A Calculus of Substitutions for Incomplete-Proof Representation in Type Theory

César Muñoz

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE

A Calculus of Substitutions for Incomplete-Proof Representation in Type Theory

César A. Muñoz H.

N ° 3309

Novembre 1997

THÈME 2

ISSN 0249-6399

A Calculus of Substitutions for Incomplete-Proof Representation in Type Theory

César A. Muñoz H. *

Thème 2 — Génie logiciel
et calcul symbolique

Projet Coq

Rapport de recherche n ° 3309 — Novembre 1997 — 155 pages

Abstract: In the framework of intuitionnistic logic and type theory, the concepts of "propositions" and "types" are identified. This principle is known as the Curry-Howard isomorphism, and it is at the base of mathematical formalisms where proofs are represented as typed lambda-terms.

In order to see the process of proof construction as an incremental process of term construction, it is necessary to extend the lambda-calculus with new operators. First, we consider typed meta-variables to represent the parts of a proof that are under construction, and second, we make explicit the substitution mechanism in order to deal with capture of variables that are bound in terms containing meta-variables.

Unfortunately, the theory of explicit substitution calculi with typed meta-variables is more complex than that of lambda-calculus. And worse, in general they do not share the same properties, notably with respect to confluence and strong normalization. A contribution of this thesis is to show that the properties of confluence and strong normalization are not incompatible with explicit substitution calculi.

This thesis also proposes a calculus with explicit substitutions and typed meta-variables for dependent type systems, in particular for the Calculus of Constructions, which allows incomplete proof-terms to be represented. For these type systems, we prove the main typing properties: Type Uniqueness, Subject Reduction, Weak Normalization, Confluence and Typing Decidability.

Finally, we give an application of this formalism to proof synthesis. The proposed method merges a procedure for term enumeration with a technique of higher-order unification via explicit substitutions where unification variables are coded as meta-variables.

Keywords: Explicit Substitutions, Incomplete-Proof Representation, Proof Synthesis, Curry-Howard Isomorphism, Typed Lambda-calculus, Dependent Types, Calculus of Constructions

Note: This work is a translation into English of the author's PhD thesis "Un calcul de substitutions pour la représentation de preuves partielles en théorie de types", defended at the University of Paris 7 on November 4th 1997. Supervisors: Gérard Huet, Gilles Dowek. Jury: Pierre-Louis Curien, Herman Geuvers, Guy Cousineau, René David, Thérèse Hardin, Klaus Madlener.

(Résumé : tsvp)

*Cesar.Munoz@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : (33 1) 39 63 55 11 – Télécopie : (33 1) 39 63 53 30

Un calcul de substitutions pour la représentation de preuves partielles en théorie de types

Résumé : Dans le cadre de la logique intuitionniste et la théorie des types, les notions de "propositions" et de "types" se correspondent. Cette correspondance, aussi appelé isomorphisme de Curry-Howard, est à la base de formalismes mathématiques où les preuves sont représentées comme des termes du lambda-calcul typé.

Afin de voir la construction de preuves comme un processus incrémental de construction de termes, il est nécessaire d'étendre le lambda-calcul avec de nouvelles constructions. On considère, d'une part, des métavariables typées pour représenter les parties d'une preuve restant à construire, et, d'autre part, on rend explicite l'opération de substitution afin de gérer les problèmes de capture de variables liées dans un terme contenant des métavariables.

Malheureusement, la théorie des calculs avec substitutions explicites et métavariables typées est plus complexe que celle du lambda-calcul. Et, plus grave encore, en général elles ne jouissent pas des mêmes propriétés, notamment la confluence et la normalisation. Un des apports de cette thèse est la démonstration que la confluence et la normalisation forte ne sont pas des propriétés incompatibles dans un calcul avec substitutions explicites.

Cette thèse propose aussi un calcul avec substitutions explicites et métavariables typées pour les systèmes avec types dépendants, et notamment pour le calcul des constructions, permettant de répresenter les preuves partielles. On démontre que ces systèmes vérifient les propriétés essentielles des calculs typés : l'unicité du type, la préservation du type par réduction, la confluence, la normalisation faible et la décidabilité du typage.

On donne enfin une application de ce formalisme à la synthèse de preuves. La méthode présentée combine une procédure d'énumération des termes, avec une technique d'unification d'ordre supérieur à l'aide de substitutions explicites où les variables d'unification sont codées comme des métavariables.

Mots-Clés : Substitutions explicites, Représentation de preuves partielles, Synthèse de preuves, Isomorphisme de Curry-Howard, Lambda-calcul typé, Types dépendants, Calcul des constructions

Note : Ce travail est une traduction à l'anglais de la thése de doctorat de l'author intitulé "Un calcul de substitutions pour la représentation de preuves partielles en théorie de types" soutenue à l'université de Paris 7 le 4 Novembre 1997. Directeurs de thèse : Gérard Huet, Gilles Dowek. Jury : Pierre-Louis Curien, Herman Geuvers, Guy Cousineau, René David, Thérèse Hardin, Klaus Madlener.

A mi familia,
Lucila, Gustavo y Norman.
Por ese amor incondiconal
que soporta mis pasos...

"... Cuando su padre le comunicó su alarma por haber olvidado hasta los hechos más impresionantes de su niñez, Aureliano le explicó su método, y José Arcadio Buendía lo puso en práctica en toda la casa y más tarde lo impuso a todo el pueblo. Con un hisopo entintado marcó cada cosa con su nombre: *mesa, silla, reloj, puerta, pared, cama, cacerola.* Fue al corral y marcó a los animales y las plantas: *vaca, chivo, puerco, gallina, yuca, malanga, guineo.* Poco a poco, estudiando las infinitas posibilidades del olvido, se dio cuenta de que podía llegar un día en que se reconocieran las cosas por sus inscripciones, pero no se recordara su utilidad. Entonces fue más explícito. El letrero que colgó en la cerviz de la vaca era una muestra ejemplar de la forma en que los habitantes de Macondo estaban dispuestos a luchar contra el olvido: *Esta es la vaca, hay que ordeñarla todas las mañanas para que produzca leche y a la leche hay que hervirla para mezclarla con el café y hacer café con leche.* Así continuaron viviendo en una realidad escurridiza, momentáneamente capturada por las palabras, pero que había de fugarse sin remedio cuando olvidaran los valores de la letra escrita ...".

Gabriel García Márquez. *Cien años de soledad*

Contents

1 Introduction **9**
- 1.1 From Proofs to Constructive Proofs . . . 9
- 1.2 Proofs as Programs . . . 11
- 1.3 Higher-Order Logics and Dependent Types . . . 12
- 1.4 Construction of Proof-Terms in the Simply Typed λ-Calculus . . . 16
  - 1.4.1 Incomplete Proofs and Other Demons . . . 17
  - 1.4.2 Explicit Substitutions . . . 18
  - 1.4.3 Place-Holders as Meta-Variables . . . 18
  - 1.4.4 Refining Place-Holders . . . 19
- 1.5 Towards the Construction of Proof-Terms in Dependent Types . . . 20

I A Calculus of Substitutions and Its Applications **23**

2 Type Free Calculi of Explicit Substitutions **25**
- 2.1 The Family of $\lambda\sigma$-Calculi . . . 26
  - 2.1.1 The λ-Calculus *à la* de Bruijn . . . 26
  - 2.1.2 The $\lambda\sigma$-Calculus . . . 27
  - 2.1.3 A confluent variant of $\lambda\sigma$: the $\lambda\sigma_{\Uparrow}$-Calculus . . . 28
  - 2.1.4 Preservation of Strong Normalization: the $\lambda\upsilon$-Calculus . . . 29
  - 2.1.5 Useful Properties of Abstract Relations and Rewrite Systems . . . 31
- 2.2 Solving a Conjecture about Confluence and PSN: the λ_ζ-Calculus . . . 34
  - 2.2.1 A Harmful Critical Pair . . . 34
  - 2.2.2 The λ_ζ-Calculus . . . 35
  - 2.2.3 Properties . . . 37
  - 2.2.4 An Open Problem . . . 42
- 2.3 A Left-Linear Variant of $\lambda\sigma$: the $\lambda_\mathcal{L}$-Calculus . . . 42
  - 2.3.1 The $\lambda_\mathcal{L}$-Calculus . . . 43
  - 2.3.2 Properties . . . 44
- 2.4 Related Works and Summary . . . 48

