A Modal Lambda Calculus with Iteration and Case Constructs

Pierre Leleu

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Modal Lambda Calculus with Iteration and Case Constructs

Pierre Leleu

N° 3322

Décembre 1997

THÈME 2

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

A Modal Lambda Calculus with Iteration and Case Constructs

Pierre Leleu

Thème 2 — Génie logiciel
et calcul symbolique
Projet Croap

Rapport de recherche n° 3322 — Décembre 1997 — 42 pages

Abstract: An extension of the simply-typed λ-calculus allowing iteration and case reasoning over terms defined by means of higher order abstract syntax has recently been introduced by Joëlle Despeyroux, Frank Pfenning and Carsten Schürmann. This thorny mixing is achieved thanks to the help of the operator '$\Box$' of modal logic IS4. Here we give a new presentation of their system, with reduction rules, instead of evaluation judgments, that compute the canonical forms of terms. Our presentation is based on a modal λ-calculus that is better from the user's point of view, is more concise and we do not impose a particular strategy of reduction during the computation. Our system enjoys the decidability of typability, soundness of typed reduction with respect to typing rules, the Church-Rosser and strong normalization properties. Finally it is a conservative extension of the simply-typed λ-calculus.

Key-words: HIGHER-ORDER ABSTRACT SYNTAX, INDUCTION, RECURSION, MODAL LOGIC, LOGICAL FRAMEWORK, TYPE THEORY

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 93 65 77 77 - International : +33 4 93 65 77 77 — Fax : 04 93 65 77 65 - International : +33 4 93 65 77 65
à partir du 01/01/1998
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

Un Lambda Calcul Modal Muni d'Opérateurs d'Itération et de Raisonnement par Cas

Résumé : Une extension du λ-calcul simplement typé permettant l'itération et le raisonnement par cas sur des termes définis au moyen de la syntaxe abstraite d'ordre supérieur a été présentée récemment par Joëlle Despeyroux, Frank Pfenning et Carsten Schürmann. Ce mélange délicat est rendu possible par l'intervention de l'opérateur "$\Box$" de la logique modale IS4. Nous donnons ici une nouvelle présentation de leur système, comportant des règles de réduction, au lieu de jugements d'évaluation, qui calculent les formes canoniques des termes. Notre présentation repose sur un λ-calcul modal meilleur pour l'utilisateur. Elle est plus concise et n'impose pas de stratégie particulière de réduction. Notre système vérifie les propriétés de décidabilité du typage, de correction de la réduction typée par rapport au typage, de Church-Rosser, de forte normalisation. Enfin, il est une extension conservative du λ-calcul simplement typé.

Mots-clés : SYNTAXE ABSTRAITE D'ORDRE SUPERIEUR, INDUCTION, RECURSION, LOGIQUE MODALE, THEORIES TYPEES

1 Introduction

Higher order abstract syntax ([PE88]) is a representation technique which proves to be useful when modelizing in a logical framework a language which involves bindings of variables. By borrowing the variables of the meta level to represent the variables of the object language, one does not need worrying about variable binding, substitution or renaming. These operations are managed at the meta level. Parametric and hypothetical judgments are also directly supported by the framework.

On the other hand, inductive definitions are frequent in mathematics and semantics of programming languages, and induction is an essential tool when developing proofs. Unfortunately it is well-known that a type defined by means of higher order abstract syntax cannot be defined as an inductive type in usual inductive type theories (like CCI [Wer94], [PM92], or Martin-Löf's Logical Framework [NPS90] for instance).

In a first step towards the resolution of this dilemma, Joëlle Despeyroux, Frank Pfenning and Carsten Schürmann have presented ([DPS97]) an extension of the simply-typed λ-calculus with recursive constructs (operators for iteration and case reasoning), which enables the use of higher order abstract syntax in an inductive type. To achieve that, they use the operator '$\Box$' of modal logic IS4 to distinguish the types '$A \to B$' of the functional terms well-typed in the simply-typed λ-calculus from the types '$\Box A \to B$' of the functional terms possibly containing recursive constructs.

In this paper, we present an alternative presentation of their system that we claim to be better in several aspects. We use the same mechanism as them to mix higher order abstract syntax and induction but our typing and reduction rules are quite different. Indeed there are several presentations of modal lambda calculus IS4 ([BdP96], [PW95], [DP96]). We have chosen the variant by Frank Pfenning and Hao-Chi Wong ([PW95]), which has context stacks instead of simple contexts. This peculiarity creates some difficulties in the metatheoretical study but the terms generated by the syntax are simpler than those of [DPS97] (no 'let' construction), and so this system is better from the user's point of view.

Moreover, instead of introducing an operational semantics which computes the canonical form (η-long normal form) using a given strategy, our system has reduction rules, which allow a certain indeterminism in the mechanism of reduction. We are able to benefit from classic proof techniques to show the important metatheoretic results: decidability of typability, soundness of typing with respect to typing rules, Church-Rosser property (CR), Strong Normalization property (SN) and conservativity of our system with respect to the simply-typed λ-calculus. The main problems we encountered in the proofs are due on the one hand to the use of functional types in the types of the recursive constructors, on the other hand to the use of η-expansion. To solve the problems due to η-expansion, we benefit from previous work done for the simply-typed λ-calculus ([JG95]) and for system F ([Gha96]).

In the second section of the paper, we introduce our version of the modal inductive system, its syntax, its typing and reduction rules. Then in the third section, we prove its essential properties (soundness of typing, CR, SN) from which we deduce that it is a conservative extension of the simply-typed lambda calculus. Finally, we discuss related works and outline future work.

2 The system

In this section, we present the syntax, the typing rules and the semantics of our system. First, we briefly recall our motivation.

2.1 Higher-Order Abstract Syntax

The mechanics of higher order abstract syntax (HOAS) has already been exposed in many places, for example in [HHP93]. Let us introduce here a simple example of representation using HOAS, that will prove useful later when we illustrate the mechanism of the reduction rules.

Let us assume that we work in the simply-typed λ-calculus with no extra equations, where we want to represent the untyped lambda terms. We introduce the type L of untyped lambda terms together with two constructors:

- $\texttt{lam} : (\texttt{L} \to \texttt{L}) \to \texttt{L}$
- $\texttt{app} : \texttt{L} \to \texttt{L} \to \texttt{L}$

It is well-known ([HHP93]) that the canonical forms (β-normal η-long) of type L are in one-to-one correspondence with the untyped lambda-terms and that this correspondence is compositional. For instance the term of type L $(\texttt{lam}\ \lambda\texttt{x} : \texttt{L}.(\texttt{app x x}))$ represents the untyped lambda-term $\lambda x.(x\ \ x)$.

Now, these constructors do not define an inductive type in usual inductive type theories like the Inductive Calculus of Constructions ([Wer94]) or the Extended Calculus of Constructions ([Luo94]) because of the leftmost occurence of L in the type of constructor lam. If we allowed this kind of inductive definition, we would be confronted with two serious problems. First, we would lose the one-to-one correspondence between the objects we represent and the canonical forms of type $L \to \cdots \to L$. For instance, if we have a Case construct (definition of a function by case over inductive terms), the term $(\texttt{lam}\ \lambda\texttt{x} : \texttt{L.Case x of}\ \ldots)$ does not represent any untyped lambda-term. Moreover we would lose the important property of strong normalization; more precisely we could write terms which would reduce to themselves. Our goal is to introduce a system which repairs these deficiencies.

Following [DPS97], we will use the modal operator '$\Box$' of modal logic IS4 to distinguish the types '$A \to B$' of the functional terms well-typed in the simply-typed λ-calculus from the types '$\Box A \to B$' of the functional terms possibly containing recursive contructs. For instance, in our system, a term such as '$\lambda x : L.\text{Case } x \text{ of } \ldots$' will have type $\Box L \to L$ whereas constructor 'lam' will have type $(L \to L) \to L$. Thus, our typing judgment will rule out undesirable terms such as '$(\text{lam}\ \ \lambda x : L.\text{Case } x \text{ of } \ldots)$'.

2.2 Syntax

The system we present is roughly the simply-typed lambda calculus extended by pairs, modality IS4 and induction. The addition of polymorphism and dependent types is left for future work.

2.2.1 Types

To describe the types of the system, we first consider a countable collection of constant types L_j ($j \in \mathbb{N}$), called the *ground* types. In our approach, they play the role of *inductive* types.

Definition 2.1 (Types)
The types *are inductively defined by:*

$$\mathit{Types} : T := L_j \mid T_1 \to T_2 \mid T_1 \times T_2 \mid \Box T$$

A type is said to be *pure* if it contains no '$\Box$' operator and no product. If it contains the operator '$\Box$', it is said to be *modal*.

2.2.2 Context stacks

Following the presentation of [PW95], we have *context stacks* instead of simple contexts. As usual a context Γ is defined as a list of unordered declarations $x : A$ (x is a variable and A is a type) where all the variables are distinct. A *context stack* Δ is an ordered list of contexts, separated by semi colons $\Gamma_1; \ldots; \Gamma_n$. '$\cdot$' denotes the empty context as well as the empty stack.

Notations A context stack is said to be *valid* if all the variables of the stack are distinct. We call *local* context of a stack $\Delta = \Gamma_1; \ldots; \Gamma_n$ the last context of the stack: Γ_n. The notation Δ, Γ, where Δ is a stack $\Gamma_1; \ldots; \Gamma_n$ and Γ is a context, is the stack $\Gamma_1; \ldots; \Gamma_n, \Gamma$. Similarly, the notation Δ, Δ', where Δ is the stack $\Gamma_1; \ldots; \Gamma_n$ and Δ' is the stack $\Gamma'_1; \ldots; \Gamma'_m$, is the stack $\Gamma_1; \ldots; \Gamma_n, \Gamma'_1; \ldots; \Gamma'_m$.

2.2.3 Terms

Before describing the set of the terms, we consider a finite collection of constant terms (the *constructors*) $C_{j,k}$, given with their pure type $C_{j,k} : \left(B_{j,k,1} \to \cdots \to B_{j,k,n_{j,k}}\right) \to L_j$, where each $B_{j,k,l}$ is a pure type. If $n_{j,k} = 0$, the type of $C_{j,k}$ is simply L_j.

Definition 2.2 (Terms)
The terms *are defined inductively by:*

$$\begin{aligned}\mathit{Terms} : M := {} & x \mid C_{j,k} \mid (M\ N) \mid \lambda x : A.M \mid \uparrow M \mid \downarrow M \mid \langle M_1, M_2 \rangle \\ & \mid \mathit{fst}\ M \mid \mathit{snd}\ M \mid \langle\sigma\rangle \mathit{Case}\ M\ \ \mathit{of}\ \ (M_{j,k}) \mid \langle\sigma\rangle \mathit{It}\ M\ \ \mathit{of}\ \ (M'_{j,k})\end{aligned}$$

where σ is a function mapping the ground types $L_j (j \in \mathbb{N})$ to types, $(M_{j,k})$ and $(M'_{j,k})$ are collections of terms indexed by the indexes of the constructors.

We delay till Section 2.5 the explanation of the arguments of operators 'Case' and 'It'. The modal operator '$\uparrow$' introduces an object of type $\Box A$ while the operator '$\downarrow$' marks the elimination of a term of type $\Box A$. As usual, terms equivalent under α-conversion are identified.

2.3 Typing Rules

The typing rules are a combination of the rules for simply-typed λ-calculus, for pairs and projections, for modal lambda-calculus IS4 ([PW95]) and the rules for the recursive constructs 'Case' and 'It'.

Rules for the simply-typed part of the calculus

First, we give the classic typing rules for the simply-typed part of the calculus and for pairs and projections.

$$\text{(Var)}\ \frac{x : A \in \text{ local context of } \Delta}{\Delta \vdash x : A}\ \Delta \text{ valid}$$

$$\text{(λ)}\ \frac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x : A.M\ :\ A \to B} \qquad \text{(App)}\ \frac{\Delta \vdash M : A \to B \qquad \Delta \vdash N : A}{\Delta \vdash (M\ N) : B}$$

$$\text{(Pair)}\ \frac{\Delta \vdash M_1 : A \qquad \Delta \vdash M_2 : B}{\Delta \vdash \langle M_1, M_2 \rangle : A \times B}$$

$$\text{(Fst)}\ \frac{\Delta \vdash M : A \times B}{\Delta \vdash \text{fst } M : A} \qquad \text{(Snd)}\ \frac{\Delta \vdash M : A \times B}{\Delta \vdash \text{snd } M : B}$$

Modal rules

The modal rules are exactly similar to those in [PW95].

$$\text{($\uparrow$)}\ \frac{\Delta; . \vdash M : A}{\Delta \vdash \uparrow M : \Box A} \qquad \text{($\downarrow$)}\ \frac{\Delta \vdash M : \Box A}{\Delta \vdash \downarrow M : A}$$

$$\text{(Pop)}\ \frac{\Delta \vdash M : \Box A}{\Delta; \Gamma \vdash M : \Box A}\ \Delta; \Gamma \text{ valid}$$

Inductive rules

$$(C_{j,k})\quad \frac{}{\Delta \vdash C_{j,k} : B_{j,k,1} \to \cdots \to B_{j,k,n_{j,k}} \to L_j}\quad \Delta \text{ valid}, n_{j,k} \in \mathbb{N}$$

where $B_{j,k,1}, \ldots, B_{j,k,n_{j,k}}$ are pure types. L_j is an inductive type. $(T_i)_{i=1,\ldots,p}$ is a collection, possibly empty, of pure types. Each T_i can be decomposed as ${T_i}^1 \to \cdots \to {T_i}^{r_i} \to L_i$, where L_i is a ground type and each ${T_i}^j$ is a pure type.

Given C, D_1, $\ldots$, D_p, we denote $D_1 \to \cdots \to D_p \to C$ by $\prod_{i=1}^{i=p} D_i.C$.

We define ${T'}_z$ by:

$$\begin{aligned} {T'}_z &:= \Box(T_1 \to \cdots \to T_p \to T_z^1) \to \cdots \to \Box(T_1 \to \cdots \to T_p \to T_z^{r_z}) \to \sigma(L_z) \\ &= \prod_{m=1}^{m=r_z} \Box\left(\prod_{i=1}^{i=p} T_i . T_z^m\right).\sigma(L_z) \end{aligned}$$

The map σ from ground types to types is extended over pure types by the following equation:

$$\sigma(A \to B) = \sigma(A) \to \sigma(B)$$

The typing rules for Case and It are the following ones:

$$\text{(Case)}\frac{\Delta \vdash M : \Box\left(\prod_{i=1}^{i=p} T_i.L_n\right) \qquad \Delta \vdash M_{j,k} : \prod_{q=1}^{q=n_{j,k}} \Box\left(\prod_{i=1}^{i=p} T_i.B_{j,k,q}\right).\sigma(L_j)}{\Delta \vdash \langle\sigma\rangle\text{Case } M \text{ of } (M_{j,k}) : \prod_{z=1}^{z=p} {T'}_z.\sigma(L_n)}$$

$$\text{(It)}\frac{\Delta \vdash M : \Box\left(\prod_{i=1}^{i=p} T_i.L_n\right) \qquad \Delta \vdash {M'}_{j,k} : \prod_{q=1}^{q=n_{j,k}} \sigma(B_{j,k,q}).\sigma(L_j)}{\Delta \vdash \langle\sigma\rangle\text{It } M \text{ of } ({M'}_{j,k}) : \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L_n)}$$

These typing rules may seem complex at first sight but they are naturally derived from the behaviour of the Case and It operators with respect to reduction (sections 2.5, 2.6).

Although expressed a bit differently, our typing rules are similar to those in [DPS96], except three notable differences:

- As noticed before, the modal core is different. We have context stacks instead of two contexts and our modal rules (↑), (↓) and (Pop) are expressed in a very simple way (like in [PW95]).
- In our last two typing rules the terms $M_{j,k}$ and ${M'}_{j,k}$ are a priori indexed by all the indexes of the constructors. This means that for each Case or It construct, we should define one term $M_{j,k}$ (or ${M'}_{j,k}$) per constructor $C_{j,k}$. Actually, we do not need all of them. When computing over inductive terms of type L_n, we only need to define the terms $M_{j,k}$ such that L_n and L_j are 'mutually' inductive. This notion is essential for a future implementation of an extension of this system as a logical framework but is quite orthogonal to the properties we state and we prove in the following. The interested reader will find the definition of mutual inductive types we could adopt here completely formalized in [DPS96].
- Our Case construct could take as argument a type A (like in [DPS96]) instead of a map σ from ground types to types. Indeed the reduction rules for Case ensure that only $\sigma(L_n)$ is significant (On the contrary, for the It construct, all the types $\sigma(L_z)$ such that L_z and L_n are mutually defined, are needed). Once again, this is orthogonal to the properties we state and our choice for an uniform presentation slightly alleviates the notations.

2.4 Basic Properties

The system allows the same basic stack manipulations as the modal λ-calculus IS4 without Case and It ([PW95]).

