^{N})

A *valuation* is a function, denoted σ, σ',..., from type variables to subsets of $\mathcal{D}_e$ that only contain value continuations.

Two interpretation of types are defined by simultaneous induction of types, a *positive interpretation* and a *negative interpretation*:

$$\begin{array}{rcl} [\alpha]^-_\sigma & := & \sigma(\alpha) \\ [A{\to}B]^-_\sigma & := & [\![A]\!]^+_\sigma :: [\![B]\!]^-_\sigma \end{array}$$

$$[\![A]\!]^+_\sigma := [A]^{-\bot}_\sigma \qquad [\![A]\!]^-_\sigma := [A]^{-\bot\bot}_\sigma$$

where $X :: Y$ denotes $\{t :: E \mid t \in X, E \in Y\}$ for any $X \subseteq \mathcal{D}_t$ and $Y \subseteq \mathcal{D}_e$.

We then define the interpretation of a typing context (i.e. a pair of a typing context for term variables and a typing context for continuation variables):

$$[\![\Gamma, \Delta]\!]_\sigma := \{\rho \mid \forall (x : A) \in \Gamma, \rho(x) \in [\![A]\!]^+_\sigma, \text{ and } \forall (\alpha : A) \in \Delta, \rho(\alpha) \in [\![A]\!]^-_\sigma\}$$

※

Lemma 27 (Adequacy Lemma for System LK^{N})

1. If $\Gamma \vdash_{\mathsf{L}} t : A \;;\; \Delta$, then for all valuations σ and for all $\rho \in [\![\Gamma, \Delta]\!]_\sigma$ we have $t\rho \in [\![A]\!]^+_\sigma$.
2. If $\Gamma \;;\; e : A \vdash_{\mathsf{L}} \Delta$, then for all valuations σ and for all $\rho \in [\![\Gamma, \Delta]\!]_\sigma$ we have $e\rho \in [\![A]\!]^-_\sigma$.
3. If $c : (\Gamma \vdash_{\mathsf{L}} \Delta)$, then for all valuations σ and for all $\rho \in [\![\Gamma, \Delta]\!]_\sigma$ we have $c\rho \in \bot$.

where $t\rho$, $e\rho$, and $c\rho$ denotes the capture-avoiding application of ρ, seen as a substitution, to t, e, and c, respectively. ※

Proof: By simultaneous induction on the typing derivations, using the axioms about $\bot$. □

Example 7 (Strong Normalisation of System LK^{N})

We define $\mathcal{D}_t$ to be the set of strongly normalising terms and $\mathcal{D}_e$ to be the set of strongly normalising continuations. We define the orthogonality relation $\bot$ between $\mathcal{D}_t$ and $\mathcal{D}_e$ as those commands that are strongly normalising.[2]

We can check that the saturation requirements are met by purely syntactical/rewriting reasoning, but it only works because there is at most one way to reduce the top-level command.

Take σ to map every type variable to $\mathcal{D}_e$. Notice that term variables are in every $[\![A]\!]^+_\sigma$ and continuation variables are in every $[\![A]\!]^-_\sigma$, and that the identity substitution ρ is in every $[\![\Gamma, \Delta]\!]_\sigma$.

We then apply the Adequacy Lemma with σ and ρ, and get that every typed term, continuation, and command is strongly normalising for $\longrightarrow_{\mathsf{CBN}}$. ※

In this section, we have proved the strong normalisation of the confluent calculus LK^{N} for classical logic, a CBN-fragment of System L. We could have done it along the same lines for the full syntax of System L (but still with the confluent reduction $\longrightarrow_{\mathsf{CBN}}$), but dealing with the extra constructs and extra reductions (ζ_{N}) would have meant a heavier machinery (along the lines of [MM09, CMM10, MM13]). We aimed instead at simplicity, which emphasises the connection with the orthogonality models for System F, and those that we present in the next section.

In summary, in a confluent calculus such as the LK^{N}, building the positive and negative interpretations of a type A can be described as follows:

	Sets of terms	Sets of continuations
Stage 1		$Y_0 := [A]^-$
Stage 2	$X_1 := Y_0^{\bot}$	$Y_1 := Y_0^{\bot\bot}$
Finished	$[\![A]\!]^+ := X_1$	$[\![A]\!]^- := Y_1$

The construction is finished in 2 steps, because the sets X_1 and Y_1, which are orthogonal, already have all of the saturation properties required to contain all the terms and continuations of type A, which is checked when proving the Adequacy Lemma.

In other words, closure under bi-orthogonality provides adequate saturation properties.

2.2.2 The case of a non-confluent calculus

Let us now consider the situation of a non-confluent calculus such as System L with its original reduction system

[2] The notion of strong normalisation in the definition of $\mathcal{D}_t$, $\mathcal{D}_e$ and $\bot$ is of course considered for $\longrightarrow_{\mathsf{CBN}}$.

$$\begin{array}{lccl}(\to) & \langle \lambda x.t_1 \mid t_2 :: e\rangle & \longrightarrow & \langle t_2 \mid \mu x.\langle t_1 \mid e\rangle\rangle \\ (\overleftarrow{\mu}) & \langle \mu\beta.c \mid e\rangle & \longrightarrow & \{^{e}/_{\beta}\}\, c \\ (\overrightarrow{\mu}) & \langle t \mid \mu x.c\rangle & \longrightarrow & \{^{t}/_{x}\}\, c\end{array}$$

The Adequacy Lemma might still work if we had the saturation requirements:

If $\langle t_2 \mid \mu x.\langle t_1 \mid e\rangle\rangle\, \rho \in \bot$ then $\langle \lambda x.t_1 \mid t_2 :: e\rangle\, \rho \in \bot$
If $c(\rho, \beta \mapsto e) \in \bot$ and $e \in \mathcal{D}_e$ then $\langle \mu\beta.c \mid e\rangle\, \rho \in \bot$
If $c(\rho, x \mapsto t) \in \bot$ and $t \in \mathcal{D}_t$ then $\langle t \mid \mu x.c\rangle\, \rho \in \bot$

But in any case, because of non-confluence, these requirements are not met if we define $\mathcal{D}_t$ to be the set of strongly normalising terms and $\mathcal{D}_e$ to be the set of strongly normalising continuations, and $\bot$ the set of strongly normalising commands.[3]

This means that because of non-confluence, we need to change our notion of saturation, so that $[\![A]\!]^+$ and $[\![A]\!]^-$ respectively contain enough terms and continuations for the Adequacy Lemma to hold, and because of that change, the pair $([\![A]\!]^+, [\![A]\!]^-)$ will not be constructed in 2 steps as in the confluent case, but in infinitely many steps:

	Sets of terms		Sets of continuations	
Stage 1	X_0	$\bot$	Y_0	not saturated
Stage 2	X_1	$\bot$	Y_1	not saturated
Stage 3	X_2	$\bot$	Y_2	not saturated
	$\dots$	$\bot$	$\dots$	$\dots$
Stage ∞	X_∞	$\bot$	Y_∞	saturated
Finished	$[\![A]\!]^+$	$\bot$	$[\![A]\!]^-$	saturated

We get a saturated pair of sets in infinitely many steps (via a fixpoint construct). In [LM08], we showed that the fixpoint construct could not be captured by a bi-orthogonal completion step.

We now see the details of the technique. In the rest of this section, we fix $\bot$ to be the set of strongly normalising commands.

DEFINITION 32 (Orthogonality and saturation)

Let Lab_t denote the set of term variables and Lab_e denote the set of continuation variables.

Given a set $\mathcal{U}$ of terms and a set $\mathcal{V}$ of continuations, the pair $(\mathcal{U}, \mathcal{V})$ is

- *orthogonal* if $\forall t \in \mathcal{U}, \forall u \in \mathcal{V}, t \perp u$
- *saturated* if the following two conditions hold
  1. $\mathsf{Lab}_t \subseteq \mathcal{U}$ and $\mathsf{Lab}_e \subseteq \mathcal{V}$
  2. $\{\mu\alpha.c \,|\, \forall e \in \mathcal{V}, \{^{v}/_{x}\}\, c \in \bot\} \subseteq \mathcal{U}$ and $\{\mu x.c \,|\, \forall t \in \mathcal{U}, \{^{v}/_{x}\}\, t \in \bot\} \subseteq \mathcal{V}$.

A set of terms (resp. continuations) is said to be *simple* if it is non-empty and it contains no term of the form $\mu\alpha.c$ (resp. $\mu x.c$).

For every set X of terms (set Y of continuations), we define a function

$$\Phi_X(\mathcal{W}) := X \cup \mathsf{Lab}_t \cup \{\mu\alpha.c \,|\, \forall e \in \mathcal{W}, \{^{e}/_{\alpha}\}\, c \in \bot\}$$

resp.

$$\Phi_Y(\mathcal{W}) := Y \cup \mathsf{Lab}_e \cup \{\mu x.c \,|\, \forall t \in \mathcal{W}, \{^{t}/_{x}\}\, c \in \bot\}$$

※

[3] The notion of strong normalisation in the definition of $\mathcal{D}_t$, $\mathcal{D}_e$ and $\bot$ is now considered for the full reduction relation $\longrightarrow$.

Lemma 28 Given a set of terms X_0 and a set of continuations Y_0,

1. Φ_{X_0} and Φ_{Y_0} are anti-monotonic.[4]
2. Hence, $\Phi_{X_0} \circ \Phi_{Y_0}$ is monotonic and admits a fixpoint X_∞, with $\Phi_{X_0}(\Phi_{Y_0}(X_\infty)) = X_\infty$.
3. Writing Y_∞ for $\Phi_{Y_0}(X_\infty)$, we clearly have
$$X_\infty = X \cup \mathsf{Lab}_t \cup \{\mu\alpha.c \,|\, \forall e \in Y_\infty, \{e/\alpha\}\, c \in \bot\}$$
$$Y_\infty = Y \cup \mathsf{Lab}_e \cup \{\mu x.c \,|\, \forall t \in X_\infty, \{t/x\}\, c \in \bot\}$$
4. So (X_∞, Y_∞) is saturated, and a quick case analysis shows that it is orthogonal if X_0 and Y_0 are simple and orthogonal to each other.
5. Finally, $X_0 \subseteq X_\infty$ and $Y_0 \subseteq Y_\infty$.

We finally define $\mathsf{satur}(X_0, Y_0)$ as (X_∞, Y_∞). ※

Definition 33 (Interpretation of types for System L)

A *valuation* is a function, denoted σ, σ',..., from type variables to orthogonal pairs of simple sets.

Two interpretation of types are defined by simultaneous induction of types, a *positive interpretation* and a *negative interpretation*:

$$\begin{array}{rcl}
([a]^+_\sigma, [a]^-_\sigma) & := & \sigma(\alpha) \\
([A{\to}B]^+_\sigma, [A{\to}B]^-_\sigma) & := & (\{t \in ([\![A]\!]^+_\sigma :: [\![B]\!]^-_\sigma)^\bot \mid t \text{ not of the form } \mu\alpha.c\}, [\![A]\!]^+_\sigma :: [\![B]\!]^-_\sigma) \\
 & & \text{where } X::Y \text{ denotes } \{t::e \mid t \in X, e \in Y\} \\
([\![A]\!]^+_\sigma, [\![A]\!]^-_\sigma) & := & \mathsf{satur}([A]^+_\sigma, [A]^-_\sigma)
\end{array}$$

Again, we define

$$[\![\Gamma, \Delta]\!]_\sigma := \{\rho \mid \forall (x\!:\!A) \in \Gamma, \rho(x) \in [\![A]\!]^+_\sigma, \text{ and } \forall (\alpha\!:\!A) \in \Delta, \rho(x) \in [\![A]\!]^-_\sigma\}$$

※

Now notice the difference with Definition 31: the definition of $[A{\to}B]^-_\sigma$ is the same but if we just took $[\![A{\to}B]\!]^+_\sigma$ to be its orthogonal, the pair $([\![A{\to}B]\!]^+_\sigma, [A{\to}B]^-_\sigma)$ would not be saturated, as we have already seen; so instead we restrict $[A{\to}B]^{-\bot}_\sigma$ to those terms that are not of the form $\mu\alpha.c$, and thus form an orthogonal (but not saturated) pair of simple sets $([A{\to}B]^+_\sigma, [A{\to}B]^-_\sigma)$. Then we saturate that pair into $([\![A{\to}B]\!]^+_\sigma, [\![A{\to}B]\!]^-_\sigma)$, which is orthogonal and saturated:

Lemma 29 (Interpretations of types are orthogonal and saturated)

For all valuations σ and all types A, $([\![A]\!]^+_\sigma, [\![A]\!]^-_\sigma)$ is orthogonal and saturated. ※

The rest is now just as in the CBN case:

Lemma 30 (Adequacy Lemma for System L)

1. If $\Gamma \vdash_{\mathsf{L}} t\!:\!A \,;\, \Delta$, then for all valuations σ and for all $\rho \in [\![\Gamma, \Delta]\!]_\sigma$ we have $t\rho \in [\![A]\!]^+_\sigma$.
2. If $\Gamma \,;\, e\!:\!A \vdash_{\mathsf{L}} \Delta$, then for all valuations σ and for all $\rho \in [\![\Gamma, \Delta]\!]_\sigma$ we have $e\rho \in [\![A]\!]^-_\sigma$.
3. If $c\!:\!(\Gamma \vdash_{\mathsf{L}} \Delta)$, then for all valuations σ and for all $\rho \in [\![\Gamma, \Delta]\!]_\sigma$ we have $c\rho \in \bot$.

where $t\rho$, $e\rho$, and $c\rho$ denotes the capture-avoiding application of ρ, seen as a substitution, to t, e, and c, respectively. ※

Proof: By simultaneous induction on the typing derivations, using the Lemma 29. □

[4] In other words for Φ_{X_0}, if $\mathcal{W} \subseteq \mathcal{W}'$ then $\Phi_{X_0}(\mathcal{W}) \supseteq \Phi_{X_0}(\mathcal{W}')$. And similarly for Φ_{Y_0}.

THEOREM 31 (Strong Normalisation of System L)

Take σ to map every type variable to the orthogonal pair $(\mathsf{Lab}_t, \mathsf{Lab}_e)$ of simple sets. Notice again that the identity substitution ρ is in every $[\![\Gamma, \Delta]\!]_\sigma$.

We then apply the Adequacy Lemma with σ and ρ, and get that every typed term, continuation, and command is strongly normalising for $\longrightarrow$. ※

The points to remember are

- As for System F, we have proved strong normalisation by building a term model
  - which does not equate terms up to reduction (non-confluence would make that very problematic)
  - where axiom (A3) is replaced by a saturation property.
- Because of non-confluence,
  - saturation has to be a property of pairs $([\![A]\!]^+_\sigma, [\![A]\!]^-_\sigma)$, not a property of each component separately;
  - saturating is difficult (adding terms in one component of the pair affects the other component), and obtained by a fixpoint construction.