3 The System $\mathrm{CC}_\mathcal{L}$ **51**
- 3.1 Simple Types . . . 51
  - 3.1.1 The Simply-Typed Version of $\lambda_\mathcal{L}$ . . . 52
  - 3.1.2 Properties . . . 55

3.2 Dependent Types and the Calculus of Constructions . . . 56
3.2.1 Typing of Meta-variables . . . 59
3.2.2 Typing of Substitutions . . . 60
3.2.3 A Calculus of Pre-Expressions . . . 62
3.3 The Systems $\lambda\Pi_{\mathcal{L}}$ and $CC_{\mathcal{L}}$ . . . 64
3.3.1 The Case of (SCons) in $\lambda\sigma$ . . . 69
3.4 Related Works and Summary . . . 70

4 An Application to Proof Synthesis **71**
4.1 Proof Synthesis . . . 71
4.1.1 An Example with Simple Types . . . 71
4.1.2 An Example with Dependent Types . . . 72
4.1.3 An Example with Polymorphism . . . 72
4.2 A Type System with Constraints . . . 73
4.3 Graftings, Derivations and Solutions . . . 76
4.4 Solving a Signature . . . 80
4.5 Examples . . . 85
4.5.1 In Dependent Types . . . 85
4.5.2 An Example with Polymorphism . . . 86
4.6 Efficiency Improvements . . . 88
4.6.1 Using the Rigid-Rigid Constraints . . . 88
4.6.2 Using the Flexible-Rigid Constraints . . . 89
4.6.3 Solving Signatures in $\lambda\Pi_{\mathcal{L}}$ . . . 90
4.7 Related Works and Summary . . . 91

II A Calculus of Substitutions and Its Properties **93**

5 Geuvers' Lemma **97**
5.1 The $\lambda_{\mathcal{L}\Pi}$-Calculus in Curry Style . . . 97
5.2 Erasing Typing Annotations and Decorating Expressions . . . 99
5.3 Geuvers' Lemma . . . 102

6 Elementary Properties **103**
6.1 Basic Lemmas . . . 104
6.2 Sort Soundness . . . 106
6.3 Type Uniqueness . . . 110
6.4 Subject Reduction . . . 112
6.5 Typing Soundness . . . 117

7 Weak Normalization **119**
7.1 Simple Types . . . 120
7.1.1 Saturated Sets and Interpretation of Types . . . 120
7.1.2 Weak Normalization Proof for $\lambda_{\mathcal{L}}$ . . . 122
7.2 Systems $CC_{\mathcal{L}}$ and $\lambda\Pi_{\mathcal{L}}$ . . . 125
7.2.1 Interpretation of Terms in $\lambda\Pi_{\mathcal{L}}$ . . . 128

7.2.2 Weak Normalization Proof for $\lambda\Pi_{\mathcal{L}}$ 128
7.2.3 Interpretation of Terms in $\mathrm{CC}_{\mathcal{L}}$ 133
7.2.4 Weak Normalization Proof for $\mathrm{CC}_{\mathcal{L}}$ 135

8 Church-Rosser and Confluence **139**

9 Conclusion **143**
9.1 Related Works 143
9.2 Summary and Perspectives 144

Chapter 1

Introduction

1.1 From Proofs to Constructive Proofs

There is an old joke which says how to distinguish computer scientists from logicians. You ask: "Do you know what time is it?". All of them look their watches, but computer scientists say: "4:12.35, 4:12.36, 4:12.37, ...", while logicians answer: "Yes". Actually, both answers are right; but the first one has a constructive aspect: it exhibits some witness. This joke reflects, in a humorous way, the reasoning mechanism of two logical frameworks: *Intuitionistic Logic* and *Classical Logic*.

Intuitionistic reasoning rejects the classical law of the excluded middle: "$A \vee \neg A$", and thus a proof of $\exists x.P(x)$ always exhibits an object a such that $P(a)$ holds. The *Heyting-Kolmogorov semantic* of intuitionistic logic considers proofs as mathematical objects. For example: a proof of a proposition $A \to B$ is a function that takes a proof of A as an argument and returns a proof of B, a proof of $A \wedge B$ is a pair formed by a proof of A and a proof of B, a proof of $\exists x.P(x)$ is a pair formed by an object a and a proof of $P(a)$, and so on for all the logical connectives. Intuitionistic proofs, in opposition to classical ones, are always constructive.

Curry and Howard remarked the correspondence between the Heyting-Kolmogorov semantic of intuitionistic proofs and terms of typed λ-calculus. This correspondence is known as the *Curry-Howard isomorphism*, or the *propositions-as-types principle*.

In order to understand the Curry-Howard isomorphism, we consider an intuitionistic minimal logic where propositional formulas are built from atomic propositions and the implication connective. We use uppercase Latin letters $A, B, \ldots$ to denote formulas and the Greek letter Γ to denote a set of formulas. As usual, we use parentheses to indicate precedences. In this logic, a *provable judgment* is a pair formed by a set of formulas Γ and a formula A, noted $\Gamma \vdash_{\mathcal{I}} A$, which can be derived by finite applications of the following rules:

$$\frac{}{\Gamma, A \vdash_{\mathcal{I}} A}\,(\text{Axiom}) \qquad \frac{\Gamma, A \vdash_{\mathcal{I}} B}{\Gamma \vdash_{\mathcal{I}} A \to B}\,(\text{Intro}^{\to}) \qquad \frac{\Gamma \vdash_{\mathcal{I}} A \to B \quad \Gamma \vdash_{\mathcal{I}} A}{\Gamma \vdash_{\mathcal{I}} B}\,(\text{Elim}^{\to})$$

The rules above are formulated following the sequent presentation of Natural Deduction due to Gentzen [31], but they correspond to well-known rules of traditional logic. The rule (Intro$^{\to}$) is just the Deduction Theorem, and the rule (Elim$^{\to}$) is best known as Modus-Ponens. Informally, the judgment $\Gamma \vdash_{\mathcal{I}} A$ means that the proposition A is a logical consequence of the hypotheses in Γ.

A formula A is a *tautology* if and only if $\vdash_{\mathcal{I}} A$. For instance, the formula $A \to ((A \to B) \to B)$ is a tautology since it can be derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A, A{\to}B \vdash_{\mathcal{I}} A{\to}B}\,(\text{Axiom}) \qquad \dfrac{}{A, A{\to}B \vdash_{\mathcal{I}} A}\,(\text{Axiom})}{A, A{\to}B \vdash_{\mathcal{I}} B}\,(\text{Elim}^{\to})}{A \vdash_{\mathcal{I}} (A{\to}B){\to}B}\,(\text{Intro}^{\to})}{\vdash_{\mathcal{I}} A{\to}((A{\to}B){\to}B)}\,(\text{Intro}^{\to})$$

On the other hand we have the Type Theory. It was originally introduced by Russell and Whitehead at the beginning of this century, and beside refined by Church in the 1940's, as a foundational system to develop mathematics. Just as modern set theories, it solves the Russell's paradox of Cantor's Set Theory.

In the simply-typed λ-calcul introduced by Church, the mathematical objects are the terms of λ-calculus, and the *type* of an object represents its functional degree. Objects can be structured according to their types.