Proposition 2.3 (Stack Manipulations) *If a term is well-typed of type A in a context stack, it is still well-typed of the same type after the following manipulations of the stack:*

- *swapping of contexts (different from the local context if type A is not of the form $\Box B$),*

$$\Delta;\Gamma;\Delta';\Gamma';\Delta'' \rightsquigarrow \Delta;\Gamma';\Delta';\Gamma;\Delta''$$

- *thinning (adding a declaration inside a context anywhere in a stack),*

$$\Delta;\Gamma;\Delta' \rightsquigarrow \Delta;\Gamma,x:A;\Delta' \text{ if } \Delta;\Gamma,x:A;\Delta' \text{ is valid}$$

- *modal weakening (adding a context into the stack, anywhere except in the last position),*

$$\Delta;\Delta' \rightsquigarrow \Delta;\Gamma;\Delta' \text{ if } \Delta;\Gamma;\Delta' \text{ is valid}$$

- *strengthening (removing an unnecessary declaration, i.e. a declaration of a variable which is not free in the term to be typed)*

$$\Delta;\Gamma,x:A;\Delta' \rightsquigarrow \Delta;\Gamma;\Delta'$$

- *fusion (replacing some semi colons in a stack by commas, i.e. melting two contexts into a single one).*

$$\Delta;\Gamma;\Gamma';\Delta' \rightsquigarrow \Delta;\Gamma,\Gamma';\Delta'$$

Proof
This is easily proved by induction on the typing judgment. □

The notion of substitution is defined as usual. If M, N are terms and x is a variable, $M[N/x]$ denotes the term M where N is substituted for x. Note that:

$$\begin{array}{rcl}
(\uparrow M)[N/x] & \equiv & \uparrow (M[N/x]) \\
(\downarrow M)[N/x] & \equiv & \downarrow (M[N/x]) \\
(\langle\sigma\rangle\text{Case } M \text{ of } (M_{j,k}))[N/x] & \equiv & (\langle\sigma\rangle\text{Case } M[N/x] \text{ of } (M_{j,k}[N/x])) \\
(\langle\sigma\rangle\text{It } M \text{ of } ({M'}_{j,k}))[N/x] & \equiv & (\langle\sigma\rangle\text{It } M[N/x] \text{ of } ({M'}_{j,k}[N/x]))
\end{array}$$

Fortunately the rule of substitution is still admissible, i.e. provable with the typing rules given above:

Proposition 2.4 (Admissibillity of (Subst) Rule)
The following rule is admisible:

$$(Subst)\frac{\Delta \vdash N : A \qquad \Delta, x : A \vdash M : B}{\Delta \vdash M[N/x] : B}$$

Proof
This is easily proved by induction on the derivation of $\Delta, x : A \vdash M : B$. □

Notation If Δ is a valid stack of m contexts $\Gamma_1; \ldots; \Gamma_m$ and n is an integer, Δ^n denotes the stack Δ where the last n contexts have been removed: $\Gamma_1; \ldots; \Gamma_{m-n}$ if $n < m$, and the empty stack '.' if $n \geq m$.

The inversion lemmas are not usual because our typing rules are not syntax-driven, because of the (Pop) rule. The rules that make it a bit more complex are rules (App), (Fst), (Snd), (Case) and (It). For instance, the rule (App), seen from bottom to top, may break a modal type $\Box B$ into two types $A \to \Box B$ and A which are no more modal. Thus, when faced with an application, we try to apply the rule (Pop) as much as we can (i.e. remove the local context till we have free variables of the term in the local context). Nevertheless the following inversion lemmas remain fairly simple and they enable us to give a type checking algorithm.

Proposition 2.5 (Inversion lemmas)

- *If $\Delta \vdash x : A$ then either the declaration $x : A$ belongs to the local context of Δ, or A is of the form $\Box B$ and the declaration $x : A$ belongs to Δ.*
- *If $\Delta \vdash \lambda x : A.M : A \to B$ then $\Delta, x : A \vdash M : B$.*
- *If $\Delta \vdash (M \quad N) : B$ then there is a type A such that $\Delta^n \vdash M : A \to B$ and $\Delta^n \vdash N : A$ ($n \in \mathbb{N}$), where $n = 0$ if B is not of the form $\Box C$ and otherwise n is the least integer such that Δ^n contains some free variables of $(M \quad N)$.*
- *If $\Delta \vdash \langle M_1, M_2 \rangle : A \times B$ then $\Delta \vdash M_1 : A$ and $\Delta \vdash M_2 : B$.*
- *If $\Delta \vdash fst\ M : A$ then there is a type B such that $\Delta^n \vdash M : A \times B$ ($n \in \mathbb{N}$), where $n = 0$ if A is not of the form $\Box C$ and otherwise n is the least integer such that Δ^n contains some free variables of $fst\ M$.*
- *If $\Delta \vdash snd\ M : B$ then there is a type A such that $\Delta^n \vdash M : A \times B$ ($n \in \mathbb{N}$), where $n = 0$ if B is not of the form $\Box C$ and otherwise n is the least integer such that Δ^n contains some free variables of $snd\ M$.*
- *If $\Delta \vdash\uparrow M : \Box A$ then $\Delta; . \vdash M : A$.*
- *If $\Delta \vdash\downarrow M : A$ then $\Delta \vdash M : \Box A$.*

- *If* $\Delta \vdash \langle\sigma\rangle Case\ \ M\ \ of\ \ (M_{j,k}) : A$ *then there are pure types* $T_1, \ldots, T_p$ *and a ground type* L_j *such that* $A = \prod_{z=1}^{z=p} \left(\prod_{m=1}^{m=r_z} \Box \left(\prod_{i=1}^{i=p} T_i.T_z^m \right) .\sigma(L_z) \right).\sigma(L_j)$, $\Delta^n \vdash M : \Box \left(\prod_{i=1}^{i=p} T_i.L_j \right)$, *and* $\Delta^n \vdash M_{j,k} : \left(\prod_{q=1}^{q=n_{j,k}} \Box \left(\prod_{i=1}^{i=p} T_i.B_{j,k,q} \right) \right) .\sigma(L_j)$ *for* $n \in \mathbb{N}$, *where* $n = 0$ *if the type* A *is not of the form* $\Box C$ *and otherwise* n *is the least integer such that the local context of* Δ^n *contains some free variables of* $\langle\sigma\rangle Case\ \ M\ \ of\ \ (M_{j,k})$.
- *If* $\Delta \vdash \langle\sigma\rangle It\ \ M\ \ of\ \ (M'_{j,k}) : A$ *then there are pure types* $T_1, \ldots, T_p$ *and a ground type* L_j *such that* $A = \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L_j)$, $\Delta^n \vdash M : \Box \left(\prod_{i=1}^{i=p} T_i.L_j \right)$ *and* $\Delta^n \vdash M'_{j,k} : \prod_{q=1}^{q=n_{j,k}} \sigma(B_{j,k,q}).\sigma(L_j)$ *for* $n \in \mathbb{N}$, *where* $n = 0$ *if the type* A *is not of the form* $\Box C$ *and otherwise* n *is the least integer such that the local context of* Δ^n *contains some free variables of* $\langle\sigma\rangle It\ \ M\ \ of\ \ (M'_{j,k})$.

Proof
By induction on the derivation of the hypothesis. □

Using the inversion lemmas, an easy induction on the typing derivation shows uniqueness of type:

Corollary 2.6 (Uniqueness of type)
If $\Delta \vdash M : A$ *and* $\Delta \vdash M : A'$ *then* $A = A'$.

Despite the non-determinism of our typing system, given a stack Δ, the type of a term M is unique. Indeed, the only source of non-determinism in a typing derivation is Rule (Pop), which can be applied at any stage where we have a term of type $\Box C$. Fortunately, this rule does not modify the type of the term. Another important property of the typing system is decidability of typability.

Theorem 2.7 (Decidability of typability)
Typability is decidable, i.e. given a stack Δ *and a term* M, *there is an algorithm finding if* M *is typable in* Δ *and returning type* A *if* $\Delta \vdash M : A$.

Proof
We prove the theorem by describing the algorithm. First, we check if Δ is valid. If it is not, we know we will not be able to type any term in Δ. Then we inductively go through term M, applying the algorithm on immediate subterms of M. At each step, the inversion rules ensure soundness of the algorithm. For the sake of brevity, we only show two cases here:

- If $M \equiv x$, we lookup the type of x in Δ. If we do not find it, x cannot be typed in Δ. Otherwise, let us call A its associated type. If A is not of the form $\Box B$ and x is not declared in the local context then x cannot be typed in Δ. Otherwise we return A.

- If $M \equiv \langle\sigma\rangle\text{Case } M \text{ of } (M_{j,k})$, we iteratively remove the local context of Δ till it contains a free variable of M. Then we use the algorithm to type M and the $M_{j,k}$ in the new stack. If the type returned for M is $\Box\left(\prod_{i=1}^{i=p} T_i.L_j\right)$ and the type returned for $M_{j,k}$ is $\left(\prod_{q=1}^{q=n_{j,k}} \Box\left(\prod_{i=1}^{i=p} T_i.B_{j,k,q}\right)\right).\sigma(L_j)$ then we return type:

$$\prod_{z=1}^{z=p}\left(\prod_{m=1}^{m=r_z}\Box\left(\prod_{i=1}^{i=p} T_i.T_z^m\right).\sigma(L_z)\right).\sigma(L_j)$$

  Otherwise M cannot be typed in Δ. □

2.5 Reduction rules for Case and It on a simple example

Now, we turn to the semantics of our system. They are inspired by the reduction rules for Case and It that have been proposed by Martin Hofmann [Hof96] as a means to describe the evaluation mechanism of [DPS97]. These reduction rules are also the ones underlying the terms and induction principles presented in [DH94].

The arguments of Case and It constructs have different roles which are best illustrated by the reduction rules. First we show the reduction rules for Case and It in the simple setting of the example of Section 2.1. For the sake of simplicity we introduce some notations.

Notations. For any type B, we define B_n by the following equations:

$$\begin{aligned} B_0 &= B \\ B_n &= B \to B_{n-1} \end{aligned}$$

We consider a map σ from the inductive types to types such that $\sigma(L) = A$, terms M_{app} of type $L_n \to L_n \to A$, M_{lam} of type $L_{n+1} \to A$, M'_{app} of type $A \to A \to A$ and M'_{lam} of type $(A \to A) \to A$.

For the sake of simplicity, we define two macros 'case' and 'it' by:

$$\begin{aligned} \text{case } M &:= \langle\sigma\rangle\text{Case } M \text{ of } M_{\text{app}}\ M_{\text{lam}} \\ \text{it } M &:= \langle\sigma\rangle\text{It } M \text{ of } M'_{\text{app}}\ M'_{\text{lam}} \end{aligned}$$

Reduction rules. In this example, the reduction rules for Case and It are the following ones:

$$\begin{aligned} (\text{case } \uparrow\lambda\vec{x}:L^n.(\text{app } P\ Q)) &\hookrightarrow \lambda\vec{u}:A^n.(M_{\text{app}}\ \uparrow\lambda\vec{x}:L^n.P\ \uparrow\lambda\vec{x}:L^n.Q) \\ (\text{case } \uparrow\lambda\vec{x}:L^n.(\text{lam } P)) &\hookrightarrow \lambda\vec{u}:A^n.(M_{\text{lam}}\ \uparrow\lambda\vec{x}:L^n.P) \\ (\text{case } \uparrow\lambda\vec{x}:L^n.x_i) &\hookrightarrow \lambda\vec{u}:A^n.u_i \end{aligned}$$

$$\begin{array}{rcl}(\text{it } \uparrow \lambda\vec{x} : L^n.(\text{app } P\ Q)) & \hookrightarrow & \lambda\vec{u} : A^n.(M'_{\text{app}}\ ((\text{it } \uparrow \lambda\vec{x} : L^n.P)\ \vec{u})\ ((\text{it } \uparrow \lambda\vec{x} : L^n.Q)\ \vec{u})) \\ (\text{it } \uparrow \lambda\vec{x} : L^n.(\text{lam } P)) & \hookrightarrow & \lambda\vec{u} : A^n.(M'_{\text{lam}}\ ((\text{it } \uparrow \lambda\vec{x} : L^n.P)\ \vec{u})) \\ (\text{it } \uparrow \lambda\vec{x} : L^n.x_i) & \hookrightarrow & \lambda\vec{u} : A^n.u_i \end{array}$$

The first argument of the Case and It constructs, M, is the inductive term to dissect (representing an untyped lambda term in our example). The second one, M_{app}, is the function which processes the case of constructor 'app'. The third argument, M_{lam}, is the function which processes the case of constructor 'lam'. Roughly speaking the 'Case' construct computes its result by applying M_{app} or M_{lam} to the sons of its main argument. For iteration, the mechanism of reduction is a bit different: the terms M'_{app} and M'_{lam} are applied to the result of 'It' on the sons of the main argument. In fact, the effect of 'It' on a term M amounts to replacing the constructors lam and app by the terms M'_{lam} and M'_{app} in M (see [DPS97]).

Now since we want to benefit from higher order declarations, the main argument of Case/It can have a functional type. In particular we also want to be able to compute Case/It of a projection $\lambda\vec{x} : L^n.x_i$ without a leftmost constructor. That is the reason for the functional type of Case/It constructs : they take as input the values of the computation for the projections (see [DPS97]).

Examples. Let us assume that we have defined the types of the integers 'Nat' by declaring two constructors '0 : Nat' and 'S : Nat $\to$ Nat'.

We can informally define the function which counts the number of bound variables in an untyped λ-term by:

- Count(app M N) = Count(M) + Count(N)
- Count(lam $\lambda x : L.(M\ x)$) = Count($M\ x$) , where Count(x) = 1

This function can be implemented in our system with the It construct:

$$\text{Count } M := \langle\sigma\rangle\text{It } \ M \ \text{ of } \ \lambda m : \text{Nat}.\lambda n : \text{Nat}.(\text{plus } m\ n) \ \ \lambda p : \text{Nat} \to \text{Nat}.(p\ (S\ 0))$$

Count has type $\Box L \to \text{Nat}$. Let us give an example of computation: let us count the number of bound variables in the untyped lambda term $(\lambda x.x\ \ \lambda x.x)$:

$$\begin{array}{l}. \vdash (\text{Count } \ \uparrow (\text{app } (\text{lam } \lambda x : L.x)\ (\text{lam } \lambda x : L.x))) \hookrightarrow_* \\ \quad (\lambda m : \text{Nat}.\lambda n : \text{Nat}.(\text{plus } m\ n) \ \ (\text{Count } \ \uparrow (\text{lam } \lambda x : L.x)) \ \ (\text{Count } \ \uparrow (\text{lam } \lambda x : L.x)))\end{array}$$

which reduces to $(S\ (S\ 0))$ since

$$\begin{array}{l}. \vdash (\text{Count } \ \uparrow (\text{lam } \lambda x : L.x)) \hookrightarrow \\ \quad (\lambda p : \text{Nat} \to \text{Nat}.(p\ (S\ 0)) \ \ (\text{Count } \ \uparrow \lambda x : L.x)) \hookrightarrow_* \\ \quad ((\text{Count } \ \uparrow \lambda x : L.x) \ \ (S\ 0)) \hookrightarrow_* (S\ 0)\end{array}$$

To further illustrate the reduction mechanism, let us define the function 'Form' of type $\Box(L \to L) \to \mathrm{Nat}$, which returns 0 if its argument is a free variable, 1 if it is an abstraction term and 2 if it is an application.

$$\mathrm{Form}\ M := (\langle\sigma\rangle\mathrm{Case}\ \ M\ \ \mathrm{of}\ \ \lambda u : \Box(L \to L).\lambda v : \Box(L \to L).2\ \ \lambda f : \Box(L \to L \to L).1\ \ \ 0)$$

For instance,

$$(\mathrm{Form}\ \ \uparrow \lambda x : L.x) \hookrightarrow (\lambda n : \mathrm{Nat}.n\ 0) \hookrightarrow 0$$

and

$$(\mathrm{Form}\ \ \uparrow \lambda x : L.(\mathrm{lam}\ \lambda y : L.x)) \hookrightarrow (\lambda n : \mathrm{Nat}.(\lambda f : \Box(L \to L \to L).1\ \ \uparrow \lambda x : L.\lambda y : L.x)\ 0) \hookrightarrow 1$$

2.6 Reduction rules

Now we describe here the whole set of reduction rules. Given a term of our calculus, what we want to obtain at the end of the computation is the term of the object language it represents. As we have seen earlier (Section 2.1), the canonical forms (β-normal η-long) are in one-to-one correspondence with the object terms. Thus we want the computation to return canonical forms.

That means our reduction rules will incorporate η-expansion. From a technical point of view, the choice of η-expansion allows us to keep simple Case/It redexes. Indeed if we allowed η-contraction, we would have the following reductions:

$$\begin{array}{lll} . & \vdash & \langle\sigma\rangle\mathrm{Case}\ \uparrow \lambda x : L.\lambda y : L.(\mathrm{app}\ x\ y)\ \ \mathrm{of}\ \ M_{\mathrm{app}}\ M_{\mathrm{lam}} \hookrightarrow \\ & & \lambda u : A.\lambda v : A.(M_{\mathrm{app}}\ \ \uparrow \lambda x : L.\lambda y : L.x\ \ \uparrow \lambda x : L.\lambda y : L.y) : A \to A \to A \end{array}$$

by reduction of the Case-redex and

$$\begin{array}{lll} . & \vdash & \langle\sigma\rangle\mathrm{Case}\ \uparrow \lambda x : L.\lambda y : L.(\mathrm{app}\ x\ y)\ \ \mathrm{of}\ \ M_{\mathrm{app}}\ M_{\mathrm{lam}} \hookrightarrow \\ & & \langle\sigma\rangle\mathrm{Case}\ \uparrow \lambda x : L.(\mathrm{app}\ x)\ \ \mathrm{of}\ \ M_{\mathrm{app}}\ M_{\mathrm{lam}} : A \to A \to A \end{array}$$

after η-contraction inside the Case. This example shows that in order to have confluence of the reduction, we should allow η-contractions of Case-redexes to be also Case-redexes. In other words, a term like $\langle\sigma\rangle$Case app of ... would have to be a Case/It redex. It does not seem very natural since app, which is not a term in canonical form, does not represent any term at the object level.