  As shown in [LM08], the saturation process is not just a bi-orthogonality completion: if $(\mathcal{U}, \mathcal{V})$ is orthogonal, then $(\mathcal{U}^{\perp\perp}, \mathcal{V}^{\perp\perp})$ is orthogonal but not necessarily saturated.

2.3 Orthogonality models for extracting witnesses from classical proofs

We now show how to extract a witness from a classical proof of a Σ^0_1-formula, i.e. a closed formula of the form $\exists a A(a)$ where $A(a)$ is a quantifier-free formula of arithmetics.

The technique is due to Miquel [Miq09, Miq11], we simply adapted it to our proof-term calculus for classical logic, and somewhat simplified it using the concepts and notations of the previous sections.

We work in a particular setting where such a formula is expressed in the shape of $\neg\forall a \neg \mathsf{isnull}(e(a))$, the grammar of formulae being defined as follows:

DEFINITION 34 (Expressions and Formulae)

$$\begin{array}{lllll} \textbf{Expressions} & u, u', \ldots & ::= & a \,|\, \mathsf{0} \,|\, \mathsf{s}(u) \,|\, u + u' \,|\, u \times u' \,|\, u \leq u' \\ \textbf{Formulae} & A, B, \ldots & ::= & \mathsf{isnull}(u) \,|\, A \to B \,|\, \forall a A \end{array}$$

We represent integers as expressions: let $\overline{0} := \mathsf{0}$ and, for all integers n, let $\overline{n+1} := \mathsf{s}(\overline{n})$.

We define $\neg A := A \to \mathsf{isnull}(\overline{1})$. ※

This shape for a Σ^0_1-formula brings no loss of generality. Moreover, such an expression as $u(a)$, with one free variable a, expresses a primitive recursive function from $\mathbb{N}$ to $\mathbb{N}$.

We will now build an orthogonality model that we will use for witness extraction. As in the previous sections, each formula A will be interpreted as a set $[\![A]\!]^+_\sigma$ of terms and a set $[\![A]\!]^-_\sigma$ of continuations, terms and continuations being those of LK^N.

The extraction mechanism itself will be given by the reductions of LK^{N}, and more precisely by **root** CBN-reduction, which we denote $\longrightarrow_{\mathsf{CBNr}}$.[5]

In other words, from a proof of $\neg\forall a \neg \mathsf{isnull}(u(a))$ in (the extension to arithmetic of) System L, we will perform $\longrightarrow_{\mathsf{CBNr}}$ -reduction until we reach (in a provably finite number of steps) a command where we can directly read a witness.

For this we need to express numerals as proof-terms. We simply use Church's numerals in λ-calculus (see e.g. [Bar84]) and encode them in LK^{N} with Definition 14:

Definition 35 (Church's numerals as terms)

$$\begin{array}{lcl} c_0 & := & \langle x \mid \alpha \rangle \\ c_{n+1} & := & \langle f \mid (\mu\alpha.c_n) :: \alpha \rangle \\ \underline{n} & := & \lambda x.\lambda f.\mu\alpha.c_n \end{array}$$

※

Remark 32 Doing the same thing with the λ-terms for the successor function and the recursion function, we get two LK^{N} terms $\mathbf{s}$ and $\mathbf{rec}$ such that, for all t, u_0, u_1, for all continuation values E, and all integer n,

$$\begin{array}{lcl} \langle \mathbf{s} \mid \underline{n} :: t :: E \rangle & \longrightarrow^*_{\mathsf{CBNr}} & \langle t \mid \underline{n+1} :: E \rangle \\ \langle \mathbf{rec} \mid u_0 :: u_1 :: \underline{0} :: E \rangle & \longrightarrow^*_{\mathsf{CBNr}} & \langle u_0 \mid E \rangle \\ \langle \mathbf{rec} \mid u_0 :: u_1 :: \underline{n+1} :: E \rangle & \longrightarrow^*_{\mathsf{CBNr}} & \langle u_1 \mid \underline{n} :: (\mu\alpha.\langle \mathbf{rec} \mid u_0 :: u_1 :: \underline{n} :: \alpha \rangle) :: E \rangle \end{array}$$

using the simulation of β-reduction by $\longrightarrow_{\mathsf{CBN}}$.[6]

Let $\mathbf{ifz} := \lambda n x_0 x_1.\mu\alpha.\langle \mathbf{rec} \mid x_0 :: (\lambda y_0 y_1.x_1) :: n :: \alpha \rangle$, so that

$$\begin{array}{lcl} \langle \mathbf{ifz} \mid \underline{0} :: u_0 :: u_1 :: E \rangle & \longrightarrow^*_{\mathsf{CBNr}} & \langle u_0 \mid E \rangle \\ \langle \mathbf{ifz} \mid \underline{n+1} :: u_0 :: u_1 :: E \rangle & \longrightarrow^*_{\mathsf{CBNr}} & \langle u_1 \mid E \rangle \end{array}$$

※

Definition 36 (Orthogonality semantics)

Let $\bot$ be an arbitrary set of commands, stable under anti-reduction (if $c \longrightarrow_{\mathsf{CBNr}} c'$ and $c' \in \bot$ then $c \in \bot$).

A valuation σ is a mapping from expression variables (a, etc) to integers.

Given a valuation σ, Fig. 11 defines the interpretation of an expression u as an integer $[\![u]\!]_\sigma$ and a formula A as a set $[\![A]\!]^+_\sigma$ of terms and a set $[\![A]\!]^-_\sigma$ of continuations. ※

Remark 33

1. Clearly, $[\![\overline{n}]\!]_\sigma = n$ for all n and σ.
2. By induction on u we get $\left[\!\left[\left\{ {}^{\overline{n}}\!/_{a} \right\} u \right]\!\right]_\sigma = [\![u]\!]_{\sigma, a \mapsto n}$, and by induction on A we get $\left[\left\{ {}^{\overline{n}}\!/_{a} \right\} A \right]^-_\sigma = [A]^-_{\sigma, a \mapsto n}$ and $\left[\!\left[\left\{ {}^{\overline{n}}\!/_{a} \right\} A \right]\!\right]^+_\sigma = [\![A]\!]^+_{\sigma, a \mapsto n}$.

※

Now, for simplicity we do not specify the exact proof system for arithmetic, nor do we give a typing system corresponding to it through the Curry-Howard correspondence. We assume that it could be built as an extension of Fig. 6, and that the Adequacy Lemma can be proved (along the lines of Lemma 27 for LK^{N}):

[5] The fact that we use CBN-reduction is important to make sure that reduction **can** produce a witness; the fact that we only use root reduction is not, but in order to implement the extraction mechanism deterministically, it is convenient to never have to choose the next redex to reduce.

[6] And the fact that we can do this with root-reduction only is rather clear.

$$
\begin{array}{lcll}
[\![a]\!]_\sigma & := & \sigma(a) & \\
[\![\mathsf{0}]\!]_\sigma & := & 0 & \\
[\![\mathsf{s}(u)]\!]_\sigma & := & [\![u]\!]_\sigma + 1 & \\
[\![u_1 + u_2]\!]_\sigma & := & [\![u_1]\!]_\sigma + [\![u_2]\!]_\sigma & \\
[\![u_1 \times u_2]\!]_\sigma & := & [\![u_1]\!]_\sigma \times [\![u_2]\!]_\sigma & \\
[\![u_1 \leq u_2]\!]_\sigma & := & 1 & \text{if } [\![u_1]\!]_\sigma \leq [\![u_2]\!]_\sigma \\
[\![u_1 \leq u_2]\!]_\sigma & := & 0 & \text{if } [\![u_1]\!]_\sigma > [\![u_2]\!]_\sigma
\end{array}
$$

$$
\begin{array}{lcll}
[\mathsf{isnull}(u)]^-_\sigma & := & \mathcal{E} & \text{if } [\![u]\!]_\sigma \neq 0 \\
[\mathsf{isnull}(u)]^-_\sigma & := & \emptyset & \text{if } [\![u]\!]_\sigma = 0 \\
[A{\to}B]^-_\sigma & := & [\![A]\!]^+_\sigma :: [\![B]\!]^-_\sigma & \\
[\forall a A]^-_\sigma & := & \bigcup_{n\in\mathbb{N}} (\{\underline{n}\} :: [\![A]\!]^-_{\sigma,a\mapsto n}) &
\end{array}
$$

$$
[\![A]\!]^+_\sigma := [A]^{-\perp}_\sigma \qquad [\![A]\!]^-_\sigma := [A]^{-\perp\perp}_\sigma
$$

where $\mathcal{E}$ is the set of all continuation values and $X :: Y$ denotes $\{t :: E \mid t \in X, E \in Y\}$.

Figure 11: Semantics of expressions and formulae

A closed proof t_0 of a formula $\neg\forall a \neg\mathsf{isnull}(u(a))$ is such that,
for all possible $\perp$ closed under "anti-reduction" (the inverse of $\longrightarrow_{\mathsf{CBNr}}$),
$t_0 \in [\![\neg\forall a \neg\mathsf{isnull}(u(a))]\!]$.

We thus start with such a term t_0.

We now define a term that can check whether an integer is a witness of the property and, depending on this check, continue with one term or another:

DEFINITION 37 (Witness checker)

Let f be the primitive recursive function defined by: for any integer n, $f(n) := [\![u(a)]\!]_{a\mapsto n}$.
Let $\underline{f}$ be a term representing f in the sense that, for any integer n, and term t and any continuation value E,

$$\left\langle \underline{f} \mid \underline{n} :: t :: E \right\rangle \longrightarrow^*_{\mathsf{CBNr}} \left\langle t \mid \underline{f(n)} :: E \right\rangle$$

Such a term can be constructed from **s** and **rec**, as the projections, composition, etc are all available in System L.
We define the *witness checker* as follows:

$$d_f := \lambda nxy.\mu\alpha.\left\langle \underline{f} \mid n :: (\lambda p.\mu\alpha_1.\langle \mathbf{ifz} \mid p :: x :: y :: \alpha_1\rangle) :: \alpha \right\rangle$$

※

LEMMA 34 (Witness checker property)

For any integer n, any u_0 and u_1 and E, we have
$\langle d_f \mid \underline{n} :: u_0 :: u_1 :: E\rangle \longrightarrow^*_{\mathsf{CBNr}} \langle u_0 \mid E\rangle$ if $f(n) = 0$
$\langle d_f \mid \underline{n} :: u_0 :: u_1 :: E\rangle \longrightarrow^*_{\mathsf{CBNr}} \langle u_1 \mid E\rangle$ if $f(n) \neq 0$ ※

Proof:

$$\begin{array}{rll}
\langle d_f \mid \underline{n}::u_0::u_1::E\rangle & \longrightarrow^*_{\mathsf{CBNr}} & \Big\langle \mu\alpha.\Big\langle \underline{f} \mid \underline{n}::(\lambda p.\mu\alpha_1.\langle \mathbf{ifz} \mid p::u_0::u_1::\alpha_1\rangle)::\alpha\Big\rangle \mid E\Big\rangle \\
& \longrightarrow^*_{\mathsf{CBNr}} & \Big\langle \underline{f} \mid \underline{n}::(\lambda p.\mu\alpha_1.\langle \mathbf{ifz} \mid p::u_0::u_1::\alpha_1\rangle)::E\Big\rangle \\
& \longrightarrow^*_{\mathsf{CBNr}} & \Big\langle \lambda p.\mu\alpha_1.\langle \mathbf{ifz} \mid p::u_0::u_1::\alpha_1\rangle \mid \underline{f(n)}::E\Big\rangle \\
& \longrightarrow^*_{\mathsf{CBNr}} & \Big\langle \mu\alpha_1.\Big\langle \mathbf{ifz} \mid \underline{f(n)}::u_0::u_1::\alpha_1\Big\rangle \mid E\Big\rangle \\
& \longrightarrow^*_{\mathsf{CBNr}} & \Big\langle \mathbf{ifz} \mid \underline{f(n)}::u_0::u_1::E\Big\rangle
\end{array}$$

If $f(n)=0$, this reduces to $\langle u_0 \mid E\rangle$. Otherwise, this reduces to $\langle u_1 \mid E\rangle$. □

Definition 38 (Orthogonality and contradicter)

Let **stop** be an arbitrary term and **go** be an arbitrary continuation.

We now take a particular orthogonality set defined by

$$\bot := \{c \big|\ \text{there exists } n \text{ such that } f(n)=0 \text{ and } c\longrightarrow^*_{\mathsf{CBNr}} \langle \mathbf{stop} \mid \underline{n}::\mathbf{go}\rangle\}$$

It is closed under anti-CBNr-reduction.

We now define a "contradicter":[7] Let $t_1 := \lambda nx.\mu\alpha.\langle d_f \mid n::(\mu\alpha_0.\langle \mathbf{stop} \mid n::\mathbf{go}\rangle)::x::\alpha\rangle$. ※

Lemma 35 (Behaviour of the contradicter)

For all integers n, and all continuation values E in $[\![\neg\mathsf{isnull}(u(\overline{n}))]\!]^-$, we have $t_1 \perp \underline{n}::E$. ※

Proof: We have

$$\langle t_1 \mid \underline{n}::E\rangle \longrightarrow^*_{\mathsf{CBNr}} \langle \lambda x.\mu\alpha.\langle d_f \mid \underline{n}::(\mu\alpha_0.\langle \mathbf{stop} \mid \underline{n}::\mathbf{go}\rangle)::x::\alpha\rangle \mid E\rangle$$

To prove that this is an orthogonal command, we only have to show, as $E \in [\neg\mathsf{isnull}(u(\overline{n}))]^{-\perp\perp}$, that the left-hand side term is orthogonal to every continuation in $[\neg\mathsf{isnull}(u(\overline{n}))]^-$.

Consider such a continuation; it is of the form $t::E'$ with $t \in [\![\mathsf{isnull}(u(\overline{n}))]\!]^+$.

If $f(n)\neq 0$ then $[u(\overline{n})]\neq 0$, so $[\mathsf{isnull}(u(\overline{n}))]^- = \mathcal{E}$ and t is orthogonal to every continuation, in particular E'. So we have

$$\begin{array}{rl}
& \langle \lambda x.\mu\alpha.\langle d_f \mid \underline{n}::(\mu\alpha_0.\langle \mathbf{stop} \mid \underline{n}::\mathbf{go}\rangle)::x::\alpha\rangle \mid t::E'\rangle \\
\longrightarrow^*_{\mathsf{CBNr}} & \langle d_f \mid \underline{n}::(\mu\alpha_0.\langle \mathbf{stop} \mid \underline{n}::\mathbf{go}\rangle)::t::E'\rangle \\
\longrightarrow^*_{\mathsf{CBNr}} & \langle t \mid E'\rangle
\end{array}$$

which is in $\bot$.