Let $\mathcal{V}$ be an enumerable set of variables $x, y, \ldots$, the set of *(λ-)terms* is inductively defined by: (1) the variables are terms, (2) if M and N are terms, then $(M\ N)$ is a term, and (3) if x is a variable, M is a term and A is a type, then $\lambda x{:}A.M$ is a term. A term of the form $\lambda x{:}A.M$ is called *abstraction* and a term of the form $(M\ N)$ is called *application*. We remark that abstractions are binding structures. That means that "free" occurrences of x in M will be bound in $\lambda x{:}A.M$. As usual in theories with binding operators, terms of λ-calculus are equivalent modulo renaming of bound variables, thus $\lambda x{:}A.x$ and $\lambda y{:}A.y$ represent both the same term in λ-calculus.

Types are formed from a set of basic types and its functional closure, i.e. A is a type if and only if (1) it is a basic type or (2) it has the form $B{\to}C$ where B and C are types. We use the uppercase Latin letters $A, B, \ldots$ to range over types.

A *variable declaration* is a pair $x{:}A$ where x is a variable and A is type. A *context* is a set of variable declarations. Context assigns a unique type to each variable. We use Greek letters $\Gamma, \Delta, \ldots$ to denote contexts.

A *valid typing judgment* is a triple formed by a context Γ, a term M and a type A, denoted $\Gamma \vdash M : A$, which can be derived by finite applications of the following rules:

$$\dfrac{}{\Gamma,\ x{:}A \vdash x : A}\,(\text{Var}) \qquad \dfrac{\Gamma,\ x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : A{\to}B}\,(\text{Abs}) \qquad \dfrac{\Gamma \vdash M : A{\to}B \quad \Gamma \vdash N : A}{\Gamma \vdash (M\ N) : B}\,(\text{Appl})$$

The valid typing judgment $\Gamma \vdash M : A$ can be read as "M is a term of type A in Γ".

We say that a term M is *well-typed* if and only if there exists a type A such that $\vdash M : A$, and we say that a type A is *inhabited* if and only if there exists a term M such that $\vdash M : A$. We remark that in the two cases the context is empty, i.e. the term M does not contain free variables.

If we identify propositions (of the intuitionistic minimal logic) with types (of the simply-typed λ-calculus), then we realize that the derivation rules (Axiom), (Intro$^{\to}$) and (Elim$^{\to}$), correspond one to one to the typing rules (Var), (Abs) and (Appl). Indeed, typing rules are logical rules annotated with λ-terms. This is essentially the *Curry-Howard isomorphism*. In this framework, as expected, the set of tautologies is exactly the set of inhabited types, i.e. $\vdash_{\mathcal{I}} A$ if and only if there exists a term M such that $\vdash M : A$. Hence, the Curry-Howard isomorphism is also called the *propositions-as-types principle*.

For instance, if we decorate the above derivation tree of tautology $A{\to}((A{\to}B){\to}B$, we get the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x{:}A,\ y{:}A{\to}B \vdash y : A{\to}B}\,(\text{Var}) \qquad \dfrac{}{x{:}A,\ y{:}A{\to}B \vdash x : A}\,(\text{Var})}{x{:}A,\ y{:}A{\to}B \vdash (y\ x) : B}\,(\text{Appl})}{x{:}A \vdash \lambda y{:}A{\to}B.(y\ x) : (A{\to}B){\to}B}\,(\text{Abs})}{\vdash \lambda x{:}A.\lambda y{:}A{\to}B.(y\ x) : A{\to}((A{\to}B){\to}B)}\,(\text{Abs})$$

Seen as a type, the proposition

$$A{\to}((A{\to}B){\to}B)$$

is inhabited by the term

$$\lambda x{:}A.\lambda y{:}A{\to}B.(y\ x)$$

The isomorphism also works in another sense. If we take the well-typed term $\lambda x{:}A.\lambda y{:}A{\to}B.(y\ x)$, then it is possible to re-build the above derivation tree of $A{\to}((A{\to}B){\to}B)$. Thus, well-typed terms are one-dimensional compact representations of derivation trees. In fact, the Curry-Howard isomorphism identifies not just propositions with types, but also proofs with terms.

1.2 Proofs as Programs

Typed λ-*calculus* was not only developed to represent mathematical objects, but also to compute with them. From this point of view, the abstraction $\lambda x{:}A.M$ denotes the function with parameter x of type A and body M, and $(M\ N)$ denotes the application of a function M to an argument N. The computational aspect of λ-calculus is expressed by the β-*conversion*:

$$(\lambda x{:}A.M\ N) =_\beta M\{x := N\}$$

where $M\{x := N\}$ denotes the term in which all the free occurrences of the variable x in the term M have been replaced by N (with renaming of bound variables when necessary). The form $\{x := N\}$ is called a *substitution*.

When the β-conversion is oriented from left to right and used as a rewrite rule, we have a symbolic way to compute the application of a function $\lambda x{:}A.M$ to an argument N. For instance, the term $(\lambda x{:}A.(x\ x)\ y)$ β-reduces to $(x\ x)\{x := y\}$ which is equal to $(y\ y)$. The β-reduction is at the basis of functional programming languages.

If we consider the relation over terms induced by the β-reduction, i.e. for any terms M and N, $M \xrightarrow{\beta} N$ if and only if M is converted to N using one-step of β-reduction, and its transitive-reflexive closure, as usual denoted by $\xrightarrow{\beta^*}$, then simply-typed λ-calculus holds the following properties [4, 36, 57]:

- *confluence*, i.e. for any terms M, M' and M'' such that $M \xrightarrow{\beta^*} M'$ and $M \xrightarrow{\beta^*} M''$, there exists a term N such that $M' \xrightarrow{\beta^*} N$ and $M'' \xrightarrow{\beta^*} N$.
- *termination* i.e. if M is a well-typed term, then any reduction of M is finite.
- *subject reduction*, i.e. types are preserved by β-reductions: if $\Gamma \vdash M : A$ and $M \xrightarrow{\beta} N$, then $\Gamma \vdash N : A$.

The termination property does not hold for ill-typed terms as shown by the example:

$$(\lambda x{:}A.(x\ x)\ \lambda x{:}A.(x\ x)) \xrightarrow{\beta} (\lambda x{:}A.(x\ x)\ \lambda x{:}A.(x\ x)) \xrightarrow{\beta} \ldots$$

Now, we will explain β-reductions from a logical point of view.

Consider the following deduction:

$$\dfrac{\dfrac{\overline{\Gamma, A \vdash B}}{\Gamma \vdash A{\rightarrow}B}\,(\text{Intro}^{\rightarrow}) \qquad \overline{\Gamma \vdash A}}{\Gamma \vdash B}\,(\text{Elim}^{\rightarrow})$$

In this proof, an implication symbol is introduced and afterwards eliminated. This scheme of derivations, (Intro$^{\rightarrow}$) followed by (Elim$^{\rightarrow}$) in the same symbol, is known as a *cut*. The above derivation shows that the following rule is admissible:

$$\dfrac{\Gamma, A \vdash B \quad \Gamma \vdash A}{\Gamma \vdash B}\,(\text{Cut})$$

In a proof development, we use a cut to weak a goal by assuming an additional hypothesis. A practical use of cuts in formal proofs is the application of lemmas.

Gentzen proved in 1934 that valid judgments in intuitionistic logic admit derivation proofs without cuts. This general property is traditionally called *cut-elimination* theorem or the *Hauptsatz*.

Via the Curry-Howard isomorphism, we can decorate the above deduction as follows:

$$\dfrac{\dfrac{\overline{\Gamma,\ x{:}A \vdash M : B}}{\Gamma \vdash \lambda x{:}A.M : A{\rightarrow}B}\,(\text{Abs}) \qquad \overline{\Gamma \vdash N : A}}{\Gamma \vdash (\lambda x{:}A.M\ N) : B}\,(\text{Appl})$$

The above derivation can be simplified as

$$\dfrac{\overline{\Gamma,\ x{:}A \vdash M : B} \qquad \overline{\Gamma \vdash N : A}}{\Gamma \vdash M\{x := N\} : B}\,(\text{Cut})$$

In fact, strong-normalization theorem of typed λ-calculus is a particular version of the cut-elimination theorem.