The η-expansion reduction rule has been thoroughly studied (see [CK93], [Aka93], [JG95]). Adopting it forces us to restrict the reduction rules in some way if we still want Strong Normalization (these restrictions were first introduced by [Min79]). Thus the reduction we will consider will not be a congruence (more precisely it will not be compatible with the application) and this will induce slight changes in the usual schemes of the proofs of the Church-Rosser and Strong Normalization properties.

$$(\beta)\ \frac{\Delta \vdash (\lambda x : A.P\ \ Q) : B}{\Delta \vdash (\lambda x : A.P\ \ Q) \hookrightarrow P[Q/x] : B} \qquad (\beta\Box)\ \frac{\Delta;. \vdash \downarrow\uparrow M : A}{\Delta \vdash \downarrow\uparrow M \hookrightarrow M : A}$$

$$(\eta)\ \frac{\Delta \vdash M : A \to B \quad M \text{ is not an abstraction} \quad x \text{ fresh}}{\Delta \vdash M \hookrightarrow \lambda x : A.(M\ x) : A \to B} \qquad (\eta\Box)\ \frac{\Delta;. \vdash \uparrow\downarrow M : A}{\Delta \vdash \uparrow\downarrow M \hookrightarrow M : A}$$

$$(\mathrm{App_l})\ \frac{\Delta \vdash M \hookrightarrow M' : A \to B\ (\neq \eta\text{-step}) \quad \Delta \vdash N : A}{\Delta \vdash (M\ \ N) \hookrightarrow (M'\ \ N) : B} \qquad (\mathrm{Pop})\ \frac{\Delta \vdash M \hookrightarrow N : \Box A}{\Delta;\Gamma \vdash M \hookrightarrow N : \Box A}$$

$$(\mathrm{App_r})\ \frac{\Delta \vdash M : A \to B \qquad \Delta \vdash N \hookrightarrow N' : A}{\Delta \vdash (M\ \ N) \hookrightarrow (M\ \ N') : B} \qquad (\downarrow)\ \frac{\Delta \vdash M \hookrightarrow N : \Box A}{\Delta \vdash\downarrow M \hookrightarrow\downarrow N : A}$$

$$(\lambda)\ \frac{\Delta;\Gamma, x : A \vdash M \hookrightarrow N : B}{\Delta;\Gamma \vdash \lambda x : A.M \hookrightarrow \lambda x : A.N : A \to B} \qquad (\uparrow)\ \frac{\Delta;. \vdash M \hookrightarrow N : A}{\Delta \vdash\uparrow M \hookrightarrow\uparrow N : \Box A}$$

$$(\Pi_1)\ \frac{\Delta \vdash \langle M_1, M_2\rangle : A \times B}{\Delta \vdash \mathrm{fst}\langle M_1, M_2\rangle \hookrightarrow M_1 : A} \qquad (\Pi_2)\ \frac{\Delta \vdash \langle M_1, M_2\rangle : A \times B}{\Delta \vdash \mathrm{snd}\langle M_1, M_2\rangle \hookrightarrow M_2 : B}$$

$$(\mathrm{sp})\ \frac{\Delta \vdash \langle \mathrm{fst}\ M, \mathrm{snd}\ M\rangle : A}{\Delta \vdash \langle \mathrm{fst}\ M, \mathrm{snd}\ M\rangle \hookrightarrow M : A}$$

$$(\mathrm{Pair_l})\ \frac{\Delta \vdash M_1 \hookrightarrow M_1' : A \qquad \Delta \vdash M_2 : B}{\Delta \vdash \langle M_1, M_2\rangle \hookrightarrow \langle M_1', M_2\rangle : A \times B} \qquad (\mathrm{Fst})\ \frac{\Delta \vdash M \hookrightarrow M' : A \times B}{\Delta \vdash \mathrm{fst}\ M \hookrightarrow \mathrm{fst}\ M' : A}$$

$$(\mathrm{Pair_r})\ \frac{\Delta \vdash M_2 \hookrightarrow M_2' : B \qquad \Delta \vdash M_1 : A}{\Delta \vdash \langle M_1, M_2\rangle \hookrightarrow \langle M_1, M_2'\rangle : A \times B} \qquad (\mathrm{Snd})\ \frac{\Delta \vdash M \hookrightarrow M' : A \times B}{\Delta \vdash \mathrm{snd}\ M \hookrightarrow \mathrm{snd}\ M' : B}$$

Figure 1: Reduction rules. Simple types, product and modality

The purpose of these restrictions is to prevent η-expansions to create new β-redexes which generate reduction loops. For instance, if we allowed an abstraction to be η-expanded, we would have:

$$\Delta \vdash \lambda x : A.M \hookrightarrow_\eta \lambda y : A.(\lambda x : A.M\ \ y) \hookrightarrow_\beta \lambda y : A.M[y/x] : A \to B$$

which is α-convertible to $\lambda x : A.M$. Thus we forbid η-expansion of an abstraction. Similarly, we cannot allow η-expansion of the left argument of an application because otherwise we would have $\Delta \vdash (M\ N) \hookrightarrow (\lambda z : A.(M\ z)\ N) \hookrightarrow (M\ N) : B$.

The choice of η-expansion also means we have to keep track of the types of the terms. Indeed a term can only be η-expanded if it has type $A \to B$. Thus we will define a notion of typed reduction.

The reduction relation is defined by the inference rules in Figure 1 (simple types, product and modality), Figure 2 (reduction rules for Case and It) and Figure 3 (compatibility rules for Case and It), where we have voluntarily omitted some types in the rules. They can be immediately retrieved from Section 2.3.

$$\frac{\Delta \vdash \langle\sigma\rangle\text{Case } \uparrow\lambda\vec{x}:\vec{T}.(C_{j,k}\ M_1 \ldots M_{n_{j,k}}) \text{ of } (M_{j,k}) : \prod_{z=1}^{z=p} T'_z.\sigma(L_n)}{\begin{array}{l}\Delta \vdash \langle\sigma\rangle\text{Case } \uparrow\lambda\vec{x}:\vec{T}.(C_{j,k}\ M_1 \ldots M_{n_{j,k}}) \text{ of } (M_{j,k}) \hookrightarrow \\ \quad \lambda\vec{u}:\vec{T'}.(M_{j,k} \ \uparrow\lambda\vec{x}:\vec{T}.M_1 \ \ldots \ \uparrow\lambda\vec{x}:\vec{T}.M_{n_{j,k}}) : \prod_{z=1}^{z=p} T'_z.\sigma(L_n)\end{array}}$$

$$\frac{\Delta \vdash \langle\sigma\rangle\text{Case } \uparrow\lambda\vec{x}:\vec{T}.(x_k\ M_1 \ldots M_{r_k}) \text{ of } (M_{j,k}) : \prod_{z=1}^{z=p} T'_z.\sigma(L_n)}{\begin{array}{l}\Delta \vdash \langle\sigma\rangle\text{Case } \uparrow\lambda\vec{x}:\vec{T}.(x_k\ M_1 \ldots M_{r_k}) \text{ of } (M_{j,k}) \hookrightarrow \\ \quad \lambda\vec{u}:\vec{T'}.(u_k \ \uparrow\lambda\vec{x}:\vec{T}.M_1 \ \ldots \ \uparrow\lambda\vec{x}:\vec{T}.M_{r_k}) : \prod_{z=1}^{z=p} T'_z.\sigma(L_n)\end{array}}$$

$$\frac{\Delta \vdash \langle\sigma\rangle\text{It } \uparrow\lambda\vec{x}:\vec{T}.(C_{j,k}\ M_1 \ldots M_{n_{j,k}}) \text{ of } (M'_{j,k}) : \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L_n)}{\begin{array}{l}\Delta \vdash \langle\sigma\rangle\text{It } \uparrow\lambda\vec{x}:\vec{T}.(C_{j,k}\ M_1 \ldots M_{n_{j,k}}) \text{ of } (M'_{j,k}) \hookrightarrow \\ \quad \lambda\vec{u}:\sigma(\vec{T}).(M'_{j,k} \ (\langle\sigma\rangle\text{It } \uparrow\lambda\vec{x}:\vec{T}.M_1 \text{ of } (M'_{j,k}) \ \vec{u}) \ \ldots \\ \qquad (\langle\sigma\rangle\text{It } \uparrow\lambda\vec{x}:\vec{T}.M_{n_{j,k}} \text{ of } (M'_{j,k}) \ \vec{u})) : \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L_n)\end{array}}$$

$$\frac{\Delta \vdash \langle\sigma\rangle\text{It } \uparrow\lambda\vec{x}:\vec{T}.(x_k\ M_1 \ldots M_{r_k}) \text{ of } (M'_{j,k}) : \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L_n)}{\begin{array}{l}\Delta \vdash \langle\sigma\rangle\text{It } \uparrow\lambda\vec{x}:\vec{T}.(x_k\ M_1 \ldots M_{r_k}) \text{ of } (M'_{j,k}) \hookrightarrow \\ \quad \lambda\vec{u}:\sigma(\vec{T}).(u_k \ (\langle\sigma\rangle\text{It } \uparrow\lambda\vec{x}:\vec{T}.M_1 \text{ of } (M'_{j,k}) \ \vec{u}) \ \ldots \\ \qquad (\langle\sigma\rangle\text{It } \uparrow\lambda\vec{x}:\vec{T}.M_{r_k} \text{ of } (M'_{j,k}) \ \vec{u})) : \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L_n)\end{array}}$$

Figure 2: Reduction rules for Case and It

$$\frac{\Delta \vdash \langle\sigma\rangle\text{Case } M \text{ of } (M_{j,k}) : \prod_{z=1}^{z=p} T'_z.\sigma(L_n) \qquad \Delta \vdash M \hookrightarrow M' : \_}{\Delta \vdash \langle\sigma\rangle\text{Case } M \text{ of } (M_{j,k}) \hookrightarrow \langle\sigma\rangle\text{Case } M' \text{ of } (M_{j,k}) : \_}$$

$$\frac{\Delta \vdash \langle\sigma\rangle\text{Case } M \text{ of } (M_{j,k}) : \prod_{z=1}^{z=p} T'_z.\sigma(L_n) \qquad \Delta \vdash M_{j,k} \hookrightarrow N_{j,k} : \_}{\Delta \vdash \langle\sigma\rangle\text{Case } M \text{ of } (M_{j,k}) \hookrightarrow \langle\sigma\rangle\text{Case } M \text{ of } (N_{j,k}) : \_}$$

$$\frac{\Delta \vdash \langle\sigma\rangle\text{It } M \text{ of } (M'_{j,k}) : \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L_n) \qquad \Delta \vdash M \hookrightarrow M' : \_}{\Delta \vdash \langle\sigma\rangle\text{It } M \text{ of } (M'_{j,k}) \hookrightarrow \langle\sigma\rangle\text{It } M' \text{ of } (M'_{j,k}) : \_}$$

$$\frac{\Delta \vdash \langle\sigma\rangle\text{It } M \text{ of } (M'_{j,k}) : \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L_n) \qquad \Delta \vdash M'_{j,k} \hookrightarrow N'_{j,k} : \_}{\Delta \vdash \langle\sigma\rangle\text{It } M \text{ of } (M'_{j,k}) \hookrightarrow \langle\sigma\rangle\text{It } M \text{ of } (N'_{j,k}) : \_}$$

Figure 3: Compatibility reduction rules for Case and It

Note that we have no reason to compute $\eta_\Box$ long forms, i.e. expand a term M of type $\Box A$ into $\uparrow\downarrow M$. We want η-expanded forms because we are interested in the terms that are in one-to-one correspondence with the object-level terms. Modality only serves here as a tool to mix induction and higher order abstract syntax techniques. $\eta_\Box$ long forms have no special virtue that deserves our special attention. Similarly for products, the rule (sp) is a contraction rule.

As usual we define the relations $\hookrightarrow_*$ and $=$ (conversion) respectively as the reflexive, transitive and the reflexive, symmetric, transitive closures of $\hookrightarrow$.

We will also write $\Delta \vdash M_0 \hookrightarrow \ldots \hookrightarrow M_n : A$ for $\Delta \vdash M_0 \hookrightarrow M_1 : A, \ldots, \Delta \vdash M_{n-1} \hookrightarrow M_n : A$.

Like typing judgments, reduction judgments are preserved by some manipulations of the stack. Proposition 2.3 is still valid for reduction judgments. In particular, if stack (Δ, Γ) is valid and $\Delta \vdash M \hookrightarrow N : A$ then $\Delta, \Gamma \vdash M \hookrightarrow N : A$.

2.7 Canonical forms

The canonical forms can be defined by induction. Once again we have to take into account the types of the terms. Following the usual definition for the simply-typed λ-calculus (see [DPS97]), we define two judgments: "is atomic" and "is canonical".

Definition 2.8 (Canonical forms)

- $\Delta \vdash M \Downarrow A$ *means* M *is* atomic *of type* A *in* Δ
- $\Delta \vdash M \Uparrow A$ *means* M *is* canonical *of type* A *in* Δ

These judgments are defined by the following rules:

$$\frac{\Delta \vdash x : A}{\Delta \vdash x \Downarrow A} \qquad \frac{\Delta \vdash M \Downarrow B \quad B \text{ not an arrow type}}{\Delta \vdash M \Uparrow B}$$

$$\frac{\Delta \vdash M \Downarrow A \to B \qquad \Delta \vdash N \Uparrow A}{\Delta \vdash (M \ \ N) \Downarrow B} \qquad \frac{\Delta \vdash M \Uparrow \Box A \qquad M \neq\uparrow M'}{\Delta \vdash\downarrow M \Downarrow A}$$

$$\frac{\Delta, x : A \vdash M \Uparrow B}{\Delta \vdash \lambda x : A.M \Uparrow A \to B} \qquad \frac{\Delta; . \vdash M \Uparrow A \qquad M \neq\downarrow M'}{\Delta \vdash\uparrow M \Uparrow \Box A}$$

$$\frac{\Delta \vdash M \Downarrow \Box A}{\Delta; \Gamma \vdash M \Downarrow \Box A} \qquad \frac{\Delta \vdash M_1 \Uparrow A \quad \Delta \vdash M_2 \Uparrow B \quad (M_1, M_2) \neq (\mathit{fst}\ N_1, \mathit{snd}\ N_2)}{\Delta \vdash \langle M_1, M_2 \rangle \Uparrow A \times B}$$

$$\frac{\Delta \vdash M \Uparrow A \times B \quad M \neq \langle M_1, M_2 \rangle}{\Delta \vdash \mathit{fst}\ M \Downarrow A} \qquad \frac{\Delta \vdash M \Uparrow A \times B \quad M \neq \langle M_1, M_2 \rangle}{\Delta \vdash \mathit{snd}\ M \Downarrow B}$$

$$\frac{\Delta \vdash M \Uparrow \Box \left(\prod_{i=1}^{i=p} T_i.L_n \right) \quad \Delta \vdash M_{j,k} \Uparrow \_ \quad \langle\sigma\rangle \mathit{Case}\ M\ \mathit{of}\ (M_{j,k})\ \mathit{is\ not\ a\ redex}}{\Delta \vdash \langle\sigma\rangle \mathit{Case}\ M\ \mathit{of}\ (M_{j,k}) \Downarrow \prod_{z=1}^{z=p} {T'}_z.\sigma(L_n)}$$

$$\frac{\Delta \vdash M \Uparrow \Box \left(\prod_{i=1}^{i=p} T_i.L_n \right) \quad \Delta \vdash {M'}_{j,k} \Uparrow \_ \quad \langle\sigma\rangle \mathit{It}\ M\ \mathit{of}\ ({M'}_{j,k})\ \mathit{is\ not\ a\ redex}}{\Delta \vdash \langle\sigma\rangle \mathit{It}\ M\ \mathit{of}\ ({M'}_{j,k}) \Downarrow \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L_n)}$$

Proposition 2.9

- $\Delta \vdash M \Downarrow A$ *iff* $\Delta \vdash M : A$, M *is not an abstraction and the only reduction rule that can be applied to* M *is* η*-expansion (only possible if* A *is of the form* $B \to C$*).*
- $\Delta \vdash M \Uparrow A$ *iff* $\Delta \vdash M : A$ *and there is no reduction rule that can be applied to* M*.*

Proof We only give the sketch of the proof:
($\Rightarrow$) by induction on the hypothesis.
($\Leftarrow$) by induction on the well-typedness hypothesis. □

3 Metatheoretical results

This section is devoted to the proofs of the metatheoretical results. The classic properties of subject reduction, confluence, strong normalization have already been established for a modal lambda-calculus IS4 without induction ([Lel98]). Here we extend these results to the recursive operators Case and It.

We begin this section with the easiest result: soundness of reduction with respect to typing rules. Then, we state important lemmas about substitution and reduction and show the local confluence property. Finally, we prove the key result, i.e. the strong normalization property, from which we deduce confluence and conservative extension.

3.1 First Results

First, we state soundness of typed reduction with respect to typing rules. Then we prove important lemmas involving reduction and substitution.