If $f(n)=0$ then we have

$$\begin{array}{rl}
& \langle \lambda x.\mu\alpha.\langle d_f \mid \underline{n}::(\mu\alpha_0.\langle \mathbf{stop} \mid \underline{n}::\mathbf{go}\rangle)::x::\alpha\rangle \mid t::E'\rangle \\
\longrightarrow^*_{\mathsf{CBNr}} & \langle d_f \mid n::(\mu\alpha_0.\langle \mathbf{stop} \mid \underline{n}::\mathbf{go}\rangle)::t::E'\rangle \\
\longrightarrow^*_{\mathsf{CBNr}} & \langle \mu\alpha_0.\langle \mathbf{stop} \mid \underline{n}::\mathbf{go}\rangle \mid E'\rangle \\
\longrightarrow^*_{\mathsf{CBNr}} & \langle \mathbf{stop} \mid \underline{n}::\mathbf{go}\rangle
\end{array}$$

□

Corollary 36 (Classical witness extraction)

$\langle t_0 \mid t_1::\mathbf{go}\rangle \longrightarrow^*_{\mathsf{CBNr}} \langle \mathbf{stop} \mid \underline{n}::\mathbf{go}\rangle$ for some integer n such that $f(n)=0$. ※

Proof:

From Lemma 35 we get that $t_1 \in [\![\forall a\neg\mathsf{isnull}(u(a))]\!]^+$ and therefore $t_1::\mathbf{go} \in [\neg\forall a\neg\mathsf{isnull}(u(a))]^-$.

Since we have assumed $t_0 \in [\![\neg\forall a\neg\mathsf{isnull}(u(a))]\!]^+$, we have $t_0 \perp t_1::\mathbf{go}$, from which we conclude. □

[7] In the sense that it will contradict what the proof t_0 claims.

In other words, once given a classical proof, we match it against the continuation $t_1 :: \mathbf{go}$ and we are certain that CBNr will produce $\langle \mathbf{stop} \mid \underline{n} :: \mathbf{go} \rangle$ in a finite number of steps, with n being a witness.

For a comparison with other techniques of classical witness extraction, see [Miq09, Miq11].

Conclusion

In summary, we have seen in this chapter a fundamental concept for model construction, namely orthogonality. We built several orthogonality models for various purposes: rephrase strong normalisation proofs for System F, prove the strong normalisation of a confluent proof-term calculus for classical logic as well as a non-confluent calculus (thereby proving cut-elimination), and finally extract witnesses from classical proofs of Σ_1^0-formulae.

In each of those model constructions, we have interpreted formulae first with basic inhabitants (terms or continuations), and then closed their interpretation by a completion process that could simply be taking the bi-orthogonal, in the case of confluent calculi, or a more complex fixpoint, in the case of a non-confluent calculus.

Whether in those constructions we first define the negative interpretation of a formula (as a set of "counter-proofs") or its positive interpretation (as a set of "proofs"), is a question that depends on the formula's *polarity*. This is the topic of the next chapter.

Chapter 3

Polarisation and focussing

Contents

In the previous chapters, we have seen that

- A proof-term syntax, together with a typing system, can be used to represent classical proofs (e.g. System L), such that the symmetry of classical logic is reflected by the symmetry between programs and continuations. The use of classical reasoning corresponds to letting a program capture its continuation.
- A rewrite system on proof-terms can be given to represent cut-elimination, following the intuitions of continuations and control. This gives a non-confluent calculus because (unrestricted) cut-elimination is non-confluent in classical logic, reflected by the fact that programs and continuations compete for the control of computation.
- Still, the rewrite system is strongly normalising on typed proof-terms (i.e. those representing real proofs), showing that cuts are admissible. The proof of strong normalisation was the occasion to introduce orthogonality techniques, although non-confluence requires more, namely specific saturation properties.
- The semantics of classical proofs, or typed proof-terms, is problematic until confluence is recovered in some way.

Back to the main issue, a CCC with $\neg\neg A \simeq A$ collapses, and out of the 3 natural ways to avoid the collapse, namely

1. break the symmetry between $\wedge$ and $\vee$,

2. break the cartesian product,
3. break the curryfication,

we investigate the breaking of symmetry between $\wedge$ and $\vee$.

In Chapter 1 we saw how to break the $\wedge\vee$-symmetry by the CBV/CBN approach. In this chapter, we break the $\wedge\vee$ symmetry by *polarisation*.

3.1 Recovering confluence by polarisation

3.1.1 Symmetry, asymmetry, and η-expansions

We start this section by coming back to a fundamental question: What is symmetrical about Classical Logic? There is definitely a symmetry based on the duality of negation / De Morgan's duality. It can be seen in the truth semantics of formulae, in e.g. truth tables or more generally in a boolean algebra: meet / join and top / bottom are swapped when flipping the order upside-down, and all the axioms of a boolean algebra are preserved.

At the level of proofs, there is also a symmetry that can be seen for instance in the two-sided sequent calculus: the left-introduction rule of a connective is symmetric to the right-introduction rule of its dual connective (in other words, the rules are preserved under duality flipping).

Cut-elimination is symmetrical (e.g. the rewrite system of Fig. 6), but to make semantical sense of it, one breaks the symmetry by making a choice between CBN and CBV that is completely arbitrary.

More interestingly, the following example reveals something asymmetric between the left-introduction of $\vee$ and the right-introduction of $\vee$:

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A\vee B \vdash \Delta} \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A\vee B, \Delta} \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A\vee B, \Delta}$$

Of course, we have never claimed that there is a symmetry between the left-introduction of $\vee$ and the right-introduction of $\vee$, but an interesting question is raised by the following situation: it is known (see e.g. [TS00]) that in the sequent calculus, the axiom rule

$$\frac{}{A \vdash A}$$

(say in a context-splitting setting) can be restricted, without losing logical completeness, to the *atomic* axiom rule

$$\frac{}{a \vdash a}$$

Every instance of the general instance can be replaced by a small proof only using atomic axioms, which is proved by induction on A: in particular, transforming the axiom $\dfrac{}{A\vee B \vdash A\vee B}$ into a proof with atomic axioms, we produce

$$\dfrac{\dfrac{\dfrac{\dots}{A \vdash A}}{A \vdash A\vee B} \qquad \dfrac{\dfrac{\dots}{B \vdash B}}{B \vdash A\vee B}}{A\vee B \vdash A\vee B}$$

and then recursively transform $A \vdash A$ and $B \vdash B$ (until all of the used axioms are atomic).

So the interesting question is whether there is a fundamental reason why $\vee$ is decomposed on the left before being decomposed on the right (looking at the bottom-up construction of the proof). Starting the decomposition on the right would have failed.

A related situation occurs with η-expansion in λ-calculus:

In λ-calculus, the use of an axiom corresponds to a variable in the proof-term. Typing the term

$$\lambda z^{A\to B}.z$$

(where we indicate the types of variables as superscripts) uses an axiom on $A\to B$. Typing its η-expansion

$$\lambda z^{A\to B}.\lambda y^{A}.z\ y$$

uses, strictly speaking, an axiom on $A\to B$ and an axiom on A, but as z is immediately applied and its type $A\to B$ immediately destructed, the η-expansion only uses, "morally" speaking, axioms on the smaller formulae A and B. Turning this moral intuition into something formal can be done by taking a proof-term calculus for sequent calculus (rather than natural deduction), as we shall see below, but still: we first have the λ-abstraction, and underneath it the application. Why again do they have to appear in that order?

Indeed, in a classical calculus such a System L, the axiom on $A\to B$ is represented as

$$\frac{}{z:A\to B\vdash z:A\to B\ ;}$$

The η-expansion of z is:

$$\dfrac{\dfrac{}{z:A\to B,y:A\vdash z:A\to B\ ;\alpha:B}\qquad\dfrac{\dfrac{}{z:A\to B,y:A\vdash y:A\ ;\alpha:B}\qquad\dfrac{}{z:A\to B,y:A\ ;\alpha:B\vdash\alpha:B}}{z:A\to B,y:A\ ;(y::\alpha):A\to B\vdash\alpha:B}}{\dfrac{\dfrac{\langle z\mid y::\alpha\rangle:(z:A\to B,y:A\vdash\alpha:B)}{z:A\to B,y:A\vdash\mu\alpha.\langle z\mid y::\alpha\rangle:B\ ;}}{z:A\to B\vdash\lambda y.\mu\alpha.\langle z\mid y::\alpha\rangle:A\to B\ ;}}$$

and then we can recursively transform the axioms on $y:A$ and $\alpha:B$ (until axioms are atomic). Of course, this η-expansion still features the use of $z:A\to B$, but only to implement a contraction (or even more precisely to implement the placing of the formula $A\to B$ where it can be decomposed), not to implement a proper axiom.

Now the η-expansion we used in the λ-calculus to illustrate our point is only one particular instance of η-expansion: the general form

$$M\longrightarrow_\eta\ \lambda y^{A}.M\ y\qquad y\notin\mathsf{FV}(M)$$

can be recovered, by the capture avoiding substitution of M for z, from the axiomatic η-expansion on axiom z:

$$z\longrightarrow_\eta\ \lambda y^{A}.z\ y$$

And in λ-calculus, no matter M (where y is not free), M and $\lambda y.M\ y$ have the same computational behaviour (with respect to β-reduction). In technical terms, M and $\lambda y.M\ y$ cannot be separated (even using untyped terms) [Bar84].

In System L, the η-expansion on axiom z

$$z\longrightarrow_\eta\ \lambda y.\mu\alpha.\langle z\mid y::\alpha\rangle$$

also provides, after instantiation of z by t (where y and α are not free), a general form of η-expansion:

$$t \longrightarrow_\eta \ \lambda y.\mu\alpha.\langle t \mid y::\alpha\rangle$$

which transforms

$$\Gamma \vdash t : A{\to}B \ ; \ \Delta$$

into

$$\dfrac{\dfrac{\Gamma \vdash t : A{\to}B \ ; \ \Delta \qquad \dfrac{\overline{\Gamma, y:A \vdash y:A \ ; \ \alpha:B, \Delta} \qquad \overline{\Gamma, y:A \ ; \ \alpha:B \vdash \alpha:B, \Delta}}{\Gamma, y:A \ ; \ (y::\alpha):A{\to}B \vdash \alpha:B, \Delta}}{\langle t \mid y::\alpha\rangle : (\Gamma, y:A \vdash \alpha:B, \Delta)}}{\dfrac{\Gamma, y:A \vdash \mu\alpha.\langle t \mid y::\alpha\rangle : B \ ; \ \Delta}{\Gamma \vdash \lambda y.\mu\alpha.\langle t \mid y::\alpha\rangle : A{\to}B \ ; \ \Delta}}$$

3.1.2 Towards polarised System L

Now the above general η-expansion can be instantiated with $t = \mu\beta.c$:

$$\mu\beta.c \longrightarrow_\eta \ \lambda y.\mu\alpha.\langle \mu\beta.c \mid y::\alpha\rangle$$

If we put those two terms in context, e.g. facing a continuation $\mu x.c'$, we get that

- $\langle \mu\beta.c \mid \mu x.c'\rangle$ rewrites to
$$\left\{ {}^{\mu\beta.c}\!/_{x} \right\} c' \qquad \text{or} \qquad \left\{ {}^{\mu x.c'}\!/_{\beta} \right\} c$$
- **but** $\langle \lambda y.\mu\alpha.\langle \mu\beta.c \mid y::\alpha\rangle \mid \mu x.c'\rangle$ rewrites only to
$$\left\{ {}^{\lambda y.\mu\alpha.\langle \mu\beta.c \mid y::\alpha\rangle}\!/_{x} \right\} c'$$

If η-convertible terms should have undistinguishable computational behaviour, we must forbid $\left\langle \mu\beta^{A_1\to A_2}.c \mid \mu x^{A_1\to A_2}.c' \right\rangle \longrightarrow \left\{ {}^{\mu x^{A_1\to A_2}.c'}\!/_{\beta} \right\} c$

The grounds for breaking the symmetry in such a way is that $\mu\beta^{A_1\to A_2}.c$ can be η-expanded, but $\mu x^{A_1\to A_2}.c'$ cannot, which reflects the fact that the right-introduction rule for $A_1 \to A_2$ is invertible while its left-introduction rule is not.

In short, when encountering

$$\dfrac{\dfrac{c : (\Gamma \vdash \beta : A_1{\to}A_2, \Delta)}{\Gamma \vdash \mu\beta.c : A_1{\to}A_2 \ ; \ \Delta} \qquad \dfrac{c' : (\Gamma, x : A_1{\to}A_2 \vdash \Delta)}{\Gamma \ ; \ \mu x.c' : A_1{\to}A_2 \vdash \Delta}}{\langle \mu\beta.c \mid \mu x.c'\rangle : (\Gamma \vdash \Delta)}$$

we could consider that the term $\mu\beta.c$ is a "cheater" in the sense that its type $A_1{\to}A_2$ could be proved or inhabited in another way (e.g. with the η-expansion of $\mu\beta.c$), avoiding the critical pair, and solving the non-confluence problem.

In particular, if β is used 0 times in c, or more than once, we can understand the typing tree

$$\dfrac{c : (\Gamma \vdash \beta : A_1{\to}A_2, \Delta)}{\Gamma \vdash \mu\beta.c : A_1{\to}A_2 \ ; \ \Delta}$$

as finishing with a weakening or a contraction. What η-expansion proves is that the proof can be transformed into a proof that finishes with a proper introduction of the implication.

In our earlier example about the connective $\vee$, it is the contrary: its left-introduction rule is invertible while its right-introduction rules are not.

This leads to considering a notion that arose from linear logic [Gir87]: *polarities*.

The intuition for *positive connectives* is that we expect no particular property of their right-introduction rules. These rules are called *synchronous*. In particular for goal-directed proof-search, applying such a rule bottom-up is *a priori* a choice which we may have to backtrack to if we fail to finish the proof. For logical completeness, (right-)weakenings or (right-)contractions may be necessary on a formula with a positive connective at its root.

The intuition for *negative connectives* is that their right-introduction rules *are* invertible. These rules are called *asynchronous*. In goal-directed proof-search we may apply such rules without loss of generality and therefore without creating backtrack points. Also, (right-)weakenings and (right-)contractions (on formulae that have a negative connective at their roots) are superfluous as far as logical completeness is concerned. On the other hand, the right-introduction rules "must interact well with the left-introduction rules" (or the right-introduction rules of the dual connective), in cut-elimination as well as in the expansion of axioms that we described in this section.