1.3 Higher-Order Logics and Dependent Types

The Curry-Howard isomorphism can be extended to intuitionistic higher-order logics by taking more complex type theories.

Consider for example the universal quantification in arithmetic. According to the Curry-Howard isomorphism, a proposition like $\forall x{:}nat.x \geq 0$ can be considered as a type. We say that it is a *dependent type* because it depends on terms (the bound variable x and the constant 0). In type theory, the type $\forall x{:}A.B$ is usually written $\Pi x{:}A.B$ and it is called a *product* type.

The typing rules that correspond to the intuitionistic logical rules of quantifier $\forall$ are:

$$\frac{\Gamma,\ x{:}B \vdash M : A}{\Gamma \vdash \lambda x{:}B.M : \Pi x{:}B.A}\ (\mathrm{Abs}_\Pi) \qquad \frac{\Gamma \vdash M : \Pi x{:}B.A \quad \Gamma \vdash N : B}{\Gamma \vdash (M\ N) : A\{x := N\}}\ (\mathrm{Appl}_\Pi)$$

From a logical point of view, the rule (Abs_Π) says that if we have a proof of A, where the variable x appears free in A, then we can generalize x in order to get a proof of $\forall x.A$. In a similar way, the rule (Appl_Π) says that if we have a proof of $\forall x.A$, then we can instantiate x in A with an arbitrary term N (provided that x and N have the same type).

We remark that (Abs_Π) and (Appl_Π) are generalizations of (Abs) and (Appl) where the type $A{\rightarrow}B$ is just a notation for $\Pi x{:}A.B$ when x does not occur free in B.

Dependent types can be considered independently from the Curry-Howard isomorphism. For example, in a programming language as PASCAL, the type `array[1..10] of bool` depends on constant `1`, `10` and `bool`. We can also imagine type declarations as[1]:

```
Var n:integer;
TYPE A=array[1..n] of bool;
```

or

```
Var n: integer;
Var t: type;
TYPE B=array[1..n] of t;
```

In these cases we say that `A` is a dependent type, and that `B` is a dependent and polymorphic type.

In contrast to simply types, not any dependent type is valid. Take for example the type declaration

```
TYPE C=array[1.."five"] of bool;
```

PASCAL's compiler will produce a declaration error in type `C`, since boundaries of arrays are expected to be objects of an enumeration type.

Well-formed contexts and terms in a λ-calculus with products are defined by the grammar:

Contexts	Γ	$::=$	$nil \mid \Gamma,\ x{:}M$
Terms	M, N	$::=$	$Kind \mid Type \mid x \mid \Pi x{:}M.N \mid \lambda x{:}M.N \mid (M\ N)$

A term having the form $\Pi x{:}M.N$ is called a *product type*, and just like abstractions, it is a binding structure.

We remark that in contrast to the simply-typed case, in a λ-calculus with products, terms and types are in the same syntactical category. The terms *Kind* and *Type* are called *sorts* and they are used to stratify the terms. This stratification is necessary to avoid circular typing judgments as "$Type : Type$", which leads to the Girard's paradox. So, the term *Kind* is used as the valid type of the term *Type*.

The relation $\xrightarrow{\beta}$ is defined as the contextual closure of the rule

$$(\lambda x{:}M'.M\ N) \longrightarrow M\{x := N\}$$

.

[1] These type definitions are not supported in PASCAL.

Example 1.1 *Using the λ-calculus with products and assuming a context Γ with the declarations $nat : Type$, $bool : Type$, $1 : nat$, $10 : nat$, we can declare*

$$array : \Pi i{:}nat.\Pi j{:}nat.\Pi t{:}Type.Type \tag{1.1}$$

$$A : \Pi n{:}nat.(((array\ 1)\ n)\ bool) \tag{1.2}$$

$$B : \Pi n{:}nat.\Pi t : Type.(((array\ 1)\ n)\ t) \tag{1.3}$$

$$M : (((array\ 1)\ 10)\ bool) \tag{1.4}$$

$$M' : (A\ 10) \tag{1.5}$$

$$M'' : ((B\ 10)\ bool) \tag{1.6}$$

Since not any well-formed type is valid, neither any well-formed context is valid. We consider typing assertions having one of the following forms:

- $\vdash \Gamma$, the context Γ is valid.
- $\Gamma \vdash M : N$, the term M has type N in Γ.

Terms are stratified by means of the sorts as follows:

1. if $\Gamma \vdash M : Kind$, then we say that *M is a kind.*
2. if $\Gamma \vdash M : Type$, then we say that *M is a type.*
3. if $\Gamma \vdash M : N$ and $\Gamma \vdash N : Kind$, then we say that *M is a constructor.*
4. if $\Gamma \vdash M : N$ and $\Gamma \vdash N : Type$, then we say that *M is an object.*

Note that (2) is a particular case of (3), i.e. a type is also a constructor.

Notation: We use the lowercase letter s to range over the set $\{Kind, Type\}$.

Typing rules for contexts and terms are all mutually dependent.

Valid Contexts

$$\frac{}{\vdash nil}\ \text{(Empty)} \qquad\qquad \frac{\Gamma \vdash M : s}{\vdash \Gamma,\ x{:}M}\ \text{(Var-Decl)}$$

The Dependent Type theory [41], namely $\lambda\Pi$, is defined by the following typing rules:

Valid Terms

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathit{Type} : \mathit{Kind}}\,(\text{Type}) \qquad \frac{\vdash \Gamma,\ x{:}M}{\Gamma \vdash x : M}\,(\text{Var})$$

$$\frac{\Gamma \vdash M : \mathit{Type} \quad \Gamma,\ x{:}M \vdash N : s}{\Gamma \vdash \Pi x{:}M.N : s}\,(\text{Prod})$$

$$\frac{\Gamma \vdash M : \Pi x{:}M_1.M_2 \quad \Gamma \vdash N : M_1}{\Gamma \vdash (M\ N) : M_2\{x := N\}}\,(\text{Appl})$$

$$\frac{\Gamma \vdash M_1 : \mathit{Type} \quad \Gamma,\ x{:}M_1 \vdash M_2 : N \quad \Gamma \vdash \Pi x{:}M_1.N : s}{\Gamma \vdash \lambda x{:}M_1.M_2 : \Pi x{:}M_1.N}\,(\text{Abs})$$

In a dependent type theory, it is possible that two types syntactically different represent, via the β-conversion, the same type. Thus, we need the rule

$$\frac{\Gamma \vdash M : M_1 \quad \Gamma \vdash M_2 : s \quad M_1 \equiv_\beta M_2}{\Gamma \vdash M : M_2}\,(\text{Conv})$$

From a logical point of view, $\lambda\Pi$ extends the Curry-Howard isomorphism to universal quantifiers.

In this theory, the general polymorphism is not admitted. In particular, the bound variables are always variables of objects. This fact is clear in the premise $\Gamma \vdash M : \mathit{Type}$ of the rule (Prod), and in the premise $\Gamma \vdash M_1 : \mathit{Type}$ of the rule (Abs). Thus, a declaration as $\mathit{array} : \Pi i{:}\mathit{nat}.\Pi j{:}\mathit{nat}.\Pi t{:}\mathit{Type}.\mathit{Type}$ (Example 1.1) is not valid in $\lambda\Pi$. However, in $\lambda\Pi$ we can declare the constructor $\mathit{array}' : \Pi i{:}\mathit{nat}.\Pi j{:}\mathit{nat}.\mathit{Type}$, the type $A' : ((\mathit{array}'\ 1)\ 10)$ and the object $N : A'$.