Theorem 3.1 (Soundness of reduction)
If $\Delta \vdash M \hookrightarrow M' : A$ *then* $\Delta \vdash M : A$ *and* $\Delta \vdash M' : A$*.*

Proof
Easy. By induction on the derivation of the hypothesis $\Delta \vdash M \hookrightarrow M' : A$. □

Now, we examine the relationship between substitution and typed reduction. In the simply-typed λ-calculus, if $P \hookrightarrow_* P'$ and $Q \hookrightarrow_* Q'$ then $P[Q/x] \hookrightarrow_* P'[Q'/x]$. The proof of this property is based on the compatibility of the reduction with the operations. Here, it will not be valid because of the side-conditions of reduction rules (η) and ($\mathrm{App_l}$). Fortunately any 'illicit' η-expansion step can be 'reversed' by a β-reduction. Indeed we have:

$$(\lambda z : A.(M\ z)\ N) \hookrightarrow (M\ N)$$

and

$$\lambda z : A.(\lambda y : A.M\ z) \hookrightarrow \lambda z : A.M[z/y] =_\alpha \lambda y : A.M$$

This property enables us to prove weak forms of the usual results. In the first lemma we examine the case where Q reduces to Q' and explicit the links between $P[Q/x]$ and $P[Q'/x]$. Let us give an example where $P[Q/x]$ does not reduce to $P[Q'/x]$. In the stack $\Delta := (f : L \to L, g : (L \to L) \to L)$, we consider $P := (x\ (g\ x))$ and $Q := f$. We have the following η-expansion:

$\Delta \vdash Q \equiv f \hookrightarrow \lambda y : L.(f\ y) \equiv Q' : L \to L$

However,

$\Delta \vdash P[Q/x] \equiv (f\ (g\ f)) \not\hookrightarrow (\lambda y : L.(f\ y)\ (g\ \lambda y : L.(f\ y))) \equiv P[Q'/x] : L$

because the left argument of an application cannot be η-expanded. Nevertheless, both $P[Q/x]$ and $P[Q'/x]$ have a common reduct: $(f\ (g\ \lambda y : L.(f\ y)))$.

Definition. In the following, we call *basic* η-expansion a reduction step $\Delta \vdash M \hookrightarrow \lambda z : A.(M\ z) : A \to B$ directly obtained by rule (η). We say that a sequence of reductions $\Delta \vdash M \hookrightarrow_* N : A \to B$ has a basic η-expansion if one of the one-step reductions is a basic η-expansion.

Lemma 3.2

Let us assume that $\Delta, x : A \vdash P : B$. *If* $\Delta \vdash Q \hookrightarrow Q' : A$ *then*

- *there is a term* R *s.t.* $\Delta \vdash P[Q/x] \hookrightarrow_* R : B$ *and* $\Delta \vdash P[Q'/x] \hookrightarrow_* R : B$
- *there is no* η*-expansion in the sequence* $\Delta \vdash P[Q'/x] \hookrightarrow_* R : B$ *(only* β*-reductions)*
- *the sequence* $\Delta \vdash P[Q/x] \hookrightarrow_* R : B$ *has basic* η *expansions only if* $P \equiv x$ *and* Q' *is obtained from* Q *by a basic* η*-expansion.*

Proof

By induction on the proof of $\Delta, x : A \vdash P : B$

- (Var) If $P \equiv z$ $(z \neq x)$ then $P[Q/x] \equiv P[Q'/x] \equiv z$. Otherwise if $P \equiv x$ then $B \equiv A$, $P[Q/x] \equiv Q$ and $P[Q'/x] \equiv Q'$. $\Delta \vdash P[Q/x] \hookrightarrow P[Q'/x] : A$ is a basic η-expansion iff $\Delta \vdash Q \hookrightarrow Q' : B$ is a basic η-expansion.
- (Lam) $B = C \to D$ and $P \equiv \lambda z : C.P_1$, where z has been renamed so that $z \neq x$ and z is not free in Q. Then $P[Q/x] \equiv \lambda z : C.P_1[Q/x]$ and $P[Q'/x] \equiv \lambda z : C.P_1[Q'/x]$. By induction hypothesis, $\Delta, z : C \vdash P_1[Q/x] \hookrightarrow_* R : D$ and $\Delta, z : C \vdash P_1[Q'/x] \hookrightarrow_* R : D$. Thus, by Rule (λ), $\Delta \vdash \lambda z : C.P_1[Q/x] \hookrightarrow_* \lambda z : C.R : C \to D$ and $\Delta \vdash \lambda z : C.P_1[Q'/x] \hookrightarrow_* \lambda z : C.R : C \to D$ with no basic η-expansion.
- (App) $P \equiv (P_1\ P_2)$. In this case $P[Q/x] \equiv (P_1[Q/x]\ \ P_2[Q/x])$. By induction hypothesis, $\Delta \vdash P_1[Q/x] \hookrightarrow_* R_1 : A \to B$, $\Delta \vdash P_1[Q'/x] \hookrightarrow_* R_1 : A \to B$ and $\Delta \vdash P_2[Q/x] \hookrightarrow_* R_2 : A$, $\Delta \vdash P_2[Q'/x] \hookrightarrow_* R_2 : A$.

  We know by induction hypothesis that $\Delta \vdash P_1[Q'/x] \hookrightarrow_* R_1 : A \to B$ has no basic η-expansion. If $\Delta \vdash P_1[Q/x] \hookrightarrow_* R_1 : A \to B$ has no basic η-expansion, then we can conclude that $\Delta \vdash (P_1[Q/x]\ P_2[Q/x]) \hookrightarrow_* (R_1\ R_2) : B$ and $\Delta \vdash (P_1[Q'/x]\ P_2[Q'/x]) \hookrightarrow_* (R_1\ R_2) : B$. The most delicate case occurs if $\Delta \vdash P_1[Q/x] \hookrightarrow_* R_1 : A \to B$ has a basic η-expansion.

By induction hypothesis, it means that it is actually a one-step basic η-expansion and that $P \equiv (x \ \ P_2)$, $P[Q/x] \equiv (Q \ \ P_2[Q/x])$ and $P[Q'/x] \equiv (\lambda z : A.(Q\ z) \ \ P_2[Q'/x])$. Both $P[Q/x]$ and $P[Q'/x]$ can be reduced by β-reduction to $(Q \ \ R_2)$ and we conclude by applying the induction hypothesis on P_2.

- The other typing rules offer no resistance because our reduction is compatible with $\downarrow$, $\uparrow$, '$\langle \_, \_ \rangle$', fst, snd, Case and It. □

Similarly, if $\Delta, x : A \vdash P \hookrightarrow P' : B$, it is not always the case that $\Delta \vdash P[Q/x] \hookrightarrow_* P'[Q/x] : B$. For example, if $P \equiv x$, $Q \equiv \lambda z : C.M$ and $P' \equiv \lambda y : C.(x\ y)$ in stack Δ (y is fresh), then $P[Q/x] \equiv Q$ and $P'[Q/x] \equiv \lambda y : C.(\lambda z : C.M\ y)$. Thus $\Delta \vdash P'[Q/x] \hookrightarrow P[Q/x] : C \to D$ by β-reduction, but $\Delta \vdash P[Q/x] \not\hookrightarrow_* P'[Q/x] : C \to D$.

Lemma 3.3
Let us assume that $\Delta \vdash Q : A$. If $\Delta, x : A \vdash P \hookrightarrow P' : B$ then

- *If P' is obtained from P by η-expansion of an occurrence of x and Q is an abstraction then $\Delta \vdash P'[Q/x] \hookrightarrow P[Q/x] : B$. Moreover this reduction is a β-reduction.*
- *Otherwise, $\Delta \vdash P[Q/x] \hookrightarrow P'[Q/x] : B$ and it is a basic η-expansion if $\Delta, x : A \vdash P \hookrightarrow P' : B$ is a basic η-expansion.*

Proof By induction on the derivation of $\Delta, x : A \vdash P \hookrightarrow P' : B$.

- (β) $P \equiv (\lambda z : A.M \ \ N)$ and $P' \equiv M[N/z]$. We rename z so that $z \neq x$ and it is not free in Q. Then $\Delta \vdash P[Q/x] \equiv (\lambda z : A.M[Q/x] \ \ N[Q/x]) \hookrightarrow M[Q/x][N[Q/x]/z]$ which is equal to $M[N/z][Q/x]$, i.e. $P'[Q/x]$.
- (Lam) $P \equiv \lambda z : A.M$. Once again we rename z so that $z \neq x$ and it is not free in Q. We assume that $\Delta \vdash \lambda z : A.M \hookrightarrow \lambda z : A.N$. Then $(\lambda z : A.M)[Q/x] \equiv \lambda z : A.(M[Q/x])$ and $(\lambda z : A.N)[Q/x] \equiv \lambda z : A.(N[Q/x])$. We conclude by using the induction hypothesis.
- (App_r) Easy using induction hypothesis and compatibility rules
- (App_l) We assume that $P \equiv (M\ N)$, $P' \equiv (M'\ N)$ and $\Delta \vdash M \hookrightarrow M' : A \to B$ (not an η-expansion step). By induction hypothesis, $\Delta \vdash M[Q/x] \hookrightarrow M'[Q/x] : A \to B$ and it is not an η-expansion step either. Thus, $\Delta \vdash (M[Q/x] \ \ N[Q/x]) \hookrightarrow (M'[Q/x] \ \ N[Q/x]) : B$.
- (η) We assume that $\Delta \vdash P \hookrightarrow P' \equiv \lambda z : A.(P \ \ z) : A \to B$ where z is a fresh variable and P is not an abstraction. Thus $P'[Q/x] \equiv \lambda z : A.(P \ \ z)[Q/x] \equiv \lambda z : A.(P[Q/x] \ \ z)$. If $P[Q/x]$ is not an abstraction then by η-expansion $P[Q/x]$reduces to $\lambda z : A.(P[Q/x] \ \ z)$, i.e. $P'[Q/x]$. Otherwise it means that:
  - either P is an abstraction, which is impossible by hypothesis.

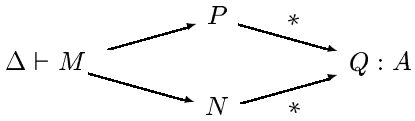
- or $P \equiv x$ and Q is an abstraction $\lambda y : A.Q_1$. In this case $\Delta \vdash P'[Q/x] \equiv \lambda z : A.(\lambda y : A.Q_1 \ \ z) \hookrightarrow_\beta \lambda z : A.Q_1[z/y] : A \to B$ and $\lambda z : A.Q_1[z/y]$ is α-convertible to $\lambda y : A.Q_1 \equiv P[Q/x]$. Thus $\Delta \vdash P'[Q/x] \hookrightarrow P[Q/x] : A \to B$.

- There is no difficulty for the other reduction rules since the reduction is compatible with the corresponding operators. □

3.2 Local Confluence

The following lemma claims local confluence for our reduction. Once we have established termination of the reduction, it allows us to prove confluence for the reduction. Actually, a bit more than local confluence is stated in the lemma. This is used to strengthen the induction hypothesis in the proof.

Lemma 3.4 (Local Confluence)
If $\Delta \vdash M \hookrightarrow N : A$ and $\Delta \vdash M \hookrightarrow P : A$ then there is a term Q such that $\Delta \vdash N \hookrightarrow_* Q : A$ and $\Delta \vdash P \hookrightarrow_* Q : A$. Moreover, if there is a basic η-expansion in either $\Delta \vdash N \hookrightarrow_* Q : A$ or $\Delta \vdash P \hookrightarrow_* Q : A$, then it is the last step of one of these sequences and the other sequence has no basic η-expansion step.



Proof By induction on the derivation of $\Delta \vdash M \hookrightarrow N : A$.

- (β) We assume that $\Delta \vdash (\lambda x : A.P \ \ Q) \hookrightarrow P[Q/x] : B$, with $\Delta, x : A \vdash P : B$ and $\Delta \vdash Q : A$. We have several possibilities for the other reduction step:

  - $\Delta \vdash (\lambda x : A.P \ \ Q) \hookrightarrow (\lambda x : A.P' \ \ Q) : B$ where $\Delta, x : A \vdash P \hookrightarrow P' : B$. Then $\Delta \vdash (\lambda x : A.P' \ \ Q) \hookrightarrow P'[Q/x] : B$ by β-reduction. Moreover by Lemma 3.3, we know that either $\Delta \vdash P[Q/x] \hookrightarrow P'[Q/x] : B$ or $\Delta \vdash P'[Q/x] \hookrightarrow P[Q/x] : B$.
  - $\Delta \vdash (\lambda x : A.P \ \ Q) \hookrightarrow (\lambda x : A.P \ \ Q') : B$ where $\Delta \vdash Q \hookrightarrow Q' : A$. Then $\Delta \vdash (\lambda x : A.P \ \ Q') \hookrightarrow_\beta P[Q'/x] : B$. By Lemma 3.2, there is a term R such that $\Delta \vdash P[Q/x] \hookrightarrow_* R : B$ and $\Delta \vdash P[Q'/x] \hookrightarrow_* R : B$. There may be an η-expansion in these sequences if $P \equiv x$ and Q is η-expanded into Q'.
  - $\Delta \vdash (\lambda x : A.P \ \ Q) \hookrightarrow (\lambda z : C.(\lambda x : A.P \ \ Q) \ \ z) : C \to D$, with $B = C \to D$. Then $\Delta \vdash \lambda z : C.(\lambda x : A.P \ \ Q) \ \ z \hookrightarrow \lambda z : C.(P[Q/x] \ \ z) : C \to D$. If $P[Q/x]$ is not an abstraction then $\Delta \vdash P[Q/x] \hookrightarrow \lambda z : C.(P[Q/x] \ \ z) : C \to D$ by η-expansion otherwise we have conversely: $\Delta \vdash \lambda z : C.(P[Q/x] \ \ z) \hookrightarrow P[Q/x] : C \to D$ via β-reduction and α-conversion.

- (Lam) We assume that $\Delta \vdash \lambda x : A.M \hookrightarrow \lambda x : A.N : A \to B$ and that $\Delta \vdash \lambda x : A.M \hookrightarrow \lambda x : A.P : A \to B$ (an abstraction can only be reduced if its body is reduced), where $\Delta, x : A \vdash M \hookrightarrow N : B$ and $\Delta, x : A \vdash M \hookrightarrow P : B$. By induction hypothesis, there is a term Q such that $\Delta, x : A \vdash N \hookrightarrow_* Q : B$ and $\Delta \vdash P \hookrightarrow_* Q : B$. Thus $\lambda x : A.N \hookrightarrow_* \lambda x : A.Q : A \to B$ and $\lambda x : A.P \hookrightarrow_* \lambda x : A.Q : A \to B$. There are no basic η-expansions in these sequences since all the reductions occur in the bodies of the abstractions.

- (App_r) We assume that $\Delta \vdash (M\ N) \hookrightarrow (M\ N') : B$. There are several possibilities for the other reduction step:

  - $\Delta \vdash (M\ N) \hookrightarrow (M\ N'') : B$. Then by induction hypothesis, there is a term Q such that $\Delta \vdash N' \hookrightarrow_* Q : A$ and $\Delta \vdash N'' \hookrightarrow_* Q : A$. Thus $\Delta \vdash (M\ N') \hookrightarrow_* (M\ Q) : B$ and $\Delta \vdash (M\ N'') \hookrightarrow_* (M\ Q) : B$ (no basic η-expansion).
  - The case $\Delta \vdash (M\ N) \hookrightarrow_\beta T : B$ has already been handled.
  - $\Delta \vdash (M\ N) \hookrightarrow \lambda z : C.(M\ N)\ z : C \to D$, with $B = C \to D$. Then we conclude easily: $\Delta \vdash (M\ N') \hookrightarrow \lambda z : C.(M\ N')\ z : C \to D$ and $\Delta \vdash \lambda z : C.(M\ N\ z) \hookrightarrow \lambda z : C.(M\ N'\ z) : C \to D$.
  - $\Delta \vdash (M\ N) \hookrightarrow (M'\ N) : B$. Then we have $\Delta \vdash (M\ N') \hookrightarrow (M'\ N') : B$ and $\Delta \vdash (M'\ N) \hookrightarrow (M'\ N') : B$ (no basic η-expansion) (Since we already have $\Delta \vdash (M\ N) \hookrightarrow (M'\ N) : B$, the reduction step $\Delta \vdash M \hookrightarrow M' : A \to B$ cannot be an η-expansion).

- (App_l) We assume that $\Delta \vdash (M\ N) \hookrightarrow (M'\ N) : B$. There are several possibilities for the other reduction step:

  - The case $\Delta \vdash (M\ N) \hookrightarrow T : B$ by β-reduction has already been handled.
  - The case $\Delta \vdash (M\ N) \hookrightarrow (M\ N') : B$ has already been handled in (App_r).
  - $\Delta \vdash (M\ N) \hookrightarrow \lambda z : C.(M\ N)\ z : C \to D$, with $B = C \to D$. Then $\Delta \vdash (M'\ N) \hookrightarrow \lambda z : C.(M'\ N)\ z : C \to D$ and $\Delta \vdash \lambda z : C.(M\ N)\ z \hookrightarrow \lambda z : C.(M'\ N)\ z : C \to D$ (since we already know that $\Delta \vdash (M\ N) \hookrightarrow (M'\ N) : B$).
  - $\Delta \vdash (M\ N) \hookrightarrow (M''\ N) : B$. Then we know that $\Delta \vdash M \hookrightarrow M' : A \to B$ and $\Delta \vdash M \hookrightarrow M'' : A \to B$ are not basic η-expansions. Thus by induction hypothesis there is a term Q such that $\Delta \vdash M' \hookrightarrow_* Q : A \to B$ and $\Delta \vdash M'' \hookrightarrow_* Q : A \to B$. If none of these sequences contain η-expansion steps then we are done since then $\Delta \vdash (M'\ N) \hookrightarrow (Q\ N) : B$ and $\Delta \vdash (M''\ N) \hookrightarrow (Q\ N) : B$. Otherwise by induction hypothesis we know that the unique η-expansion step is located at the end of one of the sequences. Let us assume that $\Delta \vdash M' \hookrightarrow_* M'_1 : A \to B$ without basic η expansion, $\Delta \vdash M'_1 \hookrightarrow_\eta \lambda z : A.(M'_1\ z) \equiv Q : A \to B$ for the first sequence and $\Delta \vdash M'' \hookrightarrow_* \lambda z : A.(M'_1\ z) \equiv Q : A \to B$ without basic η-expansion step for the other sequence. Then $\Delta \vdash (M''\ N) \hookrightarrow_* (\lambda z : A.(M'_1\ z)\ N) \hookrightarrow_\beta (M'_1\ N) : B$ and $\Delta \vdash (M'\ N) \hookrightarrow (M'_1\ N) : B$. There is no η-expansion in either sequence.