Just as in λ-calculus you can *always* inhabit a (non-empty) function type with a λ-abstraction, you can always η-expand an inhabitant of a type whose main connective is negative. Considering the η-expansion rules that we can apply in System L, we can derive the polarities of the three connectives we considered:

$$\begin{array}{llcl} \text{negative} & A{\to}B & t & \longrightarrow & \lambda y.\mu\alpha.\langle t \mid y :: \alpha\rangle \\ \text{negative} & A \wedge B & t & \longrightarrow & (\mu\alpha.\langle t \mid \mathsf{inj}_1(\alpha)\rangle, \mu\gamma.\langle t \mid \mathsf{inj}_2(\gamma)\rangle) \\ \text{positive} & A \vee B & e & \longrightarrow & (\mu x.\langle \mathsf{inj}_1(x) \mid e\rangle, \mu z.\langle \mathsf{inj}_2(z) \mid e\rangle) \end{array}$$

Now in order to solve the confluence problem, we also need to determine how to reduce $\langle \mu\beta.c \mid \mu x.c'\rangle$ when the cut-formula is atomic. This leads to splitting the set of the atomic formulae into positives and negatives as well. Unlike non-atomic formulae, the choice of polarity for each atom is arbitrary, and sometimes called the *bias* [LM09].

Now we can use these ideas to layer System L with polarities:

Definition 39 (Polarised System L) The polarised syntax of formulae is defined as

$$\begin{array}{lcl} P, P', \ldots & ::= & a^+ \mid A \vee B \\ N, N', \ldots & ::= & a^- \mid A \wedge B \mid A{\to}B \\ A, B, \ldots & ::= & P \mid N \end{array}$$

The syntax for proof-terms, together with their associated forms of typing judgements, is given below:

$$\begin{array}{llcll} -\text{terms} & t^- & ::= & x^- \mid \lambda x.t \mid (t_1, t_2) \mid \mu\beta^-.c & \Gamma \vdash t^- : N \,;\, \Delta \\ +\text{terms} & t^+ & ::= & x^+ \mid \mathsf{inj}_i(t) \mid \mu\beta^+.c & \Gamma \vdash t^+ : P \,;\, \Delta \\ \text{terms} & t & ::= & t^+ \mid t^- & \Gamma \vdash t : A \,;\, \Delta \\ & & & & \\ -\text{continuations} & e^- & ::= & \alpha^- \mid t :: e \mid \mathsf{inj}_i(e) \mid \mu x^-.c & \Gamma \,;\, e^- : N \vdash \Delta \\ +\text{continuations} & e^+ & ::= & \alpha^+ \mid (e_1, e_2) \mid \mu x^+.c & \Gamma \,;\, e^+ : P \vdash \Delta \\ \text{continuations} & t & ::= & e^+ \mid e^- & \Gamma \,;\, e : A \vdash \Delta \\ & & & & \\ \text{commands} & c & ::= & \langle t^+ \mid e^+\rangle \mid \langle t^- \mid e^-\rangle & c : (\Gamma \vdash \Delta) \end{array}$$

writing x for either x^+ or x^-. ※

Now that polarities explicitly appear in the syntax of proof-terms, it is easy to reduce $\langle \mu\alpha.c \mid \mu x.c'\rangle$:

$$\langle \mu\alpha^+.c \mid \mu x^+.c'\rangle \longrightarrow \left\{ {}^{\mu x^+.c'}\!/_{\alpha^+} \right\} c \quad \text{and} \quad \langle \mu\alpha^-.c \mid \mu x^-.c'\rangle \longrightarrow \left\{ {}^{\mu\alpha^-.c}\!/_{x^-} \right\} c'$$

This turns into the following rewrite system:

DEFINITION 40 (Reductions in the polarised System L) Again, we define values:

$$\begin{array}{lll} \text{term values} & V & ::= x \,|\, \mathsf{inj}_i(V) \,|\, t^- \\ \text{continuation values} & E & ::= \alpha \,|\, V::E \,|\, \mathsf{inj}_i(E) \,|\, e^+ \end{array}$$

The reduction relation $\longrightarrow_{\mathsf{F}}$ is defined as the contextual closure of the rules in Fig. 12. ※

$$\begin{array}{lll} (\rightarrow) & \langle \lambda x.t \mid V::E \rangle & \longrightarrow \left\langle \left\{ {}^{V}\!/_{x} \right\} t \mid E \right\rangle \\ (\wedge) & \langle (t_1, t_2) \mid \mathsf{inj}_i(E) \rangle & \longrightarrow \langle t_i \mid E \rangle \\ (\vee) & \langle \mathsf{inj}_i(V) \mid (e_1, e_2) \rangle & \longrightarrow \langle V \mid e_i \rangle \\ \\ (\overleftarrow{\mu}_-) & \langle \mu\beta^-.c \mid E \rangle & \longrightarrow \left\{ {}^{E}\!/_{\beta^+} \right\} c \\ (\overrightarrow{\mu}) & \langle t^- \mid \mu x^-.c \rangle & \longrightarrow \left\{ {}^{t^-}\!/_{x^-} \right\} c \\ (\overleftarrow{\mu}) & \langle \mu\beta^+.c \mid e^+ \rangle & \longrightarrow \left\{ {}^{e^+}\!/_{\beta^+} \right\} c \\ (\overrightarrow{\mu}_+) & \langle V \mid \mu x^+.c \rangle & \longrightarrow \left\{ {}^{V}\!/_{x^+} \right\} c \\ \\ (\zeta_{\mathsf{F}}) & \langle t^- \mid t^+::e \rangle & \longrightarrow \langle t^+ \mid \mu x^+.\langle t^- \mid x^+::e \rangle \rangle \\ (\zeta_{\mathsf{F}}) & \langle t^- \mid V::e^- \rangle & \longrightarrow \langle \mu\alpha.\langle t^- \mid V::\alpha \rangle \mid e^- \rangle \\ (\zeta_{\mathsf{F}}) & \langle t^- \mid \mathsf{inj}_i(e^-) \rangle & \longrightarrow \langle \mu\alpha.\langle t^- \mid \mathsf{inj}_i(\alpha) \rangle \mid e^- \rangle \\ (\zeta_{\mathsf{F}}) & \langle \mathsf{inj}_i(t^+) \mid e^+ \rangle & \longrightarrow \langle \mu x^+.\langle \mathsf{inj}_i(x^+) \mid e^+ \rangle \mid t^+ \rangle \end{array}$$

where the (ζ_{F})-rules apply only under the condition that t^+ and e^- are not values.

Figure 12: Rewrite system for polarised System L

As in the CBN and CBV cases, we have:

THEOREM 37 (Confluence and Subject Reduction)

$\longrightarrow_{\mathsf{F}}$ is confluent and satisfies Subject Reduction. ※

Notice that the notion of value is slightly different from that of Definition 21: Indeed, if $\wedge$ is to be taken to be negative, as the dual of (the obviously positive) $\vee$, we can take every pair to be a value (in Definition 21 we stuck to Wadler's presentation [Wad03]); this also removes the need for ζ-rules for pairs. On the other hand, for a continuation $t::e$ to be a value, we require it to be of the form $V::E$, as we no longer recover confluence by opposing left vs. right (terms vs. continuations) but by opposing positives vs. negatives.

Precisely because we now no longer make any distinction based on the left vs. right opposition (terms vs. continuations opposition), this system could equally be given as a one-sided system, merging the syntaxes of terms and continuations, but keeping of course the distinction between positive terms and negative terms.[1] At the level of formulae, we would get 4 connectives $\vee^+$ and $\wedge^+$ of positive polarity, and $\vee^-$ and $\wedge^-$ of negative polarity:

$$\begin{array}{llll} A \vee B & \text{becomes} & A \vee^+ B & \qquad A \rightarrow B \quad \text{becomes} \quad A^\perp \vee^- B \\ A \wedge B & \text{becomes} & A \wedge^- B & \qquad (A \rightarrow B)^\perp \quad \text{becomes} \quad A \wedge^+ B^\perp \end{array}$$

where $(A \rightarrow B)^\perp$ represents the dual of implication: *subtraction* (see e.g. [Cro04]).

[1] Otherwise we would get back to Barbanera and Berardi's symmetric (and non-confluent) λ-calculus, with unclear denotational semantics.

This is what we will do in Section 3.2.

3.1.3 Focussing

Now, the polarised System L presented above, which has been studied at length by Munch-Maccagnoni [MM09, CMM10, MM13], solves non-confluence, not by giving systematic priority to the left (CBV) or to the right (CBN), but by giving priority to the non-invertible side (depending on the connective).

So the system takes advantage of the invertibility properties of the asynchronous rules (right-introduction of negative connectives, left-introduction of positive connectives). Invertibility entails that, in terms of proof-search, you can *chain* the decomposition of every formula of the sequent that has an asynchronous introduction rule, before doing anything else, without loss of generality (i.e. without losing logical completeness).

Now in [AP89, And92], Andreoli proved a more surprising result: *focussing*, that says that, once you have chosen to decompose by a synchronous rule a particular formula in the sequent,[2] you can also chain without loss of generality (i.e. without losing logical completeness) the recursive decomposition of its subformulae by synchronous rules until you reveal a subformula of the opposite polarity (whose decomposition can then be done by asynchronous rules again).

This was in the context of linear logic, whence polarities have come, but it is now understood in other polarised logics (classical or intuitionistic). This result can be expressed as the completeness of a sequent calculus with a *focus* device, which syntactically highlights a formula in the sequent and forces the next proof-search step to decompose it with a synchronous rule, keeping the focus on its newly revealed subformulae. In terms of proof-search, focussing considerably reduces the search space, otherwise heavily redundant when Gentzen-style inference rules are used.

Focussed proofs are proofs that implement such a chaining of synchronous decompositions. The main idea is that focussed proofs are those whose proof-terms systematically use term values and continuation values, in other words, the normal forms for ζ-rules. Of course, such normal forms may feature cuts (ζ-rules introduce cuts), but one should notice the following properties:

Remark 38
Just like $\longrightarrow_{\zeta_{\mathsf{N}}}$ and $\longrightarrow_{\zeta_{\mathsf{V}}}$ (from Definition 21), the relation $\longrightarrow_{\zeta_{\mathsf{F}}}$ is terminating. ※

Definition 41 (LK^{F})
Let LK^{F} be the fragment of System L consisting of $\longrightarrow_{\zeta_{\mathsf{F}}}$-normal forms. ※

Remark 39 Just like LK^{N} and LK^{V} are stable under $\longrightarrow_{\mathsf{CBN}}$ and $\longrightarrow_{\mathsf{CBV}}$, the fragment LK^{F} is stable under $\longrightarrow_{\mathsf{F}}$. ※

Remark 40 These normal form fragments relate to calculi of the literature:

1. LK^{N} is exactly the calculus called LKT [DJS95, DJS97];
2. LK^{V} is exactly the calculus called LKQ [DJS95, DJS97];
3. LK^{F} relates to Liang and Miller's LKF [LM09], and this will be the object of Section 3.2.

※

[2] Positive formula on the right-hand side of a sequent, or a negative formula on its left-hand side

We therefore have 3 versions of the focussing result in classical logic, an unfocussed proof c can be turned into a focussed proof c' (in the sense of LK^{N}, LK^{V}, or LK^{F}) by normalising it with respectively $\longrightarrow_{\zeta_{\mathsf{N}}}$, $\longrightarrow_{\zeta_{\mathsf{V}}}$, or $\longrightarrow_{\zeta_{\mathsf{F}}}$, and normalising it by respectively $\longrightarrow_{\mathsf{CBN}}$, $\longrightarrow_{\mathsf{CBV}}$, or $\longrightarrow_{\mathsf{F}}$ to eliminate cuts and finally obtain a cut-free focussed proof.

3.1.4 Weak η-conversion

Now, the notion of η-conversion that we used as an introduction to polarities in Section 3.1.2, is a *strong* notion of η-conversion:

Inspired by the way we can reduce an axiom on a non-atomic formula into a proof using axioms on smaller formulae, we considered the η-expansion of variables x and α:

$$\begin{array}{rcl} x & \longrightarrow & \lambda y.\mu\alpha.\langle x \mid y :: \alpha\rangle \\ x & \longrightarrow & (\mu\alpha.\langle x \mid \mathsf{inj}_1(\alpha)\rangle, \mu\gamma.\langle x \mid \mathsf{inj}_2(\gamma)\rangle) \\ \alpha & \longrightarrow & (\mu x.\langle \mathsf{inj}_1(x) \mid \alpha\rangle, \mu z.\langle \mathsf{inj}_2(z) \mid \alpha\rangle) \end{array}$$

which we sought to generalise to

$$\begin{array}{rcl} t & \longrightarrow & \lambda y.\mu\alpha.\langle t \mid y :: \alpha\rangle \\ t & \longrightarrow & (\mu\alpha.\langle t \mid \mathsf{inj}_1(\alpha)\rangle, \mu\gamma.\langle t \mid \mathsf{inj}_2(\gamma)\rangle) \\ e & \longrightarrow & (\mu x.\langle \mathsf{inj}_1(x) \mid e\rangle, \mu z.\langle \mathsf{inj}_2(z) \mid e\rangle) \end{array}$$

for *any* term t and any continuation e.

This led to a polarity-based reduction relation that contrasts with the CBN and CBV reduction relations from Chapter 1. But that does not mean that CBN and CBV are incompatible with the concept of η-conversion: they just require *weaker* notions of η-conversion than that discussed above.

The notion of η-conversion that is suitable for CBN are

$$\begin{array}{rcl} t & \longrightarrow & \lambda y.\mu\alpha.\langle t \mid y :: \alpha\rangle \\ t & \longrightarrow & (\mu\alpha.\langle t \mid \mathsf{inj}_1(\alpha)\rangle, \mu\gamma.\langle t \mid \mathsf{inj}_2(\gamma)\rangle) \\ E & \longrightarrow & (\mu x.\langle \mathsf{inj}_1(x) \mid E\rangle, \mu z.\langle \mathsf{inj}_2(z) \mid E\rangle) \end{array}$$

where α has not been substituted by any continuation e but *only* by a continuation value E.

The notion of η-conversion that is suitable for CBV are

$$\begin{array}{rcl} V & \longrightarrow & \lambda y.\mu\alpha.\langle V \mid y :: \alpha\rangle \\ V & \longrightarrow & (\mu\alpha.\langle V \mid \mathsf{inj}_1(\alpha)\rangle, \mu\gamma.\langle V \mid \mathsf{inj}_2(\gamma)\rangle) \\ e & \longrightarrow & (\mu x.\langle \mathsf{inj}_1(x) \mid e\rangle, \mu z.\langle \mathsf{inj}_2(z) \mid e\rangle) \end{array}$$

where x has not been substituted by any term t but *only* by a term value V.

Including these notions of CBN-η-conversion and CBV-η-conversion in the CBN and CBV notions of reduction, is actually necessary if these are to capture the semantics of classical proofs in control and co-control categories, respectively: just like in Theorem 4 we needed η-conversion to make the simply-typed λ-calculus sound and complete with respect to the semantics given by CCC, here we would need the above notions of CBN-η-conversion and CBV-η-conversion in order to turn the implications of Theorem 22 (soundness) into equivalences (soundness and completeness). This is actually what Selinger proved [Sel01] in the context of the $\lambda\mu$-calculus.