The Calculus of Constructions [14, 15], which is an extension of $\lambda\Pi$ with polymorphism and constructions of types, is obtained by replacing the rule (Prod) and (Abs) of $\lambda\Pi$ by

$$\frac{\Gamma,\ x{:}M \vdash N : s}{\Gamma \vdash \Pi x{:}M.N : s}\,(\text{Prod}) \qquad \frac{\Gamma,\ x{:}M_1 \vdash M_2 : N \quad \Gamma \vdash \Pi x{:}M_1.N : s}{\Gamma \vdash \lambda_{M_1}.M_2 : \Pi x{:}M_1.N}\,(\text{Abs})$$

Example 1.2 *All the declarations of Example 1.1 are valid in the Calculus of Constructions. Assuming the same declaration of* Γ*, we have the valid judgments:*

$$\Gamma \vdash (((\mathit{array}\ 1)\ 10)\ \mathit{bool}) : \mathit{Type} \tag{1.7}$$

$$\Gamma \vdash ((\mathit{array}\ 1)\ n) : \Pi t{:}\mathit{Type}.\mathit{Type} \tag{1.8}$$

$$\Gamma \vdash \Pi t{:}\mathit{Type}.\mathit{Type} : \mathit{Kind} \tag{1.9}$$

Therefore, $(((\mathit{array}\ 1)\ 10)\ \mathit{bool})$ *is a type and a constructor,* $((\mathit{array}\ 1)\ n)$ *is a constructor, and* $\Pi t{:}\mathit{Type}.\mathit{Type}$ *is a kind.*

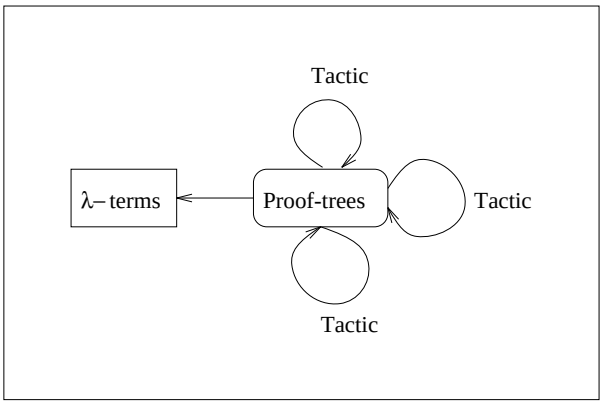


Figure 1.1: Proof construction in Coq

1.4 Construction of Proof-Terms in the Simply Typed λ-Calculus

It is clear that in an intuitionistic logic we can identify the concepts of *proofs* and *typed terms*. Now, we are interested in computer assistance to the construction process of these mathematical entities.

Generally, proofs are developed in a bottom-up way. For example, if we want to prove a proposition of the form $A \rightarrow B$, then we use (Intro$^{\rightarrow}$) and we try to prove B under the assumption A. In contrast, terms construction is essentially a top-down process. For example, if we know that M is a term of type B in a context where x is declared of type A, then we conclude that the term $\lambda x{:}A.M$ has type $A \rightarrow B$ by using the rule (Abs).

Assume that we want to find a term of type $A \rightarrow B$ in a context Γ and we do a bottom-up application of (Abs). We would like to say that the term $\lambda x{:}A.X_?$ is a proof-term of $A \rightarrow B$, where $X_?$ is any term of type B in the context Γ, $x{:}A$. The symbol $X_?$ is a *place-holder* and the term $\lambda x{:}A.X_?$ is an *incomplete proof*. Incomplete proofs represent hypothetical future proofs. We can think of place-holders as lemmas that are used in the proof, but which are not yet proved. In this way, it is possible to build proof-terms incrementally.

From a theoretical point of view, what is the status of a term like $\lambda x{:}A.X_?$ in λ-calculus? In certain proof systems, proofs and incomplete proofs are represented in different structures. For example, in the system Coq [17], proofs are represented by typed terms and incomplete proofs are represented by structures called *proof-trees* [72]. Proof-trees are refined by construction steps, namely *tactics*, and only at the end of the proving process, proof-trees representing complete proofs are translated to terms. Term construction process in Coq is shown in Fig. 1.1.

In other systems, e.g Alf [2], proofs and incomplete proofs share the same internal representation: terms of a typed λ-calculus. Incomplete proofs are terms with place-holders, as $\lambda x{:}A.X_?$, and they are refined by a mechanism of instantiation of place-holders. For example $\lambda x{:}A.X_?$ becomes $\lambda x{:}A.y$ by the instantiation of $X_?$ with y. Term construction process in Alf is shown in Fig. 1.2.

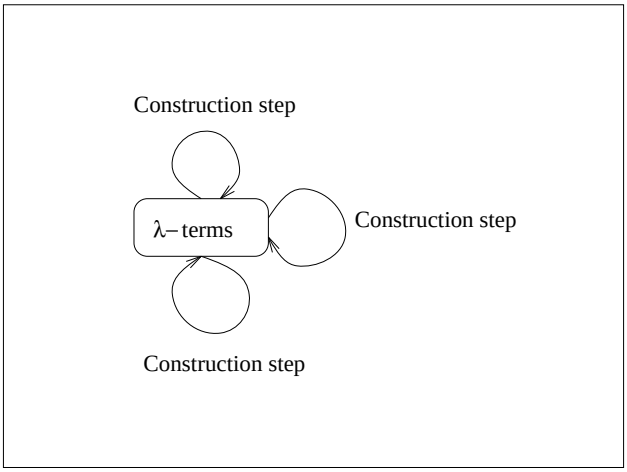


Figure 1.2: Term construction in Alf

1.4.1 Incomplete Proofs and Other Demons

In λ-calculus there is a notion of variable and a mechanism to substitute it. Are they sufficient to represent place-holders and their refinement mechanism? We are going to answer this question with some examples.

Assume that we have a hypothesis h of the proposition B and we want to find a proof of $A \rightarrow B$. We propose the incomplete proof $\lambda x{:}A.X_?$. If we substitute $X_?$ by h, then we obtain the term $\lambda x{:}A.h$ which does not contain place-holders anymore. Note that $\lambda x{:}A.h$ has type $A \rightarrow B$, since h has type B in the context. Thus, we can say that $\lambda x{:}A.h$ is a proof of $A \rightarrow B$.

Now, assume that we want to prove $A \rightarrow A$. Again, we propose the incomplete proof $\lambda x{:}A.X_?$. We would like to substitute $X_?$ by x in order to obtain something like $\lambda x{:}A.x$. However, x is a bound variable in $\lambda x{:}A.X_?$, so the result of this substitution is $\lambda y{:}A.x$ since the substitution operation renames bound variables to avoid capture of free variables. The term $\lambda y{:}A.x$ is ill-typed since x is not declared in the context. In addition, we cannot expect to get the term $\lambda x{:}A.x$ by means of a substitution mechanism.

The above remark is not new. The Huet's higher-order unification algorithm [46] uses a functional coding of place-holders in order to deal with that problem. If we apply this technique to our example, then we do not consider the incomplete term $\lambda x{:}A.X_?$, but the term $\lambda x{:}A.(Y_?\ x)$. In this case, $Y_?$ has type $A \rightarrow A$, while $X_?$ has type A. Now, to complete the proof, we can substitute $Y_?$ for any term of type $A \rightarrow A$. For instance, we can substitute $Y_?$ by $\lambda y{:}A.y$ to obtain a well-typed term $\lambda x{:}A.(\lambda y{:}A.y\ x)$. This technique works fine in unification algorithms. However, it does not allow incremental construction of proofs. Notice that a solution for $Y_?$ is also a solution for the initial proposition $A \rightarrow A$.

Therefore, if we use the standard λ-calculus to represent incomplete proofs, and its substitution mechanism to refine place-holders, then we cannot build proofs incrementally. In particular, it is not possible to simulate the `Intro` tactic that exists in many proof assistant systems —the `Intro` tactic corresponds to the application bottom-up of (Abs).