- (η) It has been partly handled. In fact, this case is rather easy to prove in its generality. We assume that $\Delta \vdash M \hookrightarrow \lambda z : A.(M\ z) : A \to B$ by η-abstraction and that $\Delta \vdash M \hookrightarrow N : A \to B$ (not an η-expansion). Thus $\Delta \vdash \lambda z : A.(M\ z) \hookrightarrow \lambda z : A.(N\ z) : A \to B$. We have two possibilities for N:
  - If N is not an abstraction then $\Delta \vdash N \hookrightarrow \lambda z : A.(N\ z) : A \to B$ by η-expansion.
  - If $N \equiv \lambda y : A.P$, then $\Delta \vdash \lambda z : A.(N\ z) \equiv \lambda z : A.(\lambda y : A.P\ z) \hookrightarrow_\beta \lambda z : A.P[z/y] : A \to B$, which is α-convertible to $\lambda y : A.P$, i.e. N.
- ($\uparrow$) We assume that $\Delta \vdash\uparrow M \hookrightarrow\uparrow N : \Box A$. We have two kinds of reduction from $\uparrow M$:
  - If $\Delta \vdash\uparrow M \hookrightarrow\uparrow P : \Box A$ then by induction hypothesis there is a term Q such that $\Delta;. \vdash N \hookrightarrow_* Q : A$ and $\Delta;. \vdash P \hookrightarrow_* Q : A$. Thus $\Delta \vdash\uparrow N \hookrightarrow_*\uparrow Q : \Box A$ and $\Delta \vdash\uparrow P \hookrightarrow_*\uparrow Q : \Box A$
  - If $M \equiv\downarrow M_1$ then we can have $\Delta \vdash\uparrow\downarrow M_1 \hookrightarrow M_1 : \Box A$ by rule ($\eta\Box$). Here we have two subcases:
    - if $M_1 \equiv\uparrow M'_1$ and if the two reduction steps are $\Delta \vdash\uparrow\downarrow\uparrow M'_1 \hookrightarrow\uparrow M'_1 : \Box A$ by rule ($\uparrow$) and $\Delta \vdash\uparrow\downarrow\uparrow M'_1 \hookrightarrow\uparrow M'_1 : \Box A$ by rule ($\eta\Box$) then there is nothing more to say: both reduction steps give the same term.
    - if the two reduction steps are $\Delta \vdash\uparrow\downarrow M_1 \hookrightarrow\uparrow\downarrow M'_1 : \Box A$ and $\Delta \vdash\uparrow\downarrow M_1 \hookrightarrow M_1 : \Box A$. Then it is easy to see that $\Delta \vdash\uparrow\downarrow M'_1 \hookrightarrow M'_1 : \Box A$ and $\Delta \vdash M_1 \hookrightarrow M'_1 : \Box A$
- ($\eta\Box$) like the previous case.
- ($\downarrow$) ($\beta\Box$) and (Pop) are easy.
- The cases of the rules (Π_1), (Π_2), (sp), (Pair_l), (Pair_r), (Fst) and (Snd) are similar to the cases of the modal reduction rules.
- (Case) is not difficult but rather long to detail. Let us show what the proof looks like by treating one of the cases. Let us assume that in Δ, $\langle\sigma\rangle\text{Case}\ \uparrow \lambda\vec{x} : \vec{T}.(C_{j_0,k_0}\ \vec{M})\ \text{ of }\ (M_{j,k})$ reduces to:

  $$\lambda\vec{u} : \vec{T'}.(M_{j_0,k_0}\ \ \lambda\vec{x} : \vec{T}.M_1 \ldots \lambda\vec{x} : \vec{T}.M_p)$$

  and to:

  $$\langle\sigma\rangle\text{Case}\ \uparrow \lambda\vec{x} : \vec{T}.(C_{j_0,k_0}\ M_0 \ldots M'_k \ldots M_p)\ \text{ of }\ (M_{j,k})$$

  Then both terms reduce to $\lambda\vec{u} : \vec{T'}.(M_{j_0,k_0}\ \ \lambda\vec{x} : \vec{T}.M_1 \ldots \lambda\vec{x} : \vec{T}.M'_k \ldots \lambda\vec{x} : \vec{T}.M_p))$.
- (It) is similar to (Case). We only show one of the cases. We assume that in stack Δ, $\langle\sigma\rangle\text{It}\ \uparrow \lambda\vec{x} : \vec{T}.(C_{j_0,k_0}\ \vec{M})\ \text{ of }\ (N_{j,k})$ reduces to:

  $\lambda\vec{u} : \sigma(\vec{T}).(N_{j_0,k_0}\ \ (\langle\sigma\rangle\text{It}\ \uparrow \lambda\vec{x} : \vec{T}.M_1\ \text{ of }\ (N_{j,k})\ \vec{u}) \ldots (\langle\sigma\rangle\text{It}\ \uparrow \lambda\vec{x} : \vec{T}.M_{n_{j,k}}\ \text{ of }\ (N_{j,k})\ \vec{u}))$

and to:

$$\langle\sigma\rangle\mathrm{It}\ \uparrow\lambda\vec{x}:\vec{T}.(C_{j_0,k_0}\ \vec{M})\ \text{ of }\ (N'_{j,k})$$

Then both terms reduce to:

$$\lambda\vec{u}:\sigma(\vec{T}).(N'_{j_0,k_0}\ \ (\langle\sigma\rangle\mathrm{It}\ \uparrow\lambda\vec{x}:\vec{T}.M_1\ \text{ of }\ (N_{j,k})\ \vec{u})\ldots(\langle\sigma\rangle\mathrm{It}\ \uparrow\lambda\vec{x}:\vec{T}.M_{n_{j,k}}\ \text{ of }\ (N'_{j,k})\ \vec{u}))$$

if N'_{j_0,k_0} is not the η-expansion of N_{j_0,k_0} and otherwise to:

$$\lambda\vec{u}:\sigma(\vec{T}).(N_{j_0,k_0}\ \ (\langle\sigma\rangle\mathrm{It}\ \uparrow\lambda\vec{x}:\vec{T}.M_1\ \text{ of }\ (N_{j,k})\ \vec{u})\ldots(\langle\sigma\rangle\mathrm{It}\ \uparrow\lambda\vec{x}:\vec{T}.M_{n_{j,k}}\ \text{ of }\ (N'_{j,k})\ \vec{u}))$$

□

3.3 Strong Normalization

Now we give a proof of the Strong Normalization theorem for our system, i.e. we prove that there is no infinite sequence of reductions starting from a well-typed term. The proof follows the sketch of normalization proofs 'à la Tait' and is inspired by [Wer94] (for the inductive part) and [Gha96] (for the η-expansion part).

3.3.1 Reducibility Candidates

First we give a definition of the reducibility candidates ([GLT89]) adapted to our setting. Let us call Λ the set of our terms, defined in Section 2.2.

Definition 3.5 (Reducibility Candidates) *Given a type A, the* candidates of reducibility CR_A *are sets of pairs (Δ, M) of a context stack and a term. They are defined as follows:*

CR1 $\forall(\Delta,M)\in\mathcal{C}$, *$M$ is strongly normalizing in Δ (i.e. there is no infinite sequence of reductions starting from M in Δ).*

CR1' $\mathcal{C}\subset\{(\Delta,M)\mid\Delta\vdash M:A\}$

CR2 $\forall(\Delta,M)\in\mathcal{C}$ *s.t.* $\Delta\vdash M\hookrightarrow M':A$, *we have* $(\Delta,M')\in\mathcal{C}$.

CR3 *If* $M\in\mathcal{NT}$, $\Delta\vdash M:A$ *and* $\forall M'$ *s. t.* $\Delta\vdash M\hookrightarrow M':A$ (*not an η-expansion*), $(\Delta,M')\in\mathcal{C}$ *then we have* $(\Delta,M)\in\mathcal{C}$.

CR4 *If* $A=B\to C$ *and* $(\Delta,M)\in\mathcal{C}$ *then* $(\Delta,\lambda z:B.(M\ \ z))\in\mathcal{C}$, *where z is a fresh variable.*

where $\mathcal{NT}=\Lambda\setminus(\{\lambda x:A.M\mid M\in\Lambda\}\cup\{\uparrow M\mid M\in\Lambda\}\cup\{\langle M,N\rangle\mid M,N\in\Lambda\})$.
The set $\mathcal{NT}$ is called the set of the neutral *terms.*

Notice that instead of taking sets of terms, we consider sets of pairs of a stack and a term. Indeed, since our reduction is typed, it is convenient for the reducibility candidates to contain well-typed terms.

In rule (CR3), the restriction "$\Delta \vdash M \hookrightarrow M' : A$ is not an η-expansion" comes from [JG95]. It has been introduced to cope with η-expansions. The rule (CR4) is also needed because of the η-expansions ([Gha96]).

Our version of reducibility candidate enjoys the usual properties:

Proposition 3.6 (Intersection of Reducibility Candidates)
If $\mathcal{C}$ and $\mathcal{D}$ belong to CR_A then $\mathcal{C} \cap \mathcal{D}$ belong to CR_A.

Since the proof is straightforward, we omit it here. An important consequence of this property is that CR_A is an *inf-semi lattice*. Next, we define the set $\mathcal{C} \to \mathcal{D}$, where $\mathcal{C}$ and $\mathcal{D}$ are two reducibility candidates.

Definition 3.7 ($\mathcal{C} \to \mathcal{D}$)
Given $\mathcal{C} \in CR_A$ and $\mathcal{D} \in CR_B$,
$\mathcal{C} \to \mathcal{D} := \{(\Delta, M) \mid \Delta \vdash M : A \to B$ *and* $\forall \Gamma$ *context*, $\forall ((\Delta, \Gamma), N) \in \mathcal{C}, ((\Delta, \Gamma), (M\ \ N)) \in \mathcal{D}\}$

In this definition, the context Γ added to the stack is primordial. In the next lemma it allows us to add fresh variables to the context.

Notation Given any strongly normalizing term M and any stack Δ, $\nu_\Delta(M)$ denotes the length of the greatest sequence of one-step reductions starting from M in Δ.

Lemma 3.8 *Let $\mathcal{C} \in CR_A$ and $\mathcal{D} \in CR_B$. If $\Delta, x : A \vdash M : B$ and if forall Γ context, forall $((\Delta, \Gamma), N) \in \mathcal{C}$, $((\Delta, \Gamma), M[N/x]) \in \mathcal{D}$ then $(\Delta, \lambda x : A.M) \in \mathcal{C} \to \mathcal{D}$.*

Proof Because of condition (CR3), we notice that $((\Delta, x : A), x : A) \in \mathcal{C}$ (x is neutral and only an η-expansion can possibly reduce it). Thus $((\Delta, x : A), M[x/x]) \in \mathcal{D}$ and M is SN in Δ.

We prove by induction on $\nu_\Delta(M) + \nu_\Delta(N)$ that if forall $((\Delta, \Gamma), N) \in \mathcal{C}$, $((\Delta, \Gamma), M[N/x]) \in \mathcal{D}$ then forall $((\Delta, \Gamma), N) \in \mathcal{C}$, $((\Delta, \Gamma), (\lambda x : A.M\ \ N)) \in \mathcal{D}$, which by definition will allow us to conclude that $(\Delta, \lambda x : A.M) \in \mathcal{C} \to \mathcal{D}$. We will use (CR3) and show that all the reducts of the neutral term $(\lambda x : A.M\ \ N)$ (except basic η-expansions) belong to $\mathcal{D}$. There are several ways to reduce this term:

- $\Delta, \Gamma \vdash (\lambda x : A.M\ \ N) \hookrightarrow (\lambda x : A.M\ \ N') : B$. $\nu_\Delta(N') < \nu_\Delta(N)$, we apply the induction hypothesis.
- $\Delta, \Gamma \vdash (\lambda x : A.M\ \ N) \hookrightarrow (\lambda x : A.M'\ \ N) : B$. We have to establish the induction hypothesis for M', namely that forall $((\Delta, \Gamma), N) \in \mathcal{C}$, $((\Delta, \Gamma), M'[N/x]) \in \mathcal{D}$.

  By Lemma 3.3, there are two possibilities of reductions: we have either $\Delta, \Gamma \vdash M[N/x] \hookrightarrow_* M'[N/x] : B$ (in this case we use Property (CR2)) or $\Delta, \Gamma \vdash M'[N/x] \hookrightarrow M[N/x] : B$. In

the latter case, N is an abstraction and M' is obtained from M by η-expanding the variable 'x'. Then we also have $\Delta,\Gamma \vdash M[\lambda z : C.(N\ z)/x] \hookrightarrow_* M'[N/x] : B$ (where z is a fresh variable), β-reducing in $M[\lambda z : C.(N\ z)/x]$ the redex $\lambda z : C.(N\ z)$ for all the occurrences of 'x' except one. Now, since $((\Delta,\Gamma),N) \in \mathcal{C}$, $((\Delta,\Gamma),\lambda z : C.(N\ z)) \in \mathcal{C}$ by (CR4) and thus $((\Delta,\Gamma),M[\lambda z : C.(N\ z)]) \in \mathcal{D}$. By (CR2), $((\Delta,\Gamma),M'[N/x]) \in \mathcal{D}$. In all the cases, the induction hypothesis is valid for M'.

- $\Delta,\Gamma \vdash (\lambda x : A.M\ \ N) \hookrightarrow M[N/x] : B$, $M[N/x] \in \mathcal{D}$ by hypothesis. □

Now, we can prove that the set $\mathcal{C} \to \mathcal{D}$ is actually a reducibility candidate.

Proposition 3.9 ($\mathcal{C} \to \mathcal{D}$ is a CR)
If $\mathcal{C} \in CR_A$ and $\mathcal{D} \in CR_B$ then $\mathcal{C} \to \mathcal{D} \in CR_{A\to B}$

Proof We show that $\mathcal{C} \to \mathcal{D}$ verifies all the properties C.R.

(CR1) We want to prove that if (Δ, M) belongs to $\mathcal{C} \to \mathcal{D}$ then M is SN in Δ. We first notice that for any variable x not free in Δ, $((\Delta, x : A), x)$ belongs to any reducibility candidate of CR_A thanks to condition (CR3). Thus if (Δ, M) is in $\mathcal{C} \to \mathcal{D}$, $((\Delta, x : A), (M\ \ x))$ is in $\mathcal{D}$. It implies that $(M\ \ x)$ is SN in stack $\Delta, x : A$.

We show by induction on $\nu_{\Delta,x:A}((M\ \ x))$ that if $(M\ \ x)$ is SN in $\Delta, x : A$ ($x \notin \mathrm{FV}(M)$) then M is SN in Δ. More precisely, we examine the reduced terms of M and prove that they are all SN.

- If $\Delta \vdash M \hookrightarrow M' : A \to B$ by a reduction step different from an η-expansion step then $\Delta, x : A \vdash (M\ \ x) \hookrightarrow (M'\ \ x) : B$. Since $(M\ \ x)$ is SN, $(M'\ \ x)$ is SN too and by induction hypothesis M' is SN.
- Otherwise $\Delta \vdash M \hookrightarrow_\eta \lambda z : A.(M\ \ z) : A \to B$, where 'z' is a fresh variable. The term $\lambda z : A.(M\ \ z)$ is SN since all the following reductions will take place in the body of the abstraction and $(M\ \ z)$ is SN (since $(M\ x)$ is SN and $(M\ x)$ and $(M\ z)$ are α-convertible).

(CR1') By definition

(CR2) If $(\Delta, M) \in \mathcal{C} \to \mathcal{D}$ and $\Delta \vdash M \hookrightarrow M' : A \to B$, we have two cases:

- If the reduction step is not an η-expansion then forall $((\Delta,\Gamma), N)$ in $\mathcal{C}$, we have $\Delta,\Gamma \vdash (M\ \ N) \hookrightarrow (M'\ \ N) : B$. By (CR2), $(\Delta,\Gamma,(M'\ \ N)) \in \mathcal{D}$ thus $(\Delta, M') \in \mathcal{C} \to \mathcal{D}$.
- If the reduction step is an η-expansion $\Delta \vdash M \hookrightarrow \lambda z : A.(M\ \ z) : A \to B$ (where the variable z is fresh), the result follows from Lemma 3.8. Indeed in our case the hypothesis of Lemma 3.8 becomes 'if $((\Delta,\Gamma),N) \in \mathcal{C}$ then $((\Delta,\Gamma),(M\ \ z)[N/z]) \equiv ((\Delta,\Gamma),(M\ \ N)) \in \mathcal{D}$', which is obviousy true since M belongs to $\mathcal{C} \to \mathcal{D}$.