3.1.5 Related works

The role of polarities and focussing in classical proof theory has been investigated by a substantial literature, inspired by Girard's linear logic [Gir87]. Following this work and Andreoli's on focussing [AP89, And92], Girard developed in [Gir91] a sequent calculus LC for classical logic with more structure than Gentzen's LK [Gen35], based on an assignment of polarities to classical formulae. Danos, Joinet and Schellinx [DJS95, DJS97] studied semantically meaningful ways to make cut-elimination confluent in the classical sequent calculus, introducing

- the calculi LKT and LKQ mentioned above
- a version of the sequent calculus called LK^{tq} where a *colour* t or q on each formula indicates whether a cut on that formula should be pushed to the right or to the left,
- more restricted versions thereof,

all inspired by the various translations of classical logic into linear and intuitionistic logics. Out of that field, which includes the duality betweenn CBN and CBV,[3] *polarised classical logic* emerged, developed as such by Laurent et al. [Lau02, LQdF05]. It develops and enriches Girard's work on LC, in particular by explaining the proof theory of classical formulae as given by LC as a combination of

- an encoding from classical formulae to polarised classical formulae
- a proof theory for polarised classical logic.

Closer to Andreoli's original line of research, which was motivated by logic programming, Liang and Miller then formalised LKF as a more strongly focussed calculus than that called LK^{F} above; we will study it in the next section.

A useful introduction to that literature can be found in Chapter 2 of Farooque's thesis [Far13].

More recently, Munch-Maccagnoni approached the concept of focussing via orthogonality models [MM09]. He built for the polarised version of System L the same kind of orthogonality model as the one we presented in Section 2.2.1 for the LK^{N}-case, with an interpretation $[\![A]\!]^+$ of a formula A built as the orthogonal or bi-orthogonal of a more basic set of (counter-)proof-terms. He essentially shows that the interpretation $[\![A]\!]^+$ of a formula is generated from its values, in the sense that $[\![A]\!]^+ = ([\![A]\!]^+ \cap \mathcal{V})^{\perp\perp}$ where $\mathcal{V}$ denotes the set of term values (and symmetrically $[\![A]\!]^- = ([\![A]\!]^- \cap \mathcal{E})^{\perp\perp}$ where $\mathcal{E}$ denotes the set of continuation values).

This sheds an interesting light on our definition of

$$[A{\to}B]^-_\sigma := [\![A]\!]^+_\sigma :: [\![B]\!]^-_\sigma$$

in our orthogonality models of LK^{N} (Definitions 31 and 36): While in LK^{N} the above construct only considers those inhabitants of $[\![B]\!]^-_\sigma$ that are continuation values anyway, there is in the general case of System L a question of whether we want continuations of the form $t::\mu x.c$,[4] which are not focussed (in the sense of LK^{N} or LK^{F}), in the interpretation $[A{\to}B]^-_\sigma$. The result that $[\![A{\to}B]\!]^- = ([\![A{\to}B]\!]^- \cap \mathcal{E})^{\perp\perp}$ means that an "unfocussed" counter-proof such as $t::\mu x.c$ would be accepted after the bi-orthogonal completion (i.e. in $[\![A{\to}B]\!]^-_\sigma = ([A{\to}B]^-_\sigma)^{\perp\perp}$).

So far we have taken advantage of focussing, i.e. the chaining of synchronous rules, to identify complete fragments LK^{N}, LK^{V}, and LK^{F} of classical sequent calculus proofs.

[3] As revealed by Curien and Herbelin's System L [CH00] and Selinger's control categories [Sel01].
[4] (with $t \in [\![A]\!]^+_\sigma$ and $\mu x.c \in [\![B]\!]^-_\sigma$)

Although we have discussed invertibility of asynchronous rules, in order to introduce the notion of polarity, we have not forced our proofs to apply asynchronous rules eagerly, before applying other rules (in LK^{F}, $\mu\alpha^{-}.c$ is still an accepted proof of a negative formula such as $A{\to}B$).

This is the main difference with the focussed proof systems in the style of e.g. Liang and Miller [LM09], where e.g. all proofs of $A{\to}B$ finish with the right-introduction of $\to$. In terms of proof-terms, it means that all proof-terms are in η-long normal forms (we can transform every proof-term into a proof-term in that form by a series of η-expansions, but it is always tricky to control the termination of η-expansion without having the types explicitly in the terms).

Coming back to the purely logical level, focussed proofs in the tradition of Miller et al. can be described in terms of "big-step focussing": going up a branch of the proof is an alternation of sychronous and asynchronous phases, which we may consider to be atomic. The next section shows a computational interpretation of that strongly focussed formalism.

3.2 Computational interpretation of a focussed calculus

The starting point of this section is Liang and Miller's LKF [LM09], a variant of the system LK^{F} described in the previous section that forces the asynchronous decomposition of formulae. It is described in purely logical terms and we will see how, by formalising the concept of *big-step focussing*, a Curry-Howard interpretation can be given to LKF, following Zeilberger's work [Zei08a, Zei08b].

We start with the formulae of *polarised classical logic*.

DEFINITION 42 (Polarised formulae)

The syntax of formulae is given by the following grammar

$$\begin{array}{lcl} \textit{Positive formulae} & P & ::= a \mid A_1 \wedge^{+} A_2 \mid A_1 \vee^{+} A_2 \\ \textit{Negative formulae} & N & ::= a^{\perp} \mid A_1 \wedge^{-} A_2 \mid A_1 \vee^{-} A_2 \\ \textit{Formulae} & A & ::= P \mid N \end{array}$$

where a ranges over a fixed set of elements called *positive atoms*, and $a^{\perp}$ ranges over a bijective copy of that set ($a \mapsto a^{\perp}$ is the bijection), whose elements are called *negative atoms*.

We extend the bijection between positive and negative atoms into an involutive bijection, called *negation*, between positive and negative formulae:

$$\begin{array}{|lcl|lcl|} \hline (a)^{\perp} & := & a^{\perp} & (a^{\perp})^{\perp} & := & a \\ (A_1 \wedge^{+} A_2)^{\perp} & := & A_1^{\perp} \vee^{-} A_2^{\perp} & (A_1 \wedge^{-} A_2)^{\perp} & := & A_1^{\perp} \vee^{+} A_2^{\perp} \\ (A_1 \vee^{+} A_2)^{\perp} & := & A_1^{\perp} \wedge^{-} A_2^{\perp} & (A_1 \vee^{-} A_2)^{\perp} & := & A_1^{\perp} \wedge^{+} A_2^{\perp} \\ \hline \end{array}$$

※

Following the suggestion made in the previous section, we fold LKF into a 1-sided sequent calculus (hence our interest for the involutive negation), as it is traditional in the field arising from linear logic [Gir87].

EXAMPLE 8 For instance, Peirce's, law, which in the previous chapters and sections we wrote as $((a{\to}b){\to}a){\to}a$, is now $((a^{\perp} \vee^{-} b) \wedge^{+} a^{\perp}) \vee^{-} a$. ※

Definition 43 (Liang-Miller's LKF)

The rules of LKF are given in Fig. 13 for two kinds of sequents:
$\vdash \Theta \Downarrow A$ focussed sequent
$\vdash \Theta \Uparrow \Gamma$ unfocussed sequent
where A is an arbitrary formula, Θ is a multiset of either negative atoms or positive formulae and Γ is a multiset of arbitrary formulae.

Derivability in LKF of the sequents $\vdash \Theta \Downarrow A$ and $\vdash \Theta \Uparrow \Gamma$ is respectively denoted $\vdash_{\mathsf{LKF}} \Theta \Downarrow A$ and $\vdash_{\mathsf{LKF}} \Theta \Uparrow \Gamma$. ※

Synchronous phase	$\dfrac{\vdash \Theta \Downarrow A_1 \qquad \vdash \Theta \Downarrow A_2}{\vdash \Theta \Downarrow A_1 \wedge^+ A_2}$ $\qquad$ $\dfrac{\vdash \Theta \Downarrow A_i}{\vdash \Theta \Downarrow A_1 \vee^+ A_2}$
End of synchronous phase	$\dfrac{\vdash \Theta \Uparrow N}{\vdash \Theta \Downarrow N}$ $\qquad$ $\dfrac{}{\vdash \Theta \Downarrow a}\ a^\perp \in \Theta$
Asynchronous phase	$\dfrac{\vdash \Theta \Uparrow A_1, \Gamma \qquad \vdash \Theta \Uparrow A_2, \Gamma}{\vdash \Theta \Uparrow A_1 \wedge^- A_2, \Gamma}$ $\qquad$ $\dfrac{\vdash \Theta \Uparrow A_1, A_2, \Gamma}{\vdash \Theta \Uparrow A_1 \vee^- A_2, \Gamma}$
End of asynchronous phase	$\dfrac{\vdash \Theta, P \Uparrow \Gamma}{\vdash \Theta \Uparrow P, \Gamma}$ $\qquad$ $\dfrac{\vdash \Theta, p^\perp \Uparrow \Gamma}{\vdash \Theta \Uparrow p^\perp, \Gamma}$ $\qquad$ $\dfrac{\vdash \Theta, P \Downarrow P}{\vdash \Theta, P \Uparrow}$

Figure 13: LKF

Liang and Miller showed in [LM09] that

- various cut-rules are admissible;
- the polarities of atoms and connectives do not change the provability of an unfocussed sequent, but they change the shape of its proofs;
- with the admissibility of cut-rules, the system is complete for classical logic, no matter which polarities are placed on connectives and literals.

To prove cut-admissibility, they do not explicitly formalise a cut-elimination procedure, but it could probably be inferred from the proof.

Note that soundness of the system with respect to classical logic is trivially checked, rule by rule, forgetting about polarities and the structure of sequents.

Polarities in classical logic raise interesting questions: $A \wedge^+ B$ and $A \wedge^- B$ are equiprovable, and so are $A \vee^+ B$ and $A \vee^- B$. But, while the difference between the (direct) proofs of $A \vee^+ B$ and (direct) proofs of $A \vee^- B$ is clear,[5] one may wonder what the real difference is between (direct) proofs of $A \wedge^+ B$ and (direct) proofs of $A \wedge^- B$, given that the two rules look very much alike.

The difference lies not in their structure, but in the way they will behave in cut-elimination:

- from a proof of $A \wedge^- B$ (facing a proof of $A^\perp \vee^+ B^\perp$), only one sub-proof is used while the other is thrown away,
- from a proof of $A \wedge^+ B$ (facing a proof of $A^\perp \vee^- B^\perp$), both sub-proofs are used.

[5] Direct proofs of $A \vee^+ B$ choose one side and throw away the other, while direct proofs of $A \vee^- B$ keep the two sides.

Metaphorically, proving either conjunction is like picking 1 boy name and 1 girl name, when your couple is pregnant: proving a negative conjunction is picking the two names when you are expecting one baby (not knowing whether it is a boy or a girl), while proving the positive conjunction is picking the two names when expecting twins (a boy and a girl). On the paper, you have the same job to do, but you will probably approach the problem very differently.

3.2.1 Informal relation to System L

It may not be obvious, but system LKF roughly expresses, without proof-terms, some derivations of System L in a 1-sided format.[6]

Indeed, think of

- a focussed sequent $\vdash \Theta \Downarrow A$ as a typing judgement for a term value V: $\quad \vdash V : A \ ; \Theta$.
- an unfocussed sequent $\vdash \Theta \Uparrow \Gamma$ as a typing judgement for a command c: $\quad c : (\vdash \ ; \Theta, \Gamma)$.

with Γ being the part of the typing context for negative continuation variables α^- with non-atomic types (which can be asynchronously decomposed), and Θ the rest of it (typing negative continuation variables α^- with atomic types, and typing positive continuation variables α^+).

To derive a focussed sequent $\vdash \Theta \Downarrow A$:

- the two rules of the group 'Synchronous phase' correspond to the typing rules for $V :: V'$ and for $\mathsf{inj}_i(V')$;
- the first rule of the group 'End of synchronous phase' does not correspond to a rule of System L but simply the realisation that the value V is a negative term t^-;
- the second rule of that group is when V is a variable.

To derive an unfocussed sequent $\vdash \Theta \Uparrow \Gamma$:

- the two rules of the group 'Asynchronous phase' correspond to the typing of (t_1, t_2) and $\lambda\alpha.t$;
- the first two rules of the group 'End of asynchronous phase' are not reflected in System L (they just move formulae that cannot be asynchronously decomposed from Γ to Θ)
- the third rule of that phase corresponds to the typing of $\langle V \mid \alpha^+ \rangle$.

Roughly speaking, we should think of LKF as typing those proof-terms of (a 1-sided version of) System L that are η-long $\longrightarrow_{\mathsf{F}}$ -normal forms. These are described by the following grammar:

$$\begin{array}{lcl} V, V' & ::= & \alpha^+ \,|\, V :: V' \,|\, \mathsf{inj}_i(V) \,|\, t^- \,|\, \mu\alpha^-.c \\ t^-, \ldots & ::= & (t_1, t_2) \,|\, \lambda\alpha.t \\ t, \ldots & ::= & \mu\alpha^+.c \,|\, t^- \,|\, \mu\alpha^-.c \\ c, \ldots & ::= & \langle V \mid \alpha^+ \rangle \;|\; \langle t^- \mid \alpha^- \rangle \end{array}$$

where the type of every $\mu\alpha^-.c$ is atomic.

It is only "roughly speaking", because in Liang-Miller's LKF:

1. when a formula is asynchronously decomposed, no copy of the formula is kept in the sequent (which means in the above grammar that in every command $\langle t^- \mid \alpha^- \rangle$ we impose $\alpha^- \notin \mathsf{FV}(t^-)$),

[6] A 1-sided version of System L would merge terms and continuations into terms, so that $V :: V'$ is a term value, and merge term variables and continuation variables into continuation variables.

2. when the focus is placed on a formula, all the formulae that could be asynchronously decomposed have already been asynchronously decomposed (which means in the above grammar that in every command $\langle V \mid \alpha^+ \rangle$, V has no free variable of the form α^+).
3. finally, the order in which formulae are decomposed in the asynchronous phase, is less deterministic than that imposed by the above grammar.

This is because, in Liang and Miller's view of focussing, and more generally in the tradition of linear logic, "what happens in the asynchronous phase stays in the asynchronous phase", in the sense that the details of the asynchronous phase (e.g. the order in which formulae are decomposed) are meaningless and should not impact the semantics of the proof.

This is difficult to reflect at the level of System L's proof-terms.

Therefore, we will now develop a Curry-Howard interpretation for that particular view of focussing, along the lines of Zeilberger's work [Zei08a, Zei08b, Zei10]: in order to forget about the inner details of the asynchronous phase, we formalise the idea of compacting each phase (asynchronous and even synchronous) into one atomic inference. This is called big-step focussing.