Actually, to build a proof of $A \rightarrow A$ from the incomplete proof $\lambda x{:}A.X_?$, we do not want to

substitute $X_?$ by x, but to replace naively the place-holder $X_?$ by x. Thus, we can imagine a λ-calculus with another operation that we call *instantiation*. Instantiation replaces a variable by a term without taking care of possible captures of free variables.

Unfortunately, as explained in [25], instantiation and β-reduction do not commute. For instance the term $(\lambda x{:}A.X_?\ y)\{X_?/x\}$ —where $\{X_?/x\}$ denotes the instantiation of $X_?$ with x— is β-equivalent to $X_?\{X_?/x\}$ which is equivalent to x, but if we perform the instantiation before the β-reduction, we obtain $(\lambda x{:}A.x\ y)$ which is β-reduced to y. When we β-reduce incomplete proofs we may lose information about the scoping of place-holders. In the later example, we would like the incomplete proof $(\lambda x{:}A.X_?\ y)$ reduces to $X_?\{x := y\}$, but where the substitution is in a stand-by state waiting for an instantiation of $X_?$. Unfortunately, the substitution in λ-calculus is an atomic external operation which cannot be delayed.

1.4.2 Explicit Substitutions

Calculi of explicit substitutions [1] support a lazy mechanism of reduction of substitutions. In these calculi, β-redexes are reduced by a rule of the form: $(\lambda x{:}A.M\ N) \xrightarrow{\text{(Beta)}} M[x := N]$, where $M[x := N]$ is not a notation but a well-formed syntactic expression. Thus, in calculi of explicit substitutions, it is necessary to have a rewrite system to effectively reduce substitutions. Usual substitutions rules are $x[x := N] \xrightarrow{(\text{Var}_1)} N$ and $y[x := N] \xrightarrow{(\text{Var}_2)} y$ if $x \neq y$. There are several presentation of calculi of explicit substitutions, see for example [20] for an overview of some of them, but also, among others [1, 81, 58, 49, 10, 60, 53, 67, 30, 70].

However, a calculus of explicit substitutions is not sufficient to express incremental construction of proofs by means of refinement steps. It is also necessary to delay the effective application of a substitution to a place-holder. Notice that if $X_?$ is a variable, then $X_?[x := N]$ will be reduced by (Var_1) or by (Var_2), but we want the term $X_?[x := N]$ in a stand-by state, waiting for an instantiation of $X_?$.

1.4.3 Place-Holders as Meta-Variables

Variables and place-holders play different roles in terms: variables denote constant and bound variables, while place-holders are unknown pieces of terms waiting for instantiations. If we consider the λ-calculus defined as an algebra on a set $\mathcal{X}$ of variables and a set of operators containing a set of constants $\mathcal{V}$, a set of unary abstractors indexed on $\mathcal{V}$, and the application operator, then we have two sort of "variables":

- those of $\mathcal{V}$ which correspond to the variables of the calculus. We use lowercase letters $x, y, \ldots$ to denote them, for example x in the term $\lambda x{:}A.x$, and

- those of $\mathcal{X}$ which denote arbitrary terms of the algebra and are usually called *meta-variables*. We use uppercase letters $X, Y, \ldots$ to denote them.

The expected behavior of place-holders is closer to that of meta-variables than to that of variables. Thus, incomplete proofs can be represented by *open terms*, i.e. terms with meta-variables, in a λ-calculus with explicit substitutions.

1.4.4 Refining Place-Holders

When we consider place-holders as meta-variables, refinement of incomplete proofs becomes instantiation of meta-variables. Soundness of the construction process can be guaranteed by a commutation property on refinements.

Definition 1.3 *A refinement operation* commutes *with typing if and only if for any terms M and N such that M is refined into N, if $\Gamma \vdash M : A$, then $\Gamma \vdash N : A$*

The general mechanism of instantiation of meta-variables does not commute with typing rules. A first restriction is to instantiate a meta-variable X only with terms having the same type as X. But this restriction is not even sufficient. For instance, consider the valid typing judgments (where the meta-variable X is assumed to be of type A):

$$z{:}(B{\to}A){\to}C,\ x{:}A \vdash (z\ \lambda x{:}B.X) : C \tag{1.10}$$

$$z{:}(B{\to}A){\to}C,\ x{:}A \vdash x : A \tag{1.11}$$

If we instantiate X with x in Eq. 1.10, then we get the ill-typed term $(z\ \lambda x{:}B.x)$ in Γ, since $\lambda x{:}B.x$ has type $B{\to}B$ and not $B{\to}A$ as expected by z.

The above example shows that it is also necessary to define a context to type all the instances of a meta-variable. In the type systems with open terms proposed in [25, 62], each meta-variable X has associated a *unique* type A_X, a *unique* context Γ_X and the implicit typing rule:

$$\frac{}{\Gamma_X \vdash X : A_X}\,(\mathrm{Meta}_X)$$

In other words, all the occurrences of a meta-variable must be typed in the same context and with the same type. For instance, the judgment $w{:}A \vdash (\lambda x{:}A.(Y\ w)\ Z) : B$ is valid if and only if we consider the following typing rules:

$$\frac{}{w{:}A,\ x{:}A \vdash Y : A{\to}B}\,(\mathrm{Meta}_Y) \qquad\qquad \frac{}{w{:}A \vdash Z : A}\,(\mathrm{Meta}_Z)$$

In contrast, for any context Γ, the term $(\lambda x{:}A.X)\ X$ is not well-typed, since in this case X would be typed in two different contexts: Γ and $\Gamma,\ x{:}A$.

Definition 1.4 *An instantiation $\{X/M\}$ is* well-defined *when M is a term such that $\Gamma_X \vdash M : A_x$.*

An additional point is that the typing context of a meta-variable may not be preserved by β-reductions. Take for example the valid judgment

$$\Gamma \vdash (\lambda x{:}A.\lambda y{:}B.x\ X) : B{\to}A$$

where X is a meta-variable assumed to be of type A in Γ. In λ-calculus, we obtain by one-step β-reduction the term $\lambda y{:}B.X$. In this term, the meta-variable X would be typed in the context $\Gamma,\ y{:}B$, which is different to Γ. Magnusson proposes in [62] to avoid the distribution of substitutions inside abstractions. In this case, $(\lambda x{:}A.\lambda y{:}B.x\ X)$ reduces to $(\lambda y{:}B.x)[x := Y]$ which is no more reducible.

Another approach is to use a calculus of explicit substitutions with a an explicit mechanism of renaming, e.g the $\lambda\sigma$-calculus [1]. The origin of the $\lambda\sigma$-calculus was the Categorical Combinators Logic (CCL) introduced by Curien in [19] as a syntactical model for the Cartesian Closed Categories. In the CCL the substitution mechanism is fully implemented by means of a rewrite system. The $\lambda\sigma$-calculus is an evolution of CCL with two different sorts to represent substitutions and terms.

Since the renaming operation is handled as an explicit substitution in $\lambda\sigma$, the typing context of meta-variables is preserved under $\lambda\sigma$-reductions and even when substitutions go through abstractions. In fact, $\lambda\sigma$ satisfies the following property.

Proposition 1.5 *Typing and instantiation of meta-variables commute in the simply-typed $\lambda\sigma$-calculus.*

Proof. See [25]. □

1.5 Towards the Construction of Proof-Terms in Dependent Types

The main goals of this thesis are to develop a formal framework to represent proof-terms with place-holders in theories with dependent types (as $\lambda\Pi$ or the Calculus of Constructions), and to show how it can be used in a proof synthesis method.

This thesis is structured in two parts. In the First Part, which is composed of chapters 2 to 4, we propose the system $\mathrm{CC}_{\mathcal{L}}$ as a formalism to represent incomplete proof-terms in a type theory with dependent types, polymorphism and constructions of types.