(CR3) Let $M \in \mathcal{NT}$ and Δ verifying the hypotheses of condition (CR3), let $((\Delta,\Gamma),N) \in \mathcal{C}$. We prove by induction on $\nu_{\Delta,\Gamma}(N)$ using (CR3) that the pair $((\Delta,\Gamma),(M\ \ N))$ belongs to $\mathcal{D}$. Indeed, $(M\ \ N) \in \mathcal{NT}$ and

- either $\Delta,\Gamma \vdash (M\ \ N) \hookrightarrow (M'\ \ N) : B$ and since $M' \in \mathcal{C} \to \mathcal{D}$, $(M'\ \ N) \in \mathcal{D}$
- or $\Delta,\Gamma \vdash (M\ \ N) \hookrightarrow (M\ \ N') : B$ and we apply the induction hypothesis.

There is no other possible reduction step, except η-expansion, because M cannot be an abstraction.

(CR4) Given $(\Delta, M) \in \mathcal{C} \to \mathcal{D}$, we want to prove that $(\Delta, \lambda z : A.(M\ z)) \in \mathcal{C} \to \mathcal{D}$. We use Lemma 3.8. By definition of $\mathcal{C} \to \mathcal{D}$, forall $((\Delta,\Gamma),N) \in \mathcal{C}$, $((\Delta,\Gamma),(M\ N)) \in \mathcal{D}$, that is to say $((\Delta,\Gamma),(M\ z)[N/z]) \in \mathcal{D}$. Lemma 3.8 allows us to conclude that $(\Delta, \lambda z : A.(M\ z)) \in \mathcal{C} \to \mathcal{D}$. □

Now we define two useful operations on reducibility candidates.

Definition 3.10 ($\Box\mathcal{C}$)
If $\mathcal{C} \in CR_A$ then we define $\Box\mathcal{C}$ as the following set $\Box\mathcal{C} := \{(\Delta, M) \mid \Delta \vdash M : \Box A$ *and* $\forall \Delta'$ *stack compatible with* Δ , $((\Delta,\Delta'), \downarrow M) \in \mathcal{C}\}$.

Proposition 3.11 ($\Box\mathcal{C}$ is a CR)
If $\mathcal{C} \in CR_A$ then $\Box\mathcal{C} \in CR_{\Box A}$.

Proof

(CR1) Obvious : $((\Delta,\Delta'), \downarrow M) \in \mathcal{C}$ thus $\downarrow M$ is SN in Δ,Δ'. This implies that M is SN in Δ because the reduction is compatible with operator $\downarrow$.

(CR1') By definition

(CR2) Let us assume that $M \in \Box\mathcal{C}$ and $\Delta \vdash M \hookrightarrow M' : \Box A$. Then $\Delta \vdash M' : \Box A$ and $\Delta,\Delta' \vdash \downarrow M \hookrightarrow \downarrow M' : A$. Thus $((\Delta,\Delta'), \downarrow M') \in \mathcal{C}$ and $M' \in \Box\mathcal{C}$.

(CR3) Let $M \in \mathcal{NT}$ and Δ verifying the hypotheses of condition (CR3) (i.e. $\forall M'$ s.t. $\Delta \vdash M \hookrightarrow M' : \Box A, (\Delta, M') \in \Box\mathcal{C})$, we want to prove that for any Δ' such that Δ,Δ' is valid, $((\Delta,\Delta'), \downarrow M) \in \mathcal{C}$. We use condition (CR3). If $\Delta,\Delta' \vdash \downarrow M \hookrightarrow N : A$ by a reduction step different from an η-expansion, it means that $N \equiv \downarrow M'$ and $\Delta,\Delta' \vdash M \hookrightarrow M' : \Box A$ (since M is neutral, it cannot be of the form $\uparrow M_1$). By hypothesis, $(\Delta, M') \in \Box\mathcal{C}$, hence $((\Delta,\Delta'), \downarrow M') \in \mathcal{C}$ by definition of $\Box\mathcal{C}$. Thus all the reducts of $\downarrow M$ belong to $\mathcal{C}$, $((\Delta,\Delta'), \downarrow M) \in \mathcal{C}$ and finally $(\Delta, M) \in \Box\mathcal{C}$.

(CR4) $\Box A$ is not of the form $B \to C$. □

Definition 3.12 ($\mathcal{C} \times \mathcal{D}$)
If $\mathcal{C} \in CR_A$ and $\mathcal{D} \in CR_B$ then we define $\mathcal{C} \times \mathcal{D}$ as the following set $\mathcal{C} \times \mathcal{D} := \{(\Delta, M) \mid \Delta \vdash M : A \times B$ *and* $\forall \Gamma$ *context compatible with* Δ , $((\Delta,\Gamma), fst\ M) \in \mathcal{C}$ *and* $((\Delta,\Gamma), snd\ M) \in \mathcal{D}\}$.

Proposition 3.13 ($\mathcal{C} \times \mathcal{D}$ **is a CR**)
If $\mathcal{C} \in CR_A$ *and* $\mathcal{D} \in CR_B$ *then* $\mathcal{C} \times \mathcal{D} \in CR_{A \times B}$.

The proof is similar to the one of Proposition 3.11. We omit it here.

3.3.2 Interpretation of types

Following the sketch of normalization proofs 'à la Tait', we define the interpretations of types $[\![T]\!]$.

Definition 3.14 ($[\![L_j]\!]$)
For any ground type L_j, *we define* $[\![L_j]\!]$ *as the set* $\{(\Delta, M) \mid \Delta \vdash M : L_j$ *and* M *is SN in* $\Delta\ \}$.

Proposition 3.15 ($[\![L_j]\!]$ **is a C.R.**)
The set $[\![L_j]\!]$ *is a reducibility candidate.*

Proof
(CR1), (CR1'), (CR2) and (CR4) are immediate. (CR3) is easy: if M is neutral and if forall M' such that $\Delta \vdash M \hookrightarrow M' : L_j$ without η-expansion, $M' \in [\![L_j]\!]$, then we notice that actually all the reducts of M belong to $[\![L_j]\!]$ since M (of ground type L_j) cannot be η-expanded. Thus M is SN in Δ and $M \in [\![L_n]\!]$. □

Now, given the interpretations of types A and B, we define the interpretation of type $A \to B$.

Definition 3.16 ($[\![A \to B]\!]$)
Given A *and* B *two types,* $[\![A \to B]\!] := [\![A]\!] \to [\![B]\!]$, *i.e.* $[\![A \to B]\!] := \{(\Delta, M) \mid \Delta \vdash M : A \to B$ *and* $\forall \Gamma, \forall((\Delta, \Gamma), N) \in [\![A]\!], ((\Delta, \Gamma), (M\ \ N)) \in [\![B]\!]\}$

Proposition 3.17 ($[\![A \to B]\!]$ **is a C.R.**)
If $[\![A]\!]$ *and* $[\![B]\!]$ *are reducibility candidates,* $[\![A \to B]\!]$ *is a reducibility candidate.*

Proof
Immediate by Proposition 3.9. □

So far, we have defined $[\![A]\!]$ for any pure type A. It is easy to see that for any pure type A, $[\![A]\!]$ is a reducibility candidate. Now, we define $[\![A \times B]\!]$ and $[\![\Box A]\!]$.

Definition 3.18 ($[\![A \times B]\!]$)
Given A *and* B *two types,* $[\![A \times B]\!] := [\![A]\!] \times [\![B]\!]$, *i.e.* $[\![A \times B]\!]$ *is the set* $\{(\Delta, M) \mid \Delta \vdash M :$ $A \times B$ *and* $\forall \Gamma$ *compatible with* Δ $, ((\Delta, \Gamma), fst\ M) \in [\![A]\!]$ *and* $((\Delta, \Gamma), snd\ M) \in [\![B]\!]\}$

Proposition 3.19 ($[\![A \times B]\!]$ **is a C.R.**)
Given A *and* B *two types, if* $[\![A]\!]$ *and* $[\![B]\!]$ *are two reducibility candidates, then* $[\![A \times B]\!]$ *is a reducibility candidate.*

Proof
Immediate by Proposition 3.13. □

For $[\![\Box A]\!]$, we have two cases, depending on the form of type A. If A is not pure, $\Box A$ cannot be the type of the main argument of a Case/It construct. The only reduction we have to deal with is the modal one.

Definition 3.20 ($[\![\Box A]\!]$, A not pure)
If A is a type that is not pure, $[\![\Box A]\!] := \Box[\![A]\!]$, i.e. $[\![\Box A]\!]$ is the set $\{(\Delta, M) \mid \Delta \vdash M : \Box A$ and $\forall \Delta'$ stack compatible with $\Delta, ((\Delta, \Delta'), \downarrow M) \in [\![A]\!]\}$

Proposition 3.21 ($[\![\Box A]\!]$, where A is not pure, is a C.R.)
Given A a type that is not pure, if $[\![A]\!]$ is a reducibility candidate, then $[\![\Box A]\!]$ is a reducibility candidate.

Proof
Immediate by Proposition 3.11. □

If A is a pure type ($A = T_1 \to \cdots \to T_p \to L$), the definition of $[\![\Box A]\!]$ is a bit more difficult to write because we have to take into account the fact that $\Box A$ may be the type of the inductive argument of Case/It.

We would like $[\![\Box(T_1 \to \ldots T_p \to L)]\!]$ to be a C.R. such that:

$$
\begin{aligned}
&(\Delta, M) \in [\![\Box(T_1 \to \ldots T_p \to L)]\!] \text{ if} \\
&\qquad \Delta \vdash M : \Box(T_1 \to \ldots T_p \to L) \text{ and} \\
&\qquad \forall \Delta' \text{ stack such that } \Delta, \Delta' \text{ is valid ,} \\
&\qquad \forall \sigma, \text{map from ground types to types,} \\
&\qquad \forall \tau, \text{map mapping each ground type } L_j \text{ to a C.R.}_{\sigma(L_j)}, \\
&\qquad \forall ((\Delta, \Delta'), M_{j,k}) \in \prod_{q=1}^{q=n_{j,k}} \left[\Box(\prod_{i=1}^{i=p} T_i.B_{j,k,q})\right].\tau(L_j), \\
&\qquad \forall ((\Delta, \Delta'), {M'}_{j,k}) \in \prod_{q=1}^{q=n_{j,k}} \tau(B_{j,k,q}).\tau(L_j), \\
&\qquad ((\Delta, \Delta'), \langle\sigma\rangle \text{Case } M \text{ of } (M_{j,k})) \in \prod_{z=1}^{z=p} [\![{T'}_z]\!].\tau(L), \\
&\qquad ((\Delta, \Delta'), \langle\sigma\rangle \text{It } M \text{ of } ({M'}_{j,k})) \in \prod_{i=1}^{i=p} \tau(T_i).\tau(L), \\
&\qquad \text{and } ((\Delta, \Delta'), \downarrow M) \in [\![T_1 \to \ldots T_p \to L]\!]
\end{aligned}
$$

where $[\![{T'}_z]\!]$ is here a shortcut for:

$$[\![\Box(T_1 \to \cdots \to T_p \to T_z^1)]\!] \to \cdots \to [\![\Box(T_1 \to \cdots \to T_p \to T_z^{r_z})]\!] \to \tau(L_z)$$

or equivalently

$$\prod_{m=1}^{m=r_z} \left[\!\left[\Box\left(\prod_{i=1}^{i=p} T_i.T_z^m\right)\right]\!\right].\tau(L_z)$$

Let us call $\mathcal{S}$ the set of the total functions that map a pure type A to a C.R. for type A. We define the operator F from $\mathcal{S}$ to $\mathcal{S}$ by giving $F(g)\left(\prod_{i=1}^{i=p} T_i.L\right)$ for $g \in \mathcal{S}$, where T_i are pure types and L is a ground type:

$$\bigcap_{\sigma,\tau} \left\{ \begin{array}{l} (\Delta, M) \mid \Delta \vdash M : \Box(T_1 \to \ldots T_p \to L) \text{ and} \\ \forall \Delta' \text{ such that } \Delta, \Delta' \text{ is valid ,} \\ \forall((\Delta,\Delta'), M_{j,k}) \in \prod_{q=1}^{q=n_{j,k}} g\left(\prod_{i=1}^{i=p} T_i.B_{j,k,q}\right).\tau(L_j), \\ \forall((\Delta,\Delta'), M'_{j,k}) \in \prod_{q=1}^{q=n_{j,k}} \tau(B_{j,k,q}).\tau(L_j), \\ ((\Delta,\Delta'), \langle\sigma\rangle\text{Case } M \text{ of } (M_{j,k})) \in \prod_{z=1}^{z=p}\left(\prod_{m=1}^{m=r_z} g\left(\prod_{i=1}^{i=p} T_i.T_z^m\right).\tau(L_z)\right).\tau(L), \\ ((\Delta,\Delta'), \langle\sigma\rangle\text{It } M \text{ of } (M'_{j,k})) \in \prod_{i=1}^{i=p} \tau(T_i).\tau(L), \\ \text{and } ((\Delta,\Delta'), \downarrow M) \in [\![T_1 \to \ldots T_p \to L]\!] \end{array} \right\}$$

where σ is quantified over the total functions from ground types to types and τ is quantified over the total functions mapping each ground type L_j to a C.R.$_{\sigma(L_j)}$.

Of course we have to prove that F is really an operator from $\mathcal{S}$ to $\mathcal{S}$, i.e. that for any g belonging to $\mathcal{S}$, $F(g)$ belongs to $\mathcal{S}$. This is the purpose of the next proposition. Then we will prove another key point : F is monotone. This important property will allow us to give a rigorous definition of $[\![\Box A]\!]$ when A is pure, involving the smallest fixpoint of F.

Proposition 3.22 (F is an operator from $\mathcal{S}$ to $\mathcal{S}$)
If g belongs to $\mathcal{S}$ then $F(g)$ belongs to $\mathcal{S}$, i.e. for any pure type A, $F(g)(A)$ is a C.R.

Proof

We assume that $A = T_1 \to \cdots \to T_p \to L$, where the types T_i are pure and L is a ground type. We assume that the set $g(T_1 \to \cdots \to T_p \to L)$ is a C.R. We have to show that $F(g)(T_1 \to \cdots \to T_p \to L)$ is a C.R. too:

(CR1) Obvious since $((\Delta,\Delta'), \downarrow M) \in [\![T_1 \to \ldots T_p \to L]\!]$.

(CR1') By definition.

(CR2) If $(\Delta, M) \in F(g)(T_1 \to \cdots \to T_p \to L)$ and $\Delta \vdash M \hookrightarrow M' : \Box(T_1 \to \cdots \to T_p \to L)$ then $\Delta \vdash M' : \Box(T_1 \to \cdots \to T_p \to L)$. Moreover we have the following reductions:

- $\Delta, \Delta' \vdash \downarrow M \hookrightarrow \downarrow M' : T_1 \to \cdots \to T_p \to L$,
- $\Delta, \Delta' \vdash \langle\sigma\rangle\text{Case } M \text{ of } (M_{j,k}) \hookrightarrow \langle\sigma\rangle\text{Case } M' \text{ of } (M_{j,k}) : \prod_{z=1}^{z=p} T'_z.\sigma(L)$

- and $\Delta, \Delta' \vdash \langle\sigma\rangle \text{It } M \text{ of } (M'_{j,k}) \hookrightarrow \langle\sigma\rangle \text{It } M' \text{ of } (M'_{j,k}) : \prod_{i=1}^{i=p} \sigma(T_i).\sigma(L)$.

Thus,

- $((\Delta, \Delta'), \downarrow M) \in [\![T_1 \to \dots T_p \to L]\!]$,
- $((\Delta, \Delta'), \langle\sigma\rangle \text{Case } M' \text{ of } (M_{j,k})) \in \prod_{z=1}^{z=p} \left(\prod_{m=1}^{m=r_z} g \left(\prod_{i=1}^{i=p} T_i.T_z^m \right) .\tau(L_z) \right) .\tau(L)$
- $((\Delta, \Delta'), \langle\sigma\rangle \text{It } M' \text{ of } (M'_{j,k})) \in \prod_{i=1}^{i=p} \tau(T_i).\tau(L)$

This implies that $(\Delta, M') \in F(g)(T_1 \to \dots \to T_p \to L)$.

(CR3) The principle is the same as in the proof of Proposition 3.11. We assume that (Δ, M) verifies the hypotheses of condition (CR3). We show that $M \in F(g)(T_1 \to \dots \to T_p \to L)$ by proving that:

- $((\Delta, \Delta'), \langle\sigma\rangle \text{Case } M' \text{ of } (M_{j,k})) \in \prod_{z=1}^{z=p} \left(\prod_{m=1}^{m=r_z} g \left(\prod_{i=1}^{i=p} T_i.T_z^m \right) .\tau(L_z) \right) .\tau(L)$
- $((\Delta, \Delta'), \langle\sigma\rangle \text{It } M \text{ of } (M'_{j,k})) \in \prod_{i=1}^{i=p} \tau(T_i).\tau(L)$, and
- $((\Delta, \Delta'), \downarrow M) \in [\![T_1 \to \dots T_p \to L]\!]$.