3.2.2 Identifying phases as atomic steps

We start by showing an example of how positive connectives are decomposed.

Example 9 Trying to prove $\vdash \Theta \Downarrow N_1 \wedge^+ (a \vee^+ N_2)$ we can build:

either

$$\dfrac{\dfrac{\text{End of synch phase}}{\vdash \Theta \Uparrow N_1}}{\vdash \Theta \Downarrow N_1} \qquad \dfrac{\dfrac{\text{End of synch phase}}{a^\perp \in \Theta}}{\vdash \Theta \Downarrow a}}{\vdash \Theta \Downarrow a \vee^+ N_2}$$
$$\vdash \Theta \Downarrow N_1 \wedge^+ (a \vee^+ N_2)$$

or

$$\dfrac{\dfrac{\text{End of synch phase}}{\vdash \Theta \Uparrow N_1}}{\vdash \Theta \Downarrow N_1} \qquad \dfrac{\dfrac{\text{End of synch phase}}{\vdash \Theta \Uparrow N_2}}{\vdash \Theta \Downarrow N_2}}{\vdash \Theta \Downarrow a \vee^+ N_2}$$
$$\vdash \Theta \Downarrow N_1 \wedge^+ (a \vee^+ N_2)$$

The whole synchronous phase can be expressed in just one step:

$$\dfrac{\vdash \Theta \Uparrow N_1 \quad a^\perp \in \Theta}{\vdash \Theta \Downarrow N_1 \wedge^+ (a \vee^+ N_2)} \qquad \text{or} \qquad \dfrac{\vdash \Theta \Uparrow N_1 \quad \vdash \Theta \Uparrow N_2}{\vdash \Theta \Downarrow N_1 \wedge^+ (a \vee^+ N_2)}$$

In other words:

$$\dfrac{\forall N \in \Gamma, \quad \vdash \Theta \Uparrow N \qquad \forall a \in \Gamma, \quad a^\perp \in \Theta}{\vdash \Theta \Downarrow N_1 \wedge^+ (a \vee^+ N_2)}$$

with $\Gamma = N_1, a$ or $\Gamma = N_1, N_2$

In either case, we say that Γ "is a positive decomposition of" $N_1 \wedge^+ (a \vee^+ N_2)$, which we denote: $N_1, a \Vdash^+ N_1 \wedge^+ (a \vee^+ N_2)$ and $N_1, N_2 \Vdash^+ N_1 \wedge^+ (a \vee^+ N_2)$. ※

Now we generalise this example into a formal definition:

DEFINITION 44 (Decomposition of positive connectives)

The *positive decomposition relation* is the binary relation, defined by the rules of Fig. 14, where $\Gamma, \Gamma_1, \Gamma_2$ are sets of positive atoms or negative formulae.

The *one-step synchronous phase* is the rule:

$$\frac{\Gamma \Vdash^+ A \qquad \forall N \in \Gamma,\ \vdash \Theta \Uparrow N \qquad \forall a \in \Gamma, \quad a^\perp \in \Theta}{\vdash \Theta \Downarrow A}\ \mathsf{synch}$$

※

$$\frac{}{N \Vdash^+ N} \qquad \frac{}{a \Vdash^+ a}$$

$$\frac{\Gamma_1 \Vdash^+ A_1 \qquad \Gamma_2 \Vdash^+ A_2}{\Gamma_1, \Gamma_2 \Vdash^+ A_1 \wedge^+ A_2} \qquad \frac{\Gamma \Vdash^+ A_i}{\Gamma \Vdash^+ A_1 \vee^+ A_2}$$

Figure 14: Positive decomposition relation

Notice the syntax we use for the synch rule: the symbols $\forall$ and $\in$ are **meta-level** symbols: the number of premisses is the cardinal of Γ (plus one if you count $\Gamma \Vdash^+ A$).

We now show an example of how negative connectives are decomposed.

EXAMPLE 10

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{End of asynch phase}}{\vdash \Theta, P_1, a^\perp \Uparrow} \qquad \dfrac{\dfrac{\text{End of asynch phase}}{\vdash \Theta, P_1, P_2 \Uparrow}}{\vdash \Theta \Uparrow P_1, P_2}}{\vdash \Theta \Uparrow P_1, a^\perp}}{\vdash \Theta \Uparrow P_1, (a^\perp \wedge^- P_2)}}{\vdash \Theta \Uparrow P_1 \vee^- (a^\perp \wedge^- P_2)}$$

The whole asynchronous phase can be expressed in just one step:

$$\frac{\vdash \Theta, P_1, a^\perp \Uparrow \qquad \vdash \Theta, P_1, P_2 \Uparrow}{\vdash \Theta \Uparrow P_1 \vee^- (a^\perp \wedge^- P_2)}\ \mathsf{asynch}$$

In other words

$$\frac{\forall \Delta, \quad \vdash \Theta, \Delta \Uparrow}{\vdash \Theta \Uparrow P_1 \vee^- (a^\perp \wedge^- P_2)}$$

where Δ ranges over $\{\ \{P_1, a^\perp\}\ ,\ \{P_1, P_2\}\ \}$

In either case, we say that Δ "is a negative decomposition of" $P_1 \vee^- (a^\perp \wedge^- P_2)$, which we denote $P_1, a^\perp \Vdash^- P_1 \vee^- (a^\perp \wedge^- P_2)$ and $P_1, P_2 \Vdash^- P_1 \vee^- (a^\perp \wedge^- P_2)$. ※

Now we generalise this example into a formal definition:

DEFINITION 45 (Decomposition of negative connectives)

The *negative decomposition relation* is the binary relation, defined by the rules of Fig. 15, where where $\Delta, \Delta_1, \Delta_2$ are sets of negative atoms or positive formulae.

The *one-step asynchronous phase* is the rule:

$$\frac{\forall \Delta, (\Delta \Vdash^- A) \Rightarrow (\vdash \Theta, \Delta \Uparrow)}{\vdash \Theta \Uparrow A}$$

※

$$\frac{}{P \Vdash^- P} \qquad \frac{}{a^\perp \Vdash^- a^\perp}$$

$$\frac{\Delta \Vdash^- A_i}{\Delta \Vdash^- A_1 \wedge^- A_2} \qquad \frac{\Delta_1 \Vdash^- A_1 \qquad \Delta_2 \Vdash^- A_2}{\Delta_1, \Delta_2 \Vdash^- A_1 \vee^- A_2}$$

Figure 15: Negative decomposition relation

Again, notice that the syntax we use for the asynch rule uses the **meta-level** symbols $\forall$ and $\Rightarrow$: the number of premisses is the number of Δ satisfying $\Delta \Vdash^- A$ for the given A.

We now put everything together in the style of Zeilberger [Zei08a, Zei08b, Zei10].

Definition 46 (Big-step LKF, v1)

The big-step LKF system is given in Fig. 16, where Θ, Δ are sets of negative atoms or positive formulae and Γ is a set of positive atoms or negative formulae. ※

$$\frac{\Gamma \Vdash^+ A \qquad \forall N \in \Gamma, \vdash \Theta \Uparrow N \qquad \forall a \in \Gamma, \quad a^\perp \in \Theta}{\vdash \Theta \Downarrow A}\ \text{synch}$$

$$\frac{\vdash \Theta, P \Downarrow P}{\vdash \Theta, P \Uparrow}\ \text{focus} \qquad \frac{\forall \Delta, (\Delta \Vdash^- A) \Rightarrow (\vdash \Theta, \Delta \Uparrow)}{\vdash \Theta \Uparrow A}\ \text{asynch}$$

Figure 16: Big-step LKF, v1

Remark 41

Sequents of the form $\vdash \Theta \Downarrow N$ and sequents of the form $\vdash \Theta \Uparrow P$ are never present in the premisses of the rules. Such sequents can only appear as the very conclusion of a whole proof-tree.

Hence, we can equivalently present the big-step LKF system as the system of Fig. 17, and declare $\vdash \Theta \Uparrow P$ as syntactic sugar for $\vdash \Theta, P \Uparrow$, and $\vdash \Theta \Downarrow N$ as syntactic sugar for $\vdash \Theta \Uparrow N$. ※

$$\frac{\Gamma \Vdash^+ P \qquad \forall N \in \Gamma, \vdash \Theta \Uparrow N \qquad \forall a \in \Gamma, \quad a^\perp \in \Theta}{\vdash \Theta \Downarrow P}$$

$$\frac{\vdash \Theta, P \Downarrow P}{\vdash \Theta, P \Uparrow} \qquad \frac{\forall \Delta, (\Delta \Vdash^- N) \Rightarrow (\vdash \Theta, \Delta \Uparrow)}{\vdash \Theta \Uparrow N}$$

Figure 17: Big-step LKF, v2

Now we should notice a complete symmetry, and therefore some redundancy, in the two decomposition relations that we have defined:

REMARK 42 $\Gamma \Vdash^+ P$ if and only if $\Gamma^\perp \Vdash^- P^\perp$. ※

Hence, we can define in Fig. 18 a simplified version of big-step LKF, where this redundancy is eliminated.

$$\frac{\Gamma \Vdash P \qquad \forall N \in \Gamma,\ \vdash \Theta \Uparrow N \qquad \forall a \in \Gamma, \quad a^\perp \in \Theta}{\vdash \Theta \Downarrow P}$$

$$\frac{\vdash \Theta, P \Downarrow P}{\vdash \Theta, P \Uparrow} \qquad \frac{\forall \Gamma, (\Gamma \Vdash N^\perp) \Rightarrow (\vdash \Theta, \Gamma^\perp \Uparrow)}{\vdash \Theta \Uparrow N}$$

where $\Gamma \Vdash P$ is $\Gamma \Vdash^+ P$.

Figure 18: Big-step LKF, v3

Now notice that the rules for negative connectives are never used in the system! Due to the duality in the syntax, given by the involutive negation, we should be able to remove negative connectives altogether. We just need to introduce a marker in the syntax of a formula, to denote every change of polarity.

Let us write $\neg$ for this marker.

DEFINITION 47 (Syntax with positive connectives only)

Formulae are now defined by the following syntax:

$$\begin{array}{rcl} P & ::= & a \,|\, A_1 \wedge^+ A_2 \,|\, A_1 \vee^+ A_2 \\ A & ::= & P \,|\, \neg P \end{array}$$

with the following involutive negation:

$$\begin{array}{rcl} P^\perp & := & \neg P \\ (\neg P)^\perp & := & P \end{array}$$

※

REMARK 43 The previous grammar can be encoded into that one:

$$\begin{array}{rcl} \overline{a} & := & a \\ \overline{A \wedge^+ B} & := & \overline{A} \wedge^+ \overline{B} \\ \overline{A \vee^+ B} & := & \overline{A} \vee^+ \overline{B} \end{array} \qquad \overline{N} := \neg(\overline{N^\perp})$$

※

In Fig. 19 we reformulate big-step LKF with this syntax for formulae.

$$\frac{\Gamma \Vdash P \qquad \forall \neg P' \in \Gamma,\ \vdash \Theta \Uparrow \neg P' \qquad \forall a \in \Gamma, \neg a \in \Theta}{\vdash \Theta \Downarrow P}$$

$$\frac{\vdash \Theta, P \Downarrow P}{\vdash \Theta, P \Uparrow} \qquad \frac{\forall \Gamma, (\Gamma \Vdash P) \Rightarrow (\vdash \Theta, \Gamma^\perp \Uparrow)}{\vdash \Theta \Uparrow \neg P}$$

where $\Gamma \Vdash P$ is $\Gamma \Vdash^+ P$.

Figure 19: Big-step LKF, v4

Finally, we notice that it is more natural to write Γ on the left-hand side of a sequent:

Definition 48 (Big-step LKF, v5)

The big-step LKF system v5 is given in Fig. 20, where Γ is a set of atoms a or formulae of the form $\neg P$ and Θ is a set of negated atoms $\neg a$ or formulae of the form P. ※

$$\frac{\Gamma \Vdash P \qquad \forall \neg P' \in \Gamma,\ \Gamma_0 \vdash \Uparrow \neg P' \qquad \forall a \in \Gamma,\ a \in \Gamma_0}{\Gamma_0 \vdash \Downarrow P}$$

$$\frac{\Gamma_0, \neg P \vdash \Downarrow P}{\Gamma_0, \neg P \vdash \Uparrow} \qquad \frac{\forall \Gamma, (\Gamma \Vdash P) \Rightarrow (\Gamma_0, \Gamma \vdash \Uparrow)}{\Gamma_0 \vdash \Uparrow \neg P}$$

where $\Gamma \Vdash P$ is $\Gamma \Vdash^+ P$.

Figure 20: Big-step LKF, v5

The lesson to be remembered from this formulation of big-step LKF, is that (the big-step version of) asynchronous rules happens to **coincide** with a rule inferred from (the big-step version of) synchronous rules. This will make cut-elimination work, and it formalises (at least in classical logic) the concept known in philosophical logic as *harmony* [Ten78, Rea00, Rea10] (expressed originally between the introduction rules and elimination rules of Natural Deduction, or between left-introduction rules and right-introduction rules of Sequent Calculus).

Now we can consider that both synchronous and asynchronous rules are defined primitively, and notice the somewhat "miraculous" coincidence, or we can adopt the view that only synchronous rules are defined primitively; asynchronous phases then work by duality from the way synchronous phases work.

In other words,

- positive connectives are "defined" by their introduction rules;
- negative connectives are "defined" by duality, from the introduction rules of their positive duals.

We may not even need to bother representing their rules.

In this view, and via the Curry-Howard correspondence, we should define how to inhabit a type $A \to B$ with (proof-)terms, from the way we inhabit A with terms and B with continuations (with $\to$ being a negative connective). Writing $\lambda x.M$ with a variable $x : A$ and a body $M : B$, would then only be a mere representation for (or a mere even implementation of) an inhabitant of $A \to B$ that pre-exists the syntactical notation.

This is of course expressed *semantically* in orthogonality models (say in Definitions 28, 31 and 36) by the fact that we first define an interpretation for positive formulae (mentioning the syntax of their basic inhabitants, such as the construct $t ::e$), and in a second step we define the interpretation of negative formulae simply as the orthogonal of the interpretation of their dual formula. The defininition for negative formulae does not even mention the syntax of their inhabitants (such as $\lambda x.M$), but if we have a syntax for them, they "happen to live" (somewhat miraculously) in the interpretation.

We now formalise a way to express this *syntactically*, as a proof-term calculus for big-step LKF.

3.2.3 Functional interpretation as pattern-matching

Earlier we wrote that "the proofs of negatives *must interact well with* the proofs of the positive dual". The intuition we formalise is that

- the proofs of a positive connective (i.e. of some $\Gamma_0 \vdash \Downarrow P$) are some data that can be *pattern-matched*;
- the proofs of a negative connective (i.e. of some $\Gamma_0 \vdash \Uparrow \neg P$) are *functions* that consume data by *pattern-matching*.