We have seen in the previous section that calculi of explicit substitutions with meta-variables offer a good framework to represent proof-terms with place-holders in the simply-typed λ-calculus. In Chapter 2 we study several type free calculi of explicit substitutions, and we compare their characteristics according to the suitability to reach our goal. In particular, we solve positively a conjecture about confluence and normalization properties in calculi of explicit substitutions, and we propose a simple left-linear calculus that preserves the nice properties of $\lambda\sigma$.

The general idea of place-holders as meta-variables, and refinement steps as instantiations, can be extended to build proof-terms in dependent type theories as shown by the following example.

Example 1.6 *To prove the first-order proposition $(\forall x{:}nat.(x \geq 0))\rightarrow(0 \geq 0)$, we need to find a term X of type $(\Pi x{:}nat.(x \geq 0))\rightarrow(0 \geq 0)$. If we apply (Abs) in a bottom-up way, then we can instantiate X with $\lambda y{:}(\Pi x{:}nat.(x \geq 0)).Y$ where Y is a meta-variable of type $(0 \geq 0)$ in a context where the variable y has the type $\Pi x{:}nat.(x \geq 0)$. Now, we can instantiate Y with the term $(y\ 0)$ which is a well-typed term of type $(0 \geq 0)$. We get the proof $\lambda y{:}(\Pi x{:}nat.(x \geq 0)).(y\ 0)$ of type $(\Pi x{:}nat.(x \geq 0))\rightarrow(0 \geq 0)$.*

However, as it is pointed out by Magnusson in [62], explicit substitution in a dependent type theory is not simple to formulate. Assume for example the induction axiom over the natural numbers, i.e.

$$\begin{aligned}\Gamma = \quad & nat{:}Type,\ 0{:}nat,\ succ{:}nat\rightarrow nat.\\ & nat\_ind{:}(\Pi p{:}nat\rightarrow Type.((p\ 0)\rightarrow(\Pi x{:}nat.(p\ x)\rightarrow(p\ (succ\ x)))\rightarrow\Pi n{:}nat.(p\ n)))\end{aligned}$$

If P, P_0, P_s and N are meta-variables, then, according to the rules (Abs_Π) and (Appl_Π), the term

$$((((nat\_ind\ P)\ P_0)\ \lambda x{:}nat.\lambda y{:}(P\ x).P_s)\ N)$$

is well-typed in Γ if

1. $\Gamma \vdash P : nat \rightarrow Type$,
2. $\Gamma \vdash P_0 : (P\ 0)$,
3. $\Gamma,\ x{:}nat,\ y{:}(P\ x) \vdash P_s : (P\ (succ\ x))$, and
4. $\Gamma \vdash N : nat$.

In this example, meta-variables appear in types, but also in contexts; for instance in the case 3, the meta-variable P appears in the context $\Gamma,\ x{:}nat,\ y{:}(P\ x)$, and in the type $(P\ (succ\ x))$. In other examples, typing judgments must take into account harmful circular dependences like:

- $\Gamma_X \vdash X : (w\ X)$
- $\Gamma_X \vdash X : (w\ Y)$ and $\Gamma_Y \vdash Y : (z\ X)$
- $\Gamma_X,\ y{:}(w\ X) \vdash X : (z\ y)$

These considerations are discussed in Chapter 3, where we show how typing judgments can be formulated on theories with dependent types in order to deal with meta-variables and explicit substitutions. In particular, we propose the system $CC_{\mathcal{L}}$ which is a formulation of the calculus of constructions with explicit substitutions and typed meta-variables.

A proof synthesis method for $CC_{\mathcal{L}}$ is proposed in Chapter 4. This method merges a procedure of term enumeration, with a technique of higher-order unification via explicit substitutions where unification variables are encoded by meta-variables.

In the Second Part of this thesis, which is composed of chapters 5 to 7, we show the main meta-theoretical properties of the system $CC_{\mathcal{L}}$.

The calculus of pre-expressions with explicit substitutions, i.e. expressions with type annotations but not necessarily well-typed, is not confluent. The problem is similar to that pointed out by Nederpelt in the λ-calculus extended with the η-rule. Geuvers proposes in [32] a method to prove confluence $\beta\eta$-reduction on typed λ-terms. Geuvers' technique is based on confluence of the underlying calculus without type annotations. In Chapter 5 we show how to adapt Geuvers' technique to calculi of explicit substitutions.

Chapter 6 studies the elementary typing properties of $CC_{\mathcal{L}}$, i.e. sort soundness, type uniqueness, subject reduction and typing soundness.

Melliès has shown in [64, 63] that typed $\lambda\sigma$-calculi may not terminate. Melliès counter-example can be adapted for $CC_{\mathcal{L}}$. Therefore, system $CC_{\mathcal{L}}$ does not enjoy strong normalization property. However, we show in Chapter 7 that there exists an strategy to find normal forms, and thus well-typed expressions in $CC_{\mathcal{L}}$ are weakly normalizing. The proof we propose follows the proof of strong normalization for the calculus of constructions given by Geuvers in [32].

Church-Rosser and confluence properties of $CC_{\mathcal{L}}$ are proved in Chapter 8. As a corollary of these properties, we have the decidability of the conversion.

A discussion of some works about meta-variables and proof construction issues based on other formalisms is presented in Chapter 9. This final chapter also summarizes the main contributions of this thesis, and raises possible issues to be further studied.

Part I

A Calculus of Substitutions and Its Applications

Chapter 2

Type Free Calculi of Explicit Substitutions

We have seen in the previous chapter that the λ-calculus uses an external and atomic operation to compute the substitution of variables by terms. Thus, in λ-calculus is not possible to delay the application of a substitution to a term or to consider terms with partially applied substitutions.

Calculi of explicit substitutions improve the description of the substitution mechanism by allowing substitutions to be part of the formal language. In particular, in λ-calculi with explicit substitutions, the β-reduction is implemented by means of internal constructors of substitutions, and reduction rules to compute with them.

Calculi of explicit substitutions come in two flavors. Those where the renaming mechanism is internalized inside the calculus, as for example the calculi of the $\lambda\sigma$-family [1, 20] which use a De Bruijn notation of variables, and those where the α-conversion is implicitly assumed, for instance the λx-calculus [10] which uses named variables.

In order to reach our but (representation of proof-terms with place-holders), we prefer to use calculi where not only substitutions are made explicit, but also renaming. We justify this choice by the following reasons:

- The renaming mechanism can be seen as a particular case of substitution. For instance, in the $\lambda\sigma$-family the update operation of variables is just an explicit substitution.
- The λ-calculus with names is a suitable notation for human reading. However, the α-conversion is a frequent source of counter-examples in λ-calculus. In contrast, calculi which handle explicitly the renaming operation are less natural for humans, but formal developments are cleaner. As we have suggested in Chapter 1, in order to have good properties in a λ-calculus with holes, it is necessary to deal with scoping problems. Many of these problems are solved in calculi where renaming is made explicit.

In this chapter we do not consider typing information. Section 1 presents the state of the art of calculi of the so-called $\lambda\sigma$-family. In Section 2, we solve positively a conjecture about the existence of a calculus of explicit substitutions which enjoys confluence and normalization properties. In Section 3 we propose a calculus of the $\lambda\sigma$-family which enjoys the same general properties as $\lambda\sigma$, but in contrast to it, the new calculus, namely $\lambda_{\mathcal{L}}$, is left-linear. Non left-linear rules in calculi of explicit substitutions are source of non-confluence problems [20], but also they raise technical

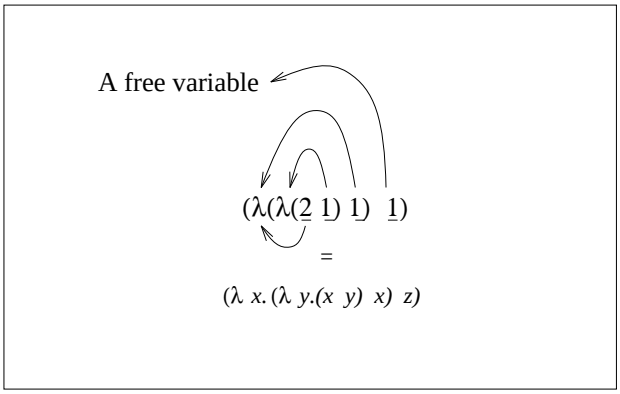


Figure 2.1: De Bruijn indices notation

problems for implementations [74], unification algorithms [55] and type systems [69]. The typed versions of the $\lambda_{\mathcal{L}}$-calculus are at the basis of our rerpresentation of incomplete proof-terms.