Let us show what the proof looks like for $\langle\sigma\rangle \text{Case } M \text{ of } (M_{j,k}) \in \mathcal{NT}$. We prove that the set $\prod_{z=1}^{z=p} \left(\prod_{m=1}^{m=r_z} g \left(\prod_{i=1}^{i=p} T_i.T_z^m \right) .\tau(L_z) \right) .\tau(L)$ contains all its reducts (except those obtained by η-expansion). Indeed there are two ways to reduce $\langle\sigma\rangle \text{Case } M \text{ of } (M_{j,k})$ (Since (Δ, M) verifies the hypotheses of (CR3), M is neutral and cannot be of the form $\uparrow M_1$. Thus, the Case construct itself cannot be eliminated):

- $\Delta, \Delta' \vdash \langle\sigma\rangle \text{Case } M \text{ of } (M_{j,k}) \hookrightarrow \text{Case } M' \text{ of } (M_{j,k}) : \prod_{z=1}^{z=p} T'_z.\sigma(L)$
- $\Delta, \Delta' \vdash \langle\sigma\rangle \text{Case } M \text{ of } (M_{j,k}) \hookrightarrow \text{Case } M \text{ of } (M_{j,k})' : \prod_{z=1}^{z=p} T'_z.\sigma(L)$

and we prove the property by induction on $\sum_{j,k} \nu(M_{j,k})$.

(CR4) The type $\square(T_1 \to \dots \to T_p \to L)$ is not of the form $B \to C$. □

Lemma 3.23 (Monotonocity of F)
F is monotone.

Proof
If we assume that for any pure type A, $g(A) \subseteq g'(A)$ then:

$$\prod_{q=1}^{q=n_{j,k}} g'\left(\prod_{i=1}^{i=p} T_i.B_{j,k,q}\right).\tau(L) \subseteq \prod_{q=1}^{q=n_{j,k}} g\left(\prod_{i=1}^{i=p} T_i.B_{j,k,q}\right).\tau(L)$$

and similarly,

$$\prod_{m=1}^{m=r_z} g'\left(\prod_{i=1}^{i=p} T_i.T_z^m\right).\tau(L_z) \subseteq \prod_{m=1}^{m=r_z} g\left(\prod_{i=1}^{i=p} T_i.T_z^m\right).\tau(L_z)$$

Thus, for any pure type A, if $g(A) \subseteq g'(A)$, then we have: $F(g)(A) \subseteq F(g')(A)$. □

Definition 3.24 (Definition of $[\![A]\!]$, A pure)
We call Y_F the smallest fixpoint of F. By definition, for any pure type A, $[\![A]\!] := Y_F(A)$.

Of course, we have to justify this definition. First, we notice that $\mathcal{S}$ is an inf-semi lattice because the set of C.R. for a given type is an inf-semi lattice. Now, the existence of Y_F is proved by Tarski's lemma: F is a monotone operator from the inf-semi lattice $\mathcal{S}$ to the inf-semi lattice $\mathcal{S}$. Note that $[\![A]\!]$, image of A by Y_F is itself a C.R.

At this point, we have defined the interpretation of type $[\![A]\!]$ for all the types A. The following theorem is an immediate corollary of previous lemmas.

Theorem 3.25 ($[\![A]\!]$ is a C.R.)
Given any type A, the set $[\![A]\!]$ is a C.R.

Proof
By induction on A using Lemmas 3.15, 3.17, 3.19, 3.21 and the justification of Definition 3.24.
□

Now we state two lemmas which show that the interpretations of types, as well as typing judgments and reduction judgments, are invariant by certain manipulations of stack.

Proposition 3.26 (Weakening of $[\![A]\!]$)
If A is a type, $(\Delta, M) \in [\![A]\!]$ and Δ, Γ is valid then $((\Delta, \Gamma), M) \in [\![A]\!]$.

Proof We examine the different cases:

- If A is a ground type and $(\Delta, M) \in [\![A]\!]$, then $\Delta, \Gamma \vdash M : A$ because $\Delta \vdash M : A$ and M is SN in Δ, Γ because it is SN in Δ.
- If $A = C \to D$ then $\Delta, \Gamma \vdash M : C \to D$ since $\Delta \vdash M : C \to D$. If $((\Delta, \Gamma, \Gamma'), N) \in [\![A]\!]$ then $((\Delta, \Gamma, \Gamma'), (M\ \ N)) \in [\![B]\!]$ by definition of $[\![A \to B]\!]$.

- If $A = C \times D$ then $\Delta, \Gamma \vdash M : C \times D$ since $\Delta \vdash M : C \times D$. If $(\Delta, \Gamma, \Gamma')$ is valid, then $((\Delta, \Gamma, \Gamma'), \text{fst } M) \in [\![C]\!]$ and $((\Delta, \Gamma, \Gamma'), \text{snd } M) \in [\![D]\!]$ by definition of $[\![C \times D]\!]$.
- If $A \equiv \Box C$, where C is not pure, then $\Delta, \Gamma \vdash M : \Box C$ since $\Delta \vdash M : \Box C$. Moreover $((\Delta, \Gamma, \Delta'), \downarrow M) \in [\![C]\!]$ by definition of $[\![\Box C]\!]$.
- If $A \equiv \Box C$, where C is pure, then we proceed like above. Thanks to our definition of $[\![\Box C]\!]$, there is no difficulty. □

Proposition 3.27 (Modal weakening of $[\![\Box A]\!]$)
If $\Box A$ is a type, $(\Delta, M) \in [\![\Box A]\!]$ and Δ, Δ' is valid then $((\Delta, \Delta'), M) \in [\![\Box A]\!]$.

Proof Easy thanks to the definitions of $[\![\Box C]\!]$. □

3.3.3 Interpretation of context stacks

Now we define the notion of *interpretation of context stack*, the last step before the key result of the next section. Our definition is a bit more complex than in the classic case of simply-typed lambda-calculus because we have to deal with context stacks instead of contexts (see [Lel98]).

Definition 3.28 (Pre-substitution)
A pre-substitution *ρ from a stack Δ to a stack Ψ is a mapping from the set of the variables declared in Δ into the set of the terms with all their free variables in Ψ.*

A pre-substitution *ρ can be applied to a term M with all its free variables in Δ. The result of this operation, denoted by $\rho(M)$, is equal to term M where all its free variables x have been replaced by their images by ρ, $\rho(x)$.*

Notations Given two stacks Δ and Ψ, a pre-substitution ρ from Δ to Ψ, a variable x not declared in Δ and M a term with all its free variables in Ψ, we denote by $\rho[x \mapsto M]$ the pre-substitution from $\Delta, x : A$ to Ψ such that:

- $\rho[x \mapsto M](y) = \rho(y)$ if y is declared in Δ,
- $\rho[x \mapsto M](x) = M$

Given a stack Δ' such that $\Delta; \Delta'$ is valid and a substitution ρ' from Δ' to Ψ, $\rho; \rho'$ denotes the pre-substitution from $\Delta; \Delta'$ to Ψ such that:

- $\rho; \rho'(x) = \rho(x)$ if x is declared in Δ,
- $\rho; \rho'(x) = \rho'(x)$ if x is declared in Δ'

Definition 3.29 (Interpretation of context stack)
Given two stacks Δ and Ψ, the interpretation *of Δ in Ψ, $[\![\Delta]\!]_\Psi$, is a set of pre-substitutions from Δ to Ψ. It is defined by induction on Δ, where the notation Ψ^n has been defined in Section 2.4:*

- $[\![.]\!]_\Psi$ *is the singleton whose only element is the empty pre-substitution from* . *to* Ψ.
- $[\![\Gamma, x : A]\!]_\Psi$ *is the set of the pre-substitutions* $\rho[x \mapsto M]$, *where* ρ *belongs to* $[\![\Gamma]\!]_\Psi$ *and* (Ψ, M) *is in* $[\![A]\!]$.
- $[\![\Delta;\Gamma]\!]_\Psi$ *is the set of pre-substitutions* $\rho;\rho'$ *such that* ρ *belongs to* $[\![\Delta]\!]_{\Psi^n}$ $(n \in I\!N)$ *and* ρ' *belongs to* $[\![\Gamma]\!]_\Psi$.

3.3.4 Soundness of typing

Now we reach the final stage of the proof. The Strong Normalization property will be an easy corollary of the key lemma 'Soundness of Typing'. First, we prove a technical lemma which involves modality and the useful typing restrictions it implies.

Lemma 3.30 *Let* $\rho \in [\![\Delta]\!]_\Psi$, M *a term such that* $\Delta;. \vdash M : A$, *with* A *pure, and* $\rho(M) \equiv \lambda\vec{x} : \vec{T}.(C_{j,k}\ M_1 \ldots M_{n_{j,k}})$ *(resp.* $\rho(M) \equiv \lambda\vec{x} : \vec{T}.(x_k\ M_1 \ldots M_{n_{j,k}})$*), then we have* $M \equiv \lambda\vec{x} : \vec{T}.(C_{j,k}\ N_1 \ldots N_{n_{j,k}})$ *(resp.* $M \equiv \lambda\vec{x} : \vec{T}.(x_k\ N_1 \ldots N_{n_{j,k}})$*).*

Proof
M is:

- either a variable y. Since we have assumed that $\Delta;. \vdash M : A$, y has a modal type. This is not compatible with the hypothesis 'A is pure',
- or $\lambda\vec{x} : \vec{T}.y$, where y is a variable distinct from any x_i. By the inversion lemmas, we have $\Delta;\vec{x} : \vec{T} \vdash y : A$. Like in the previous case, this means that y has a modal type. As before, this is not compatible with the hypotheses,
- or $\lambda\vec{x} : \vec{T}.(y\ N_1 \ldots N_{n_{j,k}})$, where y is a variable. Applying the inversion lemmas, we see that as before y should be well-typed in stack $(\Delta;\vec{x} : \vec{T})$, which means that y has a modal type. This is impossible,
- or $M \equiv \lambda\vec{x} : \vec{T}.(C_{j,k}\ N_1 \ldots N_{n_{j,k}})$ (resp. $M \equiv \lambda\vec{x} : \vec{T}.(x_k\ N_1 \ldots N_{n_{j,k}})$). This is the only remaining possibility. □

Lemma 3.31 (Soundness of Typing)
If $\Delta \vdash M : A$ *and* $\rho \in [\![\Delta]\!]_\Psi$, *then* $(\Psi, \rho(M)) \in [\![A]\!]$.

Proof By induction on the derivation of $\Delta \vdash M : A$.

- (Var) By definition of $[\![\Delta]\!]_\Psi$.

- (Lam) $\dfrac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x : A.M \ : \ A \to B}$

  Let $\rho \in [\![\Delta]\!]_\Psi$, Γ' a context and N a term such that $((\Psi, \Gamma'), N) \in [\![A]\!]$. We want to show that $((\Psi, \Gamma'), (\rho(\lambda x : A.M) \ \ N)) \in [\![B]\!]$.

  Now, thanks to α-conversion, we can assume without loss of generality that x is not declared in Ψ, Γ'. Thus, $\rho[x \mapsto x] \in [\![\Delta, x : A]\!]_{\Psi, x:A}$. By induction hypothesis, $\rho[x \mapsto x](M) \equiv \rho(M)$ is SN in $\Psi, x : A$.

  We prove that $((\Psi, \Gamma'), (\rho(\lambda x : A.M) \ \ N))$ belongs to $[\![B]\!]$ by induction on $\nu_{\Psi, x:A}(\rho(M)) + \nu_{\Psi, \Gamma'}(N)$ using condition (CR3). There are three ways to reduce $(\rho(\lambda x : A.M) \ \ N)$ without basic η-expansion:

  - $\Psi, \Gamma' \vdash (\rho(\lambda x : A.M) \ \ N) \hookrightarrow \rho(M)[N/x] : B$. $\rho[x \mapsto N] \in [\![\Delta, x : A]\!]_{\Psi, \Gamma'}$ and $\rho(M)[x \mapsto N] \equiv \rho(M)[N/x]$. Hence by induction hypothesis, $((\Psi, \Gamma'), \rho(M)[N/x]) \in [\![B]\!]$
  - $\Psi, \Gamma' \vdash (\rho(\lambda x : A.M) \ \ N) \hookrightarrow (\lambda x : A.(\rho(M))' \ \ N) : B$. We use the induction hypothesis.
  - $\Psi, \Gamma' \vdash (\rho(\lambda x : A.M) \ \ N) \hookrightarrow (\lambda x : A.\rho(M) \ \ N') : B$. We use the induction hypothesis.

- (App) $\dfrac{\Delta \vdash M : A \to B \qquad \Delta \vdash N : A}{\Delta \vdash (M \ \ N) : B}$

  Immediate by definition of $[\![A \to B]\!]$ since $\rho(M \ \ N) \equiv (\rho(M) \ \ \rho(N))$.

- (Pair) $\dfrac{\Delta \vdash M_1 : A \qquad \Delta \vdash M_2 : B}{\Delta \vdash \langle M_1, M_2 \rangle : A \times B}$

  First, by induction hypothesis, $(\Psi, \rho(M_1)) \in [\![A]\!]$ and $(\Psi, \rho(M_2)) \in [\![B]\!]$. Thus M_1 and M_2 are SN in Ψ. Let Γ be a context such that (Ψ, Γ) is valid. We have to show that $((\Psi, \Gamma), \text{fst } \rho(\langle M_1, M_2 \rangle)) \in [\![A]\!]$ and $((\Psi, \Gamma), \text{snd } \rho(\langle M_1, M_2 \rangle)) \in [\![B]\!]$. We show these properties by induction on $\nu_\Psi(M_1) + \nu_\Psi(M_2)$ using (CR3). For instance, fst $\rho(\langle M_1, M_2 \rangle)$ is neutral and there are three ways to reduce it:

  - $\Psi, \Gamma \vdash \text{fst } \rho(\langle M_1, M_2 \rangle) \hookrightarrow \text{fst } \rho(\langle M_1', M_2 \rangle) : A$. We apply the induction hypothesis.
  - $\Psi, \Gamma \vdash \text{fst } \rho(\langle M_1, M_2 \rangle) \hookrightarrow \text{fst } \rho(\langle M_1, M_2' \rangle) : A$. We apply the induction hypothesis.
  - $\Psi, \Gamma \vdash \text{fst } \rho(\langle M_1, M_2 \rangle) \equiv \text{fst } \langle \rho(M_1), \rho(M_2) \rangle \hookrightarrow \rho(M_1) : A$.

- (Fst) $\dfrac{\Delta \vdash M : A \times B}{\Delta \vdash \text{fst } M : A}$

  Immediate by definition of $[\![A \times B]\!]$ since $\rho(\text{fst } M) \equiv \text{fst } \rho(M)$.

- (Snd) $\dfrac{\Delta \vdash M : A \times B}{\Delta \vdash \text{snd } M : B}$

  Immediate by definition of $[\![A \times B]\!]$ since $\rho(\text{snd } M) \equiv \text{snd } \rho(M)$.

- $(\downarrow)$ $\dfrac{\Delta \vdash M : \Box A}{\Delta \vdash \downarrow M : A}$

  Immediate by definition of $[\![\Box A]\!]$ since $\rho(\downarrow M) \equiv \downarrow \rho(M)$.

- (Pop) $\dfrac{\Delta \vdash M : \Box A}{\Delta;\Gamma \vdash M : \Box A}$ $\Delta;\Gamma$ valid

  By Lemma 3.27.

- $(C_{j,k})$ $\qquad \overline{\Delta \vdash C_{j,k} : B_{j,k,1} \to \cdots \to B_{j,k,n_{j,k}} \to L_j}$

  $C_{j,k}$ belongs to $\mathcal{NT}$ and is only reducible by η-expansion. Thus it belongs to the interpretation of its type by (CR3).

- (Case) Immediate by definition of $[\![\Box A]\!]$, with A pure.

- For rule (It), the proof is also immediate by definition of $[\![\Box A]\!]$, with A pure.

- $(\uparrow)$ $\dfrac{\Delta;. \vdash M : A}{\Delta \vdash \uparrow M : \Box A}$

  This is the most difficult case. First, we prove that $((\Psi,\Psi'), \downarrow\uparrow \rho(M)) \in [\![A]\!]$. Since ρ is in $[\![\Delta]\!]_\Psi$, it also belongs to $[\![\Delta;.]\!]_{\Psi,\Psi'}$. Thus $((\Psi,\Psi'),\rho(M)) \in [\![A]\!]$ by induction hypothesis. We prove that $((\Psi,\Psi'),\downarrow\uparrow\rho(M)) \in [\![A]\!]$ by induction on $\nu_{\Psi,Psi'}(\rho(M))$ using (CR3). $\downarrow\uparrow\rho(M)$ belongs to $\mathcal{NT}$ and there are two ways to reduce this term:

  - $\Psi,\Psi' \vdash \downarrow\uparrow\rho(M) \hookrightarrow \rho(M) : A$
  - $\Psi,\Psi' \vdash \downarrow\uparrow\rho(M) \hookrightarrow \downarrow\uparrow(\rho(M))' : A$. We apply the induction hypothesis.