The fact that the proofs of a negative are determined by duality from the proofs of the positive dual, is reflected by the fact that the shape of a pattern-matching function is indeed completely determined by the data-type of its argument.

So the Curry-Howard interpretation of big-step LKF is an abstract system of pattern-matching.

The "proof-terms" for the decomposition of (positive) connectives are *patterns*. For instance for the connectives $\wedge^+, \vee^+$:

DEFINITION 49 (Patterns for $\wedge^+, \vee^+$) Patterns are defined by the following syntax:

$$p ::= x^+ \,|\, x^- \,|\, (p_1, p_2) \,|\, \mathsf{inj}_i(p)$$

Their typing rules are presented in Fig. 21. ※

$$\frac{}{x^- : \neg P \Vdash x^- : \neg P} \qquad \frac{}{x^+ : a \Vdash x^+ : a}$$

$$\frac{\Gamma_1 \Vdash p_1 : A_1 \qquad \Gamma_2 \Vdash p_2 : A_2}{\Gamma_1, \Gamma_2 \Vdash (p_1, p_2) : A_1 \wedge^+ A_2} \qquad \frac{\Gamma \Vdash p : A_i}{\Gamma \Vdash \mathsf{inj}_i(p) : A_1 \vee^+ A_2}$$

Figure 21: Decomposition with patterns

We now give the proof-terms for big-step LKF:

DEFINITION 50 (Pattern-matching calculus)

Let Pat be a set of elements called *patterns*, and denoted p, p', ...

The syntax of proof-terms is given by the following grammar:

Positive terms	t^+	::=	$p.\sigma$
Negative terms	t^-	::=	f
Commands	c	::=	$\langle x^- \mid t^+ \rangle \,|\, \langle f \mid t^+ \rangle$

where

- σ is a substitution from negative variables such as x^- to negative terms, and from positive variables such as x^+ to positive terms;
- f is a function from patterns to commands.

Let $\mathbb{A}$ and $\mathbb{M}$ be two sets of elements called *atoms* and *molecules* and denoted p and P, respectively.

Let *typing contexts* be functions mapping negative variables to molecules (written $x^- : \neg P$) and positive variables to atoms (written $x^+ : a$).

Let $\Gamma \Vdash p : P$ be a typing relation where p is a pattern, P is a molecule, and Γ is a typing context.

The typing rules for proof-terms are presented in Fig. 22.
There is just one cut-elimination rule:

$$\text{(pat-match)} \qquad \langle f \mid p.\sigma \rangle \longrightarrow (f(p))\sigma$$

where $c\sigma$ denotes the application of substitution σ to the command c. ※

$$\frac{\Gamma \Vdash p : P \qquad \forall (x^- : \neg P) \in \Gamma, \quad \Gamma_0 \vdash \Uparrow \sigma(x^-) : \neg P \qquad \forall (x^+ : a) \in \Gamma, \quad (\sigma(x^+) : a) \in \Gamma_0}{\Gamma_0 \vdash \Downarrow p.\sigma : P}$$

$$\frac{\forall \Gamma, (\Gamma \Vdash p : P) \Rightarrow f(p) : (\Gamma_0, \Gamma \vdash \Uparrow)}{\Gamma_0 \vdash \Uparrow f : \neg P}$$

$$\frac{\Gamma_0, x^- : \neg P \vdash \Downarrow p.\sigma : P}{\langle x^- \mid p.\sigma \rangle : (\Gamma_0, x^- : \neg P \vdash \Uparrow)} \qquad \frac{\Gamma_0 \vdash \Uparrow f : \neg P \qquad \Gamma_0 \vdash \Downarrow t^+ : P}{\left\langle f \mid t^+ \right\rangle : (\Gamma_0 \vdash \Uparrow)}$$

where Γ_0, Γ, ... are typing contexts.

Figure 22: Typing for the pattern-matching calculus

The one cut-elimination rule is the very standard mechanism of pattern-matching, with the command $\langle f \mid t^+ \rangle$ representing what we could informally write as:
"`match` t^+ `with` $\underbrace{\ldots \mapsto \ldots}_{f}$"

Remark 44

1. Notice how negative terms are not really terms, but functions of the meta-level (or meta-level functions that are reified in the term syntax); this is a higher-order definition, and we do not give any concret syntax for such functions.

   Strictly speaking, our definition depends on the notion of function space that we take for the definition of negative terms (We could for instance restrict it to computable functions, but so far we do not specify such things).

   Also, with such a definition, it may not be clear exactly what the contextual closure of the rule (pat-match) is. By $\longrightarrow_{(\mathsf{pat-match})}$ we therefore denote the reduction relation where (pat-match) is applied at the top-level of a given command (no contextual closure).
2. Also notice how we emphasised that the definition of the proof-term calculus is *independent* from the syntax of patterns and the typing system for them.

   Definition 49 and Fig. 21 give one example (where atoms are positive atomic formulae and molecules are positive formulae).

   But the construction of the Curry-Howard interpretation for big-step focussing is modular in those notions.

   This is a gain of genericity / abstraction that we will further develop in the next Chapters.

※

Theorem 45 (Subject Reduction)

If $c : (\Gamma \vdash \Uparrow)$ and $c \longrightarrow_{(\mathsf{pat-match})} c'$ then $c' : (\Gamma \vdash \Uparrow)$. ※

Reviewing the various properties that are desirable for an instance of the Curry-Howard correspondence, we find that Progress (in this case, cut-elimination), depends on how we may reduce functions (so far, $\longrightarrow_{(\mathsf{pat-match})}$ only applies at the root); Confluence does not make sense here because, until we define how to reduce functions, there is at most one redex to reduce; and for Normalisation, we need to explore models (e.g. orthogonality models) of such a calculus.

Conclusion

The study of orthogonality models for the pattern-matching calculus should be particularly interesting:

In [MM09], Munch-Maccagnoni already explored the construction of orthogonality models for polarised System L with an emphasis on focussing properties. Big-step LKF and its underlying pattern-matching calculus seems to be an even more appropriate framework to look at the connection between focussing and orthogonality models, since this framework reflects at the syntactical level what orthogonality models describe at the semantical level, namely the fact that we first declare what the "inhabitants of positive formulae" are, and then we define the "inhabitants of negative formulae" by duality as those inhabitants that "interact well with" the inhabitants of the dual (positive) formula. In case of an orthogonality model, to "interact well with" means to "be orthogonal to"; in the case of pattern-matching, it means to "be able to consume". To be more precise:

- Inhabitants of positive types have *structure*: in an orthogonality model we need an algebraic structure to interpret positive constructs such as $\_::\_$ or $\mathsf{inj}\_(\_)$; in big-step LKF, these inhabitants come as the combination of a pattern (e.g. $\_::\_$ or $\mathsf{inj}\_(\_)$) and a substitution that fills its holes.
- Inhabitants of negative formulae may lack any structure, but they come with a *behaviour*: in an orthogonality model, they can range over any abstract set for which the orthogonality relation with positive inhabitants is defined; in big-step LKF, they range over any abstract set of functions (we do not specify which) that can consume patterns.

So, in order to formalise the connections that are informally described above, the second part of this dissertation explores orthogonality models for big-step focussing systems. We shall strip anything that is not essential off the constructions we make, systematically seeking the greatest generality, and aiming at the cores of orthogonality models and focussing systems. Doing so reveals the essential difference between realisability and typing:

- in realisability, checking whether a given negative inhabitant "interacts well with" an arbitrary inhabitant of a positive formula, requires the computation of an interaction that explores the positive inhabitant's structure to an **arbitrary depth** (as nothing restricts the criterion given by orthogonality);
- in typing, checking whether a given negative inhabitant "interacts well with" an arbitrary inhabitant of a positive formula, only requires the computation of an interaction that explores the positive inhabitant's structure to a **bounded depth** (as the negative inhabitant is a function that performs a case analysis on the positive inhabitant's top-level pattern and the interaction has to uniformly treat the rest of the inhabitant's structure).

In case each positive formula comes with a finite number of patterns for it, the above distinction is what makes typing decidable and realisability undecidable (in general).

Bibliography

[ADH+12] Z. M. Ariola, P. Downen, H. Herbelin, K. Nakata, and A. Saurin. Classical call-by-need sequent calculi: The unity of semantic artifacts. In T. Schrijvers and P. Thiemann, editors, *Proc. of the 11th Int. Symp. Functional and Logic Programming (FLOPS'12)*, volume 7294 of *LNCS*, pages 32–46. Springer-Verlag, 2012. 39

[AF97] Z. M. Ariola and M. Felleisen. The call-by-need lambda calculus. *J. Funct. Programming*, 7(3):265–301, 1997 39

[AH03] Z. M. Ariola and H. Herbelin. Minimal classical logic and control operators. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proc. of the 30th Intern. Col. on Automata, Languages and Programming (ICALP)*, volume 2719 of *LNCS*, pages 871–885. Springer-Verlag, 2003. 12, 17

[AH08] Z. M. Ariola and H. Herbelin. Control reduction theories: the benefit of structural substitution. *J. Funct. Programming*, 18(3):373–419, 2008. 17

[AHS09] Z. M. Ariola, H. Herbelin, and A. Sabry. A type-theoretic foundation of delimited continuations. *Higher-Order and Symbolic Computation*, 22(3):233–273, 2009. online from 2007. 39

[AHS11] Z. M. Ariola, H. Herbelin, and A. Saurin. Classical call-by-need and duality. In C. L. Ong, editor, *Proc. of the 10th Int. Conf. on Typed Lambda Calculus and Applications (TLCA'11)*, volume 6690 of *LNCS*, pages 27–44. Springer-Verlag, 2011. 39

[And92] J. M. Andreoli. Logic programming with focusing proofs in linear logic. *J. Logic Comput.*, 2(3):297–347, 1992. 61, 63

[AP89] J.-M. Andreoli and R. Pareschi. Logic programming with sequent systems, a linear logic approach. In P. Schroeder-Heister, editor, *Proc. of the Int. Work. on Extensions of Logic Programming*, LNCS, pages 1–30. Springer-Verlag, 1989. 61, 63

[Bar84] H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. Elsevier, 1984. Second edition. 6, 7, 10, 51, 57

[BB96] F. Barbanera and S. Berardi. A symmetric lambda-calculus for classical program extraction. *Inform. and Comput.*, 125(2):103–117, 1996. 19, 45

[BL11] A. Bernadet and S. Lengrand. Filter models: non-idempotent intersection types, orthogonality and polymorphism. In M. Bezem, editor, *Proc. of the 20th Annual Conf. of the European Association for Computer Science Logic (CSL'11)*, LIPIcs. Schloss Dagstuhl LCI, 2011. 41, 42, 44, 45

[CF58] H. B. Curry and R. Feys. *Combinatory Logic*, volume I. North-Holland, 1958.3, 4, 8

[CH00] P.-L. Curien and H. Herbelin. The duality of computation. In *Proc. of the* 5^{th} *ACM SIGPLAN Int. Conf. on Functional Programming (ICFP'00)*, pages 233–243. ACM Press, 2000. 19, 34, 63

[Chu41] A. Church. *The Calculi of Lambda Conversion*. Princeton University Press, 1941.6

[CMM10] P.-L. Curien and G. Munch-Maccagnoni. The duality of computation under focus. In C. S. Calude and V. Sassone, editors, *Theoretical Computer Science*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 165–181. Springer-Verlag, 2010. 47, 61

[Cro04] T. Crolard. A formulae-as-types interpretation of subtractive logic. *J. Logic Comput.*, 14(4):529–570, 2004. 36, 39, 60

[CS09] J. R. B. Cockett and L. Santocanale. On the word problem for ΣΠ-categories, and the properties of two-way communication. In E. Grädel and R. Kahle, editors, *Proc. of the 18th Annual Conf. of the European Association for Computer Science Logic (CSL'09)*, volume 5771 of *LNCS*, pages 194–208. Springer-Verlag, 2009. 27

[DF89] O. Danvy and A. Filinski. A functional abstraction of typed contexts. Technical Report 89/12, DIKU, University of Cophenhagen, 1989. 39

[DF90] O. Danvy and A. Filinski. Abstracting control. In *Proc. of the 1990 ACM Conf. on LISP and functional programming*, pages 151–160. ACM Press, 1990. 39

[DJS95] V. Danos, J.-B. Joinet, and H. Schellinx. LKQ and LKT: sequent calculi for second order logic based upon dual linear decompositions of classical implication. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Proc. of the Work. on Advances in Linear Logic*, volume 222 of *London Math. Soc. Lecture Note Ser.*, pages 211–224. Cambridge University Press, 1995. 34, 61, 63

[DJS97] V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic: Linear logic. *J. of Symbolic Logic*, 62(3):755–807, 1997. 34, 61, 63

[DK00] V. Danos and J.-L. Krivine. Disjunctive tautologies as synchronisation schemes. In P. Clote and H. Schwichtenberg, editors, *Proc. of the 9th Annual Conf. of the European Association for Computer Science Logic (CSL'00)*, volume 1862 of *LNCS*, pages 292–301. Springer-Verlag, 2000. 41, 43

[DP04] K. Došen and Z. Petrić. *Proof-theoretical Coherence*. King's College Publications, 2004. 27

[Far13] M. Farooque. *Automated reasoning techniques as proof-search in sequent calculus*. PhD thesis, Ecole Polytechnique, 2013. 63

[Fel87] M. Felleisen. *The Calculi of λ-v-CS Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages*. PhD thesis, Department of Computer Science, Indiana University, Bloomington, Indiana, 1987. 14

[Fil89] A. Filinski. Declarative continuations and categorical duality. Master's thesis, DIKU, Computer Science Department, University of Copenhagen, 1989. DIKU Rapport 89/11. 19

[Fis72] M. J. Fischer. Lambda calculus schemata. In *Proc. of the ACM Conf. on Proving Assertions about Programs*, pages 104–109. SIGPLAN Notices, Vol. 7, No 1 and SIGACT News, No 14, 1972. 31, 34, 38

[FP06] C. Fürmann and D. Pym. Order-enriched categorical models of the classical sequent calculus. *J. Pure Appl. Algebra*, 204(1):21–78, 2006. 27

[FR94] A. Fleury and C. Retoré. The mix rule. *Math. Structures in Comput. Sci.*, 4(2):273–285, 1994. 25

[Fre79] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Verlag von Louis Nebert, 1879 5

[Gen35] G. Gentzen. Investigations into logical deduction. In *Gentzen collected works*, pages 68–131. Ed M. E. Szabo, North Holland, (1969), 1935. 6, 12, 19, 20, 24, 63

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'état, Université Paris 7, 1972. 41, 42, 45, 46

[Gir87] J.-Y. Girard. Linear logic. *Theoret. Comput. Sci.*, 50(1):1–101, 1987. 41, 58, 63, 64