  Now if A is pure, there are other conditions in the definition of $[\![\Box A]\!]$. We assume that $A = T_1 \to \cdots \to T_p \to L$, with L ground type. We assume that Ψ,Ψ' is valid, $((\Psi,\Psi'),M_{j,k}) \in \left(\prod_{q=1}^{q=n_{j,k}} \left[\!\left[\Box(\prod_{i=1}^{i=p} T_i.B_{j,k,q})\right]\!\right]\right).\tau(\sigma(L_j))$. And we want to prove that $((\Psi,\Psi'),\langle\sigma\rangle\text{Case} \uparrow \rho(M) \text{ of } (M_{j,k}))$ belongs to $\prod_{z=1}^{z=p}\left(\prod_{m=1}^{m=r_z}\left[\!\left[\Box(\prod_{i=1}^{i=p} T_i.T_z^m)\right]\!\right].\tau(L_z)\right).\tau(L)$. We use property (CR3) and we examine the possible forms of $\rho(M)$:

  - If $\uparrow\rho(M)$ is not a redex for Case, there is no difficulty. $\langle\sigma\rangle\text{Case} \uparrow \rho(M) \text{ of } (M_{j,k})$ belongs to $\mathcal{NT}$, we apply (CR3) and we are done by induction on $\nu_{\Psi,\Psi'}(\uparrow\rho(M)) + \sum_{j,k}\nu_{\Psi,\Psi'}(M_{j,k})$.

    The following cases are a bit more difficult because they correspond to the elimination of the Case construct.

– If $\rho(M) \equiv \lambda\vec{x_i} : \vec{T_i}.(C_{j,k}\ M_1 \dots M_{n_{j,k}})$ (respectively $\rho(M) \equiv \lambda\vec{x_i} : \vec{T_i}.(x_k\ M_1 \dots M_{n_{j,k}})$) , then, thanks to Lemma 3.30, we know that $M \equiv \lambda\vec{x_i} : \vec{T_i}.(C_{j,k}\ N_1 \dots N_{n_{j,k}})$ (resp. $M \equiv \lambda\vec{x_i} : \vec{T_i}.(x_k\ N_1 \dots N_{n_{j,k}})$) . The substitutions have only taken place in the subterms $N_1, \dots, N_{n_{j,k}}$. The terms $\lambda\vec{x_i} : \vec{T_i}.N_l$ are smaller than M thus we can apply the induction hypothesis: the pairs $((\Delta, \Delta'), \rho(\lambda\vec{x_i} : \vec{T_i}.N_l)$ belongs to the right interpretations of types and we are done.

For It, the proof is pretty similar. □

The strong normalization theorem is then a corollary of the previous lemma.

Theorem 3.32 (Strong Normalization)
There is no infinite sequence of reductions.

Proof
Let us assume that there is an infinite sequence of reductions which begins by $\Delta \vdash M \hookrightarrow M' : A$. Then by Proposition 3.1, we have $\Delta \vdash M : A$. Using the previous lemma and the obvious fact that the pre-substitution identity from Δ to Δ belongs to $[\![\Delta]\!]_\Delta$, we see that (Δ, M) must belong to $[\![A]\!]$ and thus that M must be SN in Δ. □

3.4 Confluence

At this point, we have proved local confluence (Lemma 3.4) and strong normalization (Theorem 3.32). The proof of confluence is a corollary of these two results (this fact is often called "Newman's Lemma", after [New42]).

Theorem 3.33 (Confluence) *If $\Delta \vdash M \hookrightarrow_* N : A$ and $\Delta \vdash M \hookrightarrow_* P : A$ then there is a term Q such that $\Delta \vdash N \hookrightarrow_* Q : A$ and $\Delta \vdash P \hookrightarrow_* Q : A$.*

Proof If $M \equiv N$ or $M \equiv P$, the result is trivial. Otherwise we use Lemma 3.4 and prove the result by induction on $\nu_\Delta(M)$.

We assume that $\Delta \vdash M \hookrightarrow N_1 \hookrightarrow_* N : A$ and $\Delta \vdash M \hookrightarrow P_1 \hookrightarrow_* P : A$. Then by Lemma 3.4 (local confluence), there is a term R such that $\Delta \vdash N_1 \hookrightarrow_* R : A$ and $\Delta \vdash P_1 \hookrightarrow_* R : A$. Then by induction hypothesis, there is a term R_1 such that $\Delta \vdash N \hookrightarrow_* R_1 : A$ and $\Delta \vdash R \hookrightarrow_* R_1 : A$. Similarly, there is a term R_2 such that $\Delta \vdash P \hookrightarrow_* R_2 : A$ and $\Delta \vdash R \hookrightarrow_* R_2 : A$. Finally the induction hypothesis allows us to claim that since $\Delta \vdash R \hookrightarrow_* R_1 : A$ and $\Delta \vdash R \hookrightarrow_* R_2 : A$, there is a term Q such that $\Delta \vdash R_1 \hookrightarrow_* Q : A$ and $\Delta \vdash R_2 \hookrightarrow_* Q : A$. Thus $\Delta \vdash N \hookrightarrow_* Q : A$ and $\Delta \vdash P \hookrightarrow_* Q : A$ (see Figure 4). □

Corollary 3.34 (Uniqueness of normal forms)
If $\Delta \vdash M : A$ then M reduces in Δ to a unique canonical form.

Proof

We already know that no sequence of reduction starting from M is infinite. Now, if $\Delta \vdash M \hookrightarrow_* N_1 : A$ and $\Delta \vdash M \hookrightarrow_* N_2 : A$, with N_1 and N_2 canonical forms, then by Lemma 3.33, there is a term P such that $\Delta \vdash N_1 \hookrightarrow_* P : A$ and $\Delta \vdash N_2 \hookrightarrow_* P : A$. Since N_1 and N_2 are canonical forms, it means that $N_1 \equiv P \equiv N_2$. □

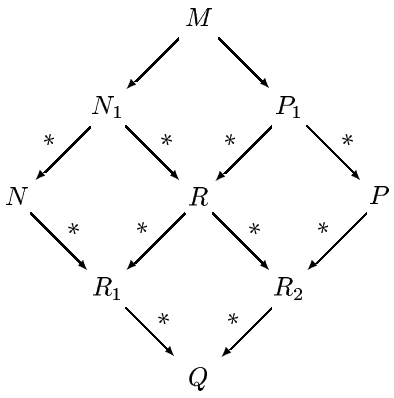


Figure 4: Proof of confluence

3.5 Conservative extension

Finally, we show that our calculus is a conservative extension of the simply-typed λ-calculus. This theorem will be a corollary of the following technical lemma.

Notations Given a stack $\Delta \equiv \Gamma_1; \ldots; \Gamma_n$, two variables x and y declared respectively in Γ_i and Γ_j, we say that y is declared in the *future* of x if $j \geq i$.

A term is said to be *pure* if it is a term of the simply-typed λ-calculus (that is to say it contains no Case/It, no modal operators, no pairs, no projections).

Lemma 3.35 (Possible forms of a canonical term)
Given a stack Δ and its local context Γ, if $\Delta \vdash M : B$, M canonical form then:

- *a) either M contains an impure variable of Δ and all the free variables of M are in the future of an impure variable,*
- *b) or M is an impure abstraction (i.e. the type of the bound variable is impure) and all the free variables of M belong to Γ,*
- *c) or M is of the form $\uparrow N$ and N is closed,*

- *d) or M is of the form $\langle M, N\rangle$ and all the free variables of M and N belong to Γ,*
- *e) or M is pure and B is pure. Moreover all the free variables of M belong to Γ.*

Proof By induction on judgment $\Delta \vdash M : B$:

- (Var) We have e) if the type of the variable is pure, a) otherwise.
- (App) $\dfrac{\Delta \vdash P : A \to B \qquad \Delta \vdash Q : A}{\Delta \vdash (P\ Q) : B}$

  We examine the possibilities for P:
  - If P verifies a) then
    - if Q verifies a) then $(P\ Q)$ verifies a),
    - otherwise Q is closed or all its free variables belong to the local context of Δ and so $(P\ Q)$ verifies a).
  - P cannot verify b) (because $(P\ Q)$ is normal) nor c) nor d) (it has not the right type).
  - If e) for P then we examine the different cases for Q.
    - If Q verifies a) then $(P\ Q)$ verifies a).
    - Q cannot verify b) nor c) nor d) because its type is pure.
    - If e) for Q then $(P\ Q)$ is pure and all the free variables of $(P\ Q)$ belong to Γ.
- (Lam) $\dfrac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x : A.M : A \to B}$
  - If a) for M then
    - if x is the only impure free variable in M then all the free variables of M belong to Γ and $\lambda x : A.M$ is an impure abstraction, hence b),
    - otherwise a) for $\lambda x : A.M$.
  - If b), c) or d) for M then b) for $\lambda x : A.M$.
  - If e) then b) for $\lambda x : A.M$ if A is not pure, e) otherwise.
- (Fst) $\dfrac{\Delta \vdash M : A \times B}{\Delta \vdash \text{fst } M : A}$
  - If a) for M then a) for fst M.
  - M cannot verify b) nor c) (not the right type) nor d) (because fst M is normal).
  - e) is impossible for M since it has type $A \times B$.
- (Snd) Very similar to the previous case.

- (Pair) $\dfrac{\Delta \vdash M_1 : A \qquad \Delta \vdash M_2 : B}{\Delta \vdash \langle M_1, M_2 \rangle : A \times B}$

  We examine the possibilities for M_1 and M_2

  - If M_1 verifies a) then
    - If M_2 verifies a) then $\langle M_1, M_2 \rangle$ verifies a),
    - otherwise M_2 is closed or all its free variables belong to the local context of Δ and so $\langle M_1, M_2 \rangle$ verifies a).
  - Symmetrically, if M_2 verifies a) then we can conclude that $\langle M_1, M_2 \rangle$ verifies a).
  - Otherwise, M_1 and M_2 have all their free variables, if any, in the local context of Δ. Hence $\langle M_1, M_2 \rangle$ verifies d).

- $(\downarrow)$ $\dfrac{\Delta \vdash M : \Box A}{\Delta \vdash \downarrow M : A}$

  - If a) for M then a) for $\downarrow M$.
  - M cannot verify b) nor d) (not the right type) nor c) (because $\downarrow M$ is normal).
  - e) is impossible for M since it has type $\Box A$.

- $(\uparrow)$ $\dfrac{\Delta; . \vdash M : A}{\Delta \vdash \uparrow M : \Box A}$

  If a) for M then a) for $\uparrow M$. Otherwise by induction hypothesis, M is closed, thus c).

- (Pop) $\dfrac{\Delta \vdash M : \Box A}{\Delta; \Gamma \vdash M : \Box A}$

  - If a) for the premiss then a) for the conclusion of the rule.
  - b) and d) are impossible for the premiss (not the right type).
  - If c) for the premiss then c) for the conclusion.
  - e) is obviously impossible for the premiss.

- (Case) $$\frac{\Delta \vdash M : \Box \left(\prod_{i=1}^{i=p} T_i . L_n \right) \qquad \Delta \vdash M_{j,k} : \left(\prod_{q=1}^{q=n_{j,k}} \Box \left(\prod_{i=1}^{i=p} T_i \to B_{j,k,q} \right) \right) . \sigma(L_j)}{\Delta \vdash \langle \sigma \rangle \mathrm{Case}^T \ M \text{ of } \ (M_{j,k}) : \prod_{z=1}^{z=p} {T'}_z . \sigma(L_n)}$$

  - If a) for M then a) for the whole term (for the terms $M_{j,k}$, whether a), or all their free variables belong to the local context).
  - b), d) and e) are obviously impossible for M.

– If c) for M then $M =\uparrow N$ with N a closed term and $\Delta \vdash\uparrow N : \Box\left(\prod_{i=1}^{i=p} T_i.L_n\right)$. By Inversion Lemma, $\Delta;. \vdash N : \prod_{i=1}^{i=p} T_i.L_n$. By Induction Hypothesis, N is pure (a) is impossible because N is closed). Since it is a canonical form of the simply-typed λ-calculus, we know that $N \equiv \lambda\vec{x_i} : \vec{T_i}.(C \dots)$ where C is a variable or a constant $C_{j,k}$. Thus we have here a redex, which is impossible since we are dealing with normal terms.

- The case for rule (It) is similar to the previous one. □

Theorem 3.36 (Conservative extension)
Our system is a conservative extension *of the simply-typed λ-calculus, i.e. if $\Delta \vdash M : A$ with Δ pure context stack and A pure type then M has a unique canonical form N which is pure. Moreover all the free variables of N belong to the local context of Δ.*

Proof
We already know that any well-typed term reduces to a unique canonical form. To prove the theorem, we just have to prove that in a pure stack, a canonical form of pure type is pure and that all its free variables are declared in the local context. This is a trivial corollary of the previous lemma. □

4 Related works

Our system is inspired by [DPS97]. The main difference is that the underlying modal λ-calculus is easier to use, and seems to be better adapted to a future extension to dependent types. Moreover, we have written reduction rules instead of evaluation inference rules. Finally, our metatheoretic proofs are much more compact and easier to read.

Raymond McDowell and Dale Miller have proposed [MM97] a meta-logic to reason about object logics coded using higher order abstract syntax. Their approach is quite different from ours, less ambitious in a sense. They do not give a typing system, supporting the judgments-as-types principle, but two logics: one for each level (object and meta). Moreover they only have induction on natural numbers, which can be used to derive other induction principles via the construction of an appropriate measure.

5 Conclusion and future work

We have presented a modal λ-calculus IS4 with primitive recursive constructs that we claim to be better than the previous proposition [DPS97]. The conservative extension theorem, which guarantees that the adequacy of encodings is preserved, is proved as well as the Church-Rosser and strong normalization properties.

Our main goal is now to extend this system to dependent types and to polymorphic types. This kind of extension is not straightforward but we expect our system to be flexible enough to allow it.

Another interesting direction of research consists in replacing our recursive operators by operators for pattern-matching such as those used in the ALF [MN94] system, implementing Martin Lof's Type Theory [NPS90]. Some hints for a concrete syntax for that extension have been given in [DPS97]. F. Pfenning and C. Schürmann are currently working on the definition of a meta logic along these lines.

Acknowledgment I would like to thank Joëlle Despeyroux for her useful comments and suggestions on earlier versions of this work.

References

[Aka93] Y. Akama. On Mints'Reduction for ccc-calculus. In *Proceedings TLCA*, pages 1–12. Springer-Verlag LNCS 664, 1993.

[BdP96] Gavin Bierman and Valeria de Paiva. Intuitionnistic necessity revisited. In *Technical Report CSRP-96-10*, School of Computer Science, University of Birmingham, 1996.

[CK93] R. Di Cosmo and D. Kesner. A Confluent Reduction for the Extensional Typed λ-calculus. In *Proceedings ICALP'93*. Springer-Verlag LNCS 700, 1993.

[DH94] J. Despeyroux and A. Hirschowitz. Higher-order syntax and induction in coq. In F. Pfenning, editor, *Proceedings of the fifth Int. Conf. on Logic Programming and Automated Reasoning (LPAR 94), Kiev, Ukraine, July 16–21, 1994*, volume 822. Springer-Verlag LNAI, 1994.

[DP96] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. In Jr. Guy Steele, editor, *Proceedings of the 23rd Annual Symposium on Principles of Programming Languages*, pages 258–270, St. Petersburg Beach, Florida, January 1996. ACM Press.

[DPS96] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. Technical Report CMU-CS-96-172, Carnegie Mellon University, September 1996.

[DPS97] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive Recursion for Higher-Order Abstract Syntax. In J.R. Hindley and P. de Groote, editors, *Int. Conf. on Typed lambda calculi and applications - TLCA '97*, pages 147–163, Nancy, France, April 1997. Springer-Verlag LNCS 1210.

[Gha96] Neil Ghani. Eta Expansions in System F. Technical Report LIENS-96-10, LIENS-DMI, Ecole Normale Superieure, 1996.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[Hof96] Martin Hofmann. Private communication, May 1996.

[JG95] C.B. Jay and N. Ghani. The Virtues of Eta-Expansion. *Journal of Functional Programming*, 5(2):135–154, April 1995.

[Lel98] Pierre Leleu. Metatheoretic Results for a Modal λ-calculus. *Draft. To be submitted for publication.*, 1998.

[Luo94] Zhaohui Luo. *Computation and Reasoning.* Oxford University Press, 1994.

[Min79] G.E. Mints. Teorija categorii i teoria dokazatelstv.l. *Aktualnye Problemy logiki i metodologii nauky*, pages 252–278, 1979.

[MM97] Raymond McDowell and Dale Miller. A logic for reasoning with higher-order abstract syntax. Computer and Information Science Department, University of Pennsylvania, 1997. LICS'97.

[MN94] Lena Magnusson and Bengt Nordström. The ALF proof editor and its proof engine. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs*, pages 213–237. Springer-Verlag LNCS 806, 1994.

[New42] M.H.A. Newman. On theories with a combinatorial definition of 'equivalence'. *Ann. Math.*, 43(2):223–243, 1942.

[NPS90] Bengt Nordström, Kent Petersson, and Jan Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford University Press, 1990.

[PE88] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN '88 Symposium on Language Design and Implementation*, pages 199–208, Atlanta, Georgia, June 1988.

[PM92] Ch. Paulin-Mohring. Inductive definitions in the system coq. rules and properties. In J.F. Groote M. Bezem, editor, *Proceedings of the Int. Conf. on Typed Lambda Calculi and Applications, TLCA'93*, Springer-Verlag LNCS 664, 1992.

[PW95] Frank Pfenning and Hao-Chi Wong. On a modal λ-calculus for S4. In S. Brookes and M. Main, editors, *Proceedings of the Eleventh Conference on Mathematical Foundations of Programming Sematics*, New Orleans, Louisiana, March 1995. To appear in *Electronic Notes in Theoretical Computer Science*, Volume 1, Elsevier.

[Wer94] Benjamin Werner. *Une Théorie des Constructions Inductives.* PhD thesis, Université Paris 7, 1994.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399