[Gir91] J.-Y. Girard. A new constructive logic: Classical logic. *Math. Structures in Comput. Sci.*, 1(3):255–296, 1991. 63

[Gri90] T. G. Griffin. A formulae-as-type notion of control. In P. Hudak, editor, *17th Annual ACM Symp. on Principles of Programming Languages (POPL'90)*, pages 47–58. ACM Press, 1990. 3, 14

[GTL89] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoret. Comput. Sci.* Cambridge University Press, 1989. 12

[Her05] H. Herbelin. *C'est maintenant qu'on calcule: au coeur de la dualité*. Thèse d'habilitation à diriger des recherches, Université Paris 11, 2005. 19

[HG08] H. Herbelin and S. Ghilezan. An approach to call-by-name delimited continuations. In G. C. Necula and P. Wadler, editors, *Proc. of the 35th Annual ACM Symp. on Principles of Programming Languages (POPL'08)*, pages 383–394. ACM Press, 2008. 39

[Hil28] D. Hilbert. Die Grundlagen der Mathematik. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 6(1):65–85, 1928. 5

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980. Reprint of a manuscript written 1969. 3, 6, 8

[HS97] M. Hofmann and T. Streicher. Continuation models are universal for $\lambda\mu$-calculus. In *Proc. of the 12th Annual IEEE Symp. on Logic in Computer Science*, pages 387–397. IEEE Computer Society Press, 1997. 31, 34, 38, 39

[HZ09] H. Herbelin and S. Zimmermann. An operational account of call-by-value minimal and classical λ-calculus in "natural deduction" form. In P. Curien, editor, *Proc. of the 9th Int. Conf. on Typed Lambda Calculus and Applications (TLCA'09)*, volume 5608 of *LNCS*, pages 142–156. Springer-Verlag, 2009. 39

[Joh36] I. Johansson. Der Minimalkalkül, ein reduzierter intuitionistischer Formalismus. *Compositio Math.*, 4:119–136, 1936. 5

[Kri71] J.-L. Krivine. *Introduction to axiomatic set theory*. Dordrecht, Reidel, 1971. 1

[Kri01] J.-L. Krivine. Typed lambda-calculus in classical Zermelo-Frænkel set theory. *Arch. Math. Log.*, 40(3):189–205, 2001. 41, 43

[Lam07] F. Lamarche. Exploring the Gap between Linear and Classical Logic. *Theory and Applications of Categories*, 18(17):473–535, 2007 27

[Lau02] O. Laurent. *Etude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, 2002. 63

[Len03] S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. volume 86(4) of *ENTCS*. Elsevier, 2003. Revision from the 3rd Int. Work. on Reduction Strategies in Rewriting and Programming (WRS'03). 19, 24

[Len06] S. Lengrand. *Normalisation & Equivalence in Proof Theory & Type Theory*. PhD thesis, Université Paris 7 & University of St Andrews, 2006. 1, 30

[LM08] S. Lengrand and A. Miquel. Classical F_ω, orthogonality and symmetric candidates. *Ann. Pure Appl. Logic*, 153:3–20, 2008. 41, 48, 50

[LM09] C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoret. Comput. Sci.*, 410(46):4747–4768, 2009. 59, 61, 64, 65

[LQdF05] O. Laurent, M. Quatrini, and L. T. de Falco. Polarized and focalized linear and classical proofs. *Ann. Pure Appl. Logic*, 134(2-3):217–264, 2005. 63

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge University Press, 1986. 12

[LS05] F. Lamarche and L. Straßburger. Constructing free boolean categories. In P. Panangaden, editor, *Proc. of the 20th Annual IEEE Symp. on Logic in Computer Science*, pages 209–218. IEEE Computer Society Press, 2005. 27

[Miq09] A. Miquel. Relating classical realizability and negative translation for existential witness extraction. In P. Curien, editor, *Proc. of the 9th Int. Conf. on Typed Lambda Calculus and Applications (TLCA'09)*, volume 5608 of *LNCS*, pages 188–202. Springer-Verlag, 2009. 42, 50, 54

[Miq11] A. Miquel. Existential witness extraction in classical realizability and via a negative translation. *Logic. Methods Comput. Science*, 7(2), 2011. 42, 50, 54

[ML82] P. Martin-Löf. Constructive mathematics and computer programming. In *Proc. of the Sixth Int. Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175. North-Holland, 1982. 3

[ML84] P. Martin-Löf. *Intuitionistic Type Theory*. Number 1 in Studies in Proof Theory, Lecture Notes. Bibliopolis, 1984. 3

[MM09] G. Munch-Maccagnoni. Focalisation and classical realisability. In E. Grädel and R. Kahle, editors, *Proc. of the 18th Annual Conf. of the European Association for Computer Science Logic (CSL'09)*, volume 5771 of *LNCS*, pages 409–423. Springer-Verlag, 2009. 34, 35, 41, 44, 47, 61, 63, 74

[MM13] G. Munch-Maccagnoni. *Syntax and Models of a Non-Associative Composition of Programs and Proofs*. PhD thesis, Université Paris Diderot – Paris 7, 2013. 47, 61

[Mog89] E. Moggi. Computational lambda-calculus and monads. In *Proc. of the 4th Annual IEEE Symp. on Logic in Computer Science*, pages 14–23. IEEE Computer Society Press, 1989. 27, 30

[MV05] P.-A. Melliès and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In P. Panangaden, editor, *Proc. of the 20th Annual IEEE Symp. on Logic in Computer Science*, pages 82–91. IEEE Computer Society Press, 2005. 41

[Par92] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *Proc. of the Int. Conf. on Logic Programming and Automated Reasoning (LPAR'92)*, volume 624 of *LNCS*, pages 190–201. Springer-Verlag, 1992. 15, 20

[Par97] M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *J. of Symbolic Logic*, 62(4):1461–1479, 1997. 41, 45, 46

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoret. Comput. Sci.*, 1:125–159, 1975. 27, 28, 29

[Pol04] E. Polonovski. Strong normalization of $\lambda\mu\tilde{\mu}$-calculus with explicit substitutions. In I. Walukiewicz, editor, *Proc. of the 7th Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS'04)*, volume 2987 of *LNCS*, pages 423–437. Springer-Verlag, 2004. 24

[Rea00] S. Read. Harmony and autonomy in classical logic. *J. of Philosophical Logic*, 29(2):123–154, 2000. 71

[Rea10] S. Read. General-elimination harmony and the meaning of the logical constants. *J. of Philosophical Logic*, 39(5):557–576, 2010. 71

[Rey72] J. C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proc. of the ACM annual Conf.* , pages 717–740, 1972. 14, 29, 34, 41

[Roc05] J. Rocheteau. lambda-μ-calculus and duality: Call-by-name and call-by-value. In J. Giesl, editor, *Proc. of the 16th Int. Conf. on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *LNCS*, pages 204–218. Springer-Verlag, 2005. 19

[Sau05] A. Saurin. Separation with streams in the lambdaμ-calculus. In P. Panangaden, editor, *Proc. of the 20th Annual IEEE Symp. on Logic in Computer Science*, pages 356–365. IEEE Computer Society Press, 2005. 39

[Sau08] A. Saurin. On the relations between the syntactic theories of lambda-mu-calculi. In *Proc. of the 17th Annual Conf. of the European Association for Computer Science Logic (CSL'08)*, volume 5213 of *LNCS*, pages 154–168. Springer-Verlag, 2008. 39

[Sau10a] A. Saurin. A hierarchy for delimited continuations in call-by-name. In C. L. Ong, editor, *Proc. of the 13th Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS'10)*, volume 6014 of *LNCS*, pages 374–388. Springer-Verlag, 2010. 39

[Sau10b] A. Saurin. Standardization and böhm trees for lambdaμ-calculus. In M. Blume, N. Kobayashi, and G. Vidal, editors, *Proc. of the 10th Int. Symp. Functional and Logic Programming (FLOPS'10)*, volume 6009 of *LNCS*, pages 134–149. Springer-Verlag, 2010. 39

[Sau10c] A. Saurin. Typing streams in the lambdaμ-calculus. *ACM Trans. on Comput. Logic*, 11(4), 2010. 39

[Sau12] A. Saurin. Böhm theorem and böhm trees for the $\lambda\mu$-calculus. *Theoret. Comput. Sci.*, 435:106–138, 2012. 39

[Sel01] P. Selinger. Control categories and duality: on the categorical semantics of the $\lambda\mu$-calculus. *Math. Structures in Comput. Sci.*, 11(2):207–260, 2001.32, 39, 62, 63

[SR98] T. Streicher and B. Reus. Classical logic, continuation semantics and abstract machines. *J. Funct. Programming*, 8(6):543–572, 1998 39

[Str11] L. Straßburger. *Towards a Theory of Proofs of Classical Logic*. Habilitation thesis, Université Denis Diderot – Paris 7, 2011. 12, 27

[SU06] M. H. B. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Studies in Logic and the Foundations of Mathematics. Elsevier, 2006. 4

[SW00] C. Strachey and C. P. Wadsworth. Continuations: A mathematical semantics for handling fulljumps. *Higher-Order and Symbolic Computation*, 13:135–152, 2000. 14

[Tai67] W. W. Tait. Intensional interpretations of functionals of finite type I. *J. of Symbolic Logic*, 32:198–212, 1967. 41

[Tai75] W. W. Tait. A realizability interpretation of the theory of species. In *Logic Colloquium*, volume 453 of *LNM*, pages 240–251. Springer-Verlag, 1975. 41, 42, 45, 46

[Ten78] N. Tennant. *Natural logic*. Edinburgh University Press, 1978 71

[Ter03] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoret. Comput. Sci.* Cambridge University Press, 2003. 1

[TS00] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 2000. 2, 20, 24, 25, 56

[Urb00] C. Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, 2000. 19, 20

[Wad03] P. Wadler. Call-by-value is dual to call-by-name. In *Proc. of the 8th ACM SIGPLAN Int. Conf. on Functional programming (ICFP'03)*, volume 38(9), pages 189–201. ACM Press, 2003. 19, 21, 34, 35, 37, 60

[Zei08a] N. Zeilberger. Focusing and higher-order abstract syntax. In G. C. Necula and P. Wadler, editors, *Proc. of the 35th Annual ACM Symp. on Principles of Programming Languages (POPL'08)*, pages 359–369. ACM Press, 2008. 64, 67, 69

[Zei08b] N. Zeilberger. On the unity of duality. *Ann. Pure Appl. Logic*, 153(1-3):66–96, 2008. 64, 67, 69

[Zei10] N. Zeilberger. Polarity and the logic of delimited continuations. In J.-P. Jouannaud, editor, *Proc. of the 25th Annual IEEE Symp. on Logic in Computer Science*, pages 219–227. IEEE Computer Society Press, 2010. 67, 69

List of Figures

Appendix A

Basic definitions for categories

Definition 51 (Category) A *category* is the combination of

- a class of elements called *objects*, denoted A, B,...
- for every pair of objects A and B, a class $\mathsf{hom}(A, B)$ of elements called *morphisms from A to B*; the expression $f\colon A \longrightarrow B$ denotes that f is a morphism from A to B;
- for every object A, a morphism Id_A called *identity*;
- for every objects A, B, C, a binary operation called *composition* mapping every $f\colon A \longrightarrow B$ and $g\colon B \longrightarrow C$ to a morphism $f \cdot g\colon A \longrightarrow C$

such that the following properties holds

- composition is associative $((f \cdot g) \cdot h = f \cdot (g \cdot h))$
- identities are units for composition $(\mathsf{Id}_A \cdot f = f \cdot \mathsf{Id}_A = f)$.

Given a category, we often use *diagrams* to represent a collection of objects -represented as vertices- and morphisms -represented as labelled arrows between vertices. Using morphism composition, each path between any two given vertices unambiguously represents a morphism. A diagram *commutes* when for each pair of vertices A and B, all paths from A to B represent equal morphisms.

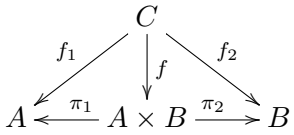
Two objects A and B are *isomorphic* when there are two morphisms $f\colon A \longrightarrow B$ and $g\colon B \longrightarrow A$ such that $f \cdot g = \mathsf{Id}_A$ and $g \cdot f = \mathsf{Id}_B$. ※

Standard examples of categories are: the categories of sets and functions (objects are sets and morphisms from A to B are functions from A to B), the category of sets and relations (objects are sets and morphisms from A to B are relations from A to B), etc. Groups form a particular kind of categories (where there is only one object, and elements of the group are the morphisms from that object to itself). Partially ordered sets form another particular kind of categories (where there is at most one morphism between any two objects), etc.

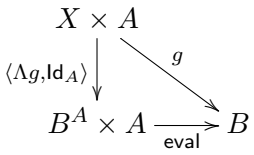
DEFINITION 52 (Cartesian Closed Category)

A *Cartesian Closed Category* (CCC), is a category such that

- there is an object, denoted 1 and called *terminal object*, such that, for each object A, there is a unique morphism $1_A \colon A \longrightarrow 1$;
- for every two objects A and B, there is an object, denoted $A \times B$ and called the *product* of A and B, together with two morphisms $\pi_1 \colon A \times B \longrightarrow A$ and $\pi_2 \colon A \times B \longrightarrow B$, called the first and second *projections*, satisfying the following property:
  for every object C and morphisms $f_1 \colon C \longrightarrow A$ and $f_2 \colon C \longrightarrow B$, there is a unique $f \colon C \longrightarrow A \times B$, denoted $\langle f_1, f_2 \rangle$, such that the following diagram commutes



- for every two objects A and B, there is an object, denoted B^A and called the *exponential* of A and B, together with a morphism $\mathsf{eval} \colon B^A \times A \longrightarrow B$, satisfying the following property:
  for every object X and morphism $g \colon X \times A \longrightarrow B$ there is a unique $f \colon X \longrightarrow B^A$, denoted Λg, such that the following diagram commutes



We choose the convention that products are associative to the left, i.e. $(A \times B) \times C$ can be abbreviated as $A \times B \times C$. A family of morphisms $\pi_{i/n} \colon A_1 \times \cdots \times A_n \longrightarrow A_i$, for $1 \leq i \leq n$, can be defined by composing the two projections in the obvious way:

$$\begin{array}{lll} \pi_{1/1} & := & \mathsf{Id}_{A_1} \\ \pi_{n/n} & := & \pi_2 \qquad\qquad \text{when } 1 < n \\ \pi_{p/n} & := & \pi_1 \cdot \pi_{p/n-1} \quad \text{when } p < n \end{array}$$

※

REMARK 46 One can quickly check that the terminal object, products and exponentials are unique up to isomorphism (i.e. two objects satisfying the property of the terminal object, or the product / exponential object for a given A and B, are isomorphic); hence the notations 1, $A \times B$, B^A. ※