2.1 The Family of $\lambda\sigma$-Calculi

The calculi of the family of $\lambda\sigma$ are characterized by two features:

- they use a de Bruijn indices notation of variables, and
- they are described by first-order rewrite systems with different sorts for terms and substitutions.

2.1.1 The λ-Calculus *à la* de Bruijn

The λ-calculus *à la* de Bruijn [21], namely λ_{dB}, uses a nameless representation of variables: the occurrences of free and bound variables are denoted by natural numbers, usually called the de Bruijn's indices.

Well-formed expressions in λ_{dB} are defined by the following grammar:

Natural Numbers	n	::=	$1 \mid 2 \mid \ldots$
Terms	M, N	::=	$\underline{n} \mid (M\ N) \mid \lambda M$

The λ-*height* of an occurrence u in M is the number of λ-symbols in the binding scope of u. Let u be the occurrence of an index $\underline{n}$ in M, and m be the λ-height of u. If $n \leq m$, then u is bound to the n-th innermost λ-symbol of its binding scope. Otherwise, u represents the $(n-m)$-th free variable of a general context. The diagram of Fig. 2.1 shows the imaginary binding links for the term $(\lambda x.(\lambda y.(x\ y)\ x)\ z)$. Note that the same index appearing in different binding scopes, represent different variables, and occurrences of the same variable appearing in different binding scopes, are denoted by different indices.

The formal definition of β-reduction in λ_{dB} is more complex than in λ-calculus with names, due to the fact that the renaming mechanism is internal to the calculus.

Definition 2.1 *The* β-reduction *is defined in* λ_{dB} *as (c.f. [20]):*

$$(\lambda M\ N) \xrightarrow{\beta} M\{1 := N\}$$

where $M\{n := N\}$ *is inductively defined for* $n > 0$ *as*

$$\begin{array}{lcl} (M\ M')\{n := N\} & = & (M\{n := N\}\ M'\{n := N\}) \\ (\lambda M)\{n := N\} & = & \lambda M\{n{+}1 := N\} \\ \underline{m}\{n := N\} & = & \begin{cases} \underline{m-1} & \text{if } m > n \\ \tau_0^{n-1}(M) & \text{if } m = n \\ \underline{m} & \text{if } m < n \end{cases} \end{array}$$

and τ_i^n *is defined for* $i, n \geq 0$ *as*

$$\begin{array}{lcl} \tau_i^n(M\ N) & = & (\tau_i^n(M)\ \tau_i^n(N)) \\ \tau_i^n(\lambda M) & = & \lambda \tau_{i+1}^n(M) \\ \tau_i^n(\underline{m}) & = & \begin{cases} \underline{m+n} & \text{if } m > i \\ \underline{m} & \text{if } m \leq i \end{cases} \end{array}$$

Informally, a substitution in λ_{dB} can be seen as an infinite list of terms $M_1.M_2\ldots$, where the index $\underline{1}$ will be substituted by M_1, $\underline{2}$ by M_2 and so on. In this way, the substitution $\{n := N\}$ is the infinite list of terms $\underline{1}\ldots\underline{n{-}1}.N.\underline{n}.\underline{n{+}1},\ldots$. Notice that in order to compute $M\{n := N\}$, not only the indices of M need to be updated, but also the indices of N. This updating operation on N is performed by the function $\tau_i^n(.)$. Thus, $\tau_0^n(N)$ increases by n the indices representing free variables in N, and $\tau_i^n(N)$ is obtained after crossing i λ's between the root of the substituted term and its occurrence.

2.1.2 The λσ-Calculus

The *λσ-calculus* is one of the most popular calculi of explicit substitutions. It is a first-order rewrite system with two sorts of expressions: terms and substitutions. Well-formed expressions in $\lambda\sigma$ are defined by the following grammar:

Terms	M, N	::=	$\mathbf{1} \mid (M\ N) \mid \lambda M \mid M[S]$
Substitutions	S, T	::=	$id \mid \uparrow \mid M \cdot S \mid S \circ T$

Substitutions in $\lambda\sigma$ are part of the formal language. For instance, the well-formed term $M[S]$ denotes the explicit application of substitution S to M.

Substitutions of $\lambda\sigma$ can be described informally as follows:

- id, usually called *identity*, denotes the substitution $\underline{1}.\underline{2}.\underline{3}\ldots$
- $\uparrow$, usually called *shift*, denotes the substitution $\underline{2}.\underline{3}.\underline{4}\ldots$
- $S \circ T$ denotes the *composition* of substitutions T and S, i.e. if S denotes the substitution $M_1.M_2.M_3\ldots$, $S \circ T$ denotes the substitution $M_1[T].M_2[T].M_3[T]\ldots$
- $M \cdot S$ denotes a simultaneous substitution formed by a *cons* of M to S, i.e. if S denotes the substitution $M_1.M_2.M_3\ldots$, $M \cdot S$ denotes the substitution $M.M_1.M_2.M_3\ldots$

$(\lambda M\ N)$	$\longrightarrow$	$M[N \cdot id]$	(Beta)
$(M\ N)[S]$	$\longrightarrow$	$(M[S]\ N[S])$	(Application)
$(\lambda M)[S]$	$\longrightarrow$	$\lambda M[\mathbf{1} \cdot (S \circ \uparrow)]$	(Lambda)
$M[S][T]$	$\longrightarrow$	$M[S \circ T]$	(Clos)
$\mathbf{1}[M \cdot S]$	$\longrightarrow$	M	(VarCons)
$M[id]$	$\longrightarrow$	M	(Id)
$(S \circ S') \circ T$	$\longrightarrow$	$S \circ (S' \circ T)$	(Ass)
$(M \cdot S) \circ T$	$\longrightarrow$	$M[T] \cdot (S \circ T)$	(Map)
$id \circ S$	$\longrightarrow$	S	(Idl)
$S \circ id$	$\longrightarrow$	S	(Idr)
$\uparrow \circ (M \cdot S)$	$\longrightarrow$	S	(ShiftCons)
$\mathbf{1} \cdot \uparrow$	$\longrightarrow$	id	(VarShift)
$\mathbf{1}[S] \cdot (\uparrow \circ S)$	$\longrightarrow$	S	(SCons)

Figure 2.2: The rewrite system $\lambda\sigma$

De Bruijn's indices are encoded in $\lambda\sigma$ by using the constant $\mathbf{1}$ and the substitution $\uparrow$ as follows: $\underline{1} \stackrel{\text{Def}}{=} \mathbf{1}$, and $\underline{n+1} \stackrel{\text{Def}}{=} \mathbf{1}[\overbrace{\uparrow \circ (\ldots \circ (\uparrow \circ \uparrow))}^{n\text{-times}}]$ for $n > 0$.

The rewrite system $\lambda\sigma$ is presented in Fig. 2.2. The σ-calculus is obtained by dropping (Beta) from $\lambda\sigma$. Terms of λ_{dB} are represented by *ground* terms (i.e. terms without meta-variables) in σ-normal form, and one step of β-reduction is simulated by one (Beta)-reduction followed by a normal σ-reduction.

Proposition 2.2 *The σ-calculus is confluent and terminating.*

Proof. See [39, 81, 91, 92]. □

The $\lambda\sigma$-calculus lacks some nice properties: it is not confluent (on general open expressions) [20], and it does not preserve strong normalization of λ-calculus (i.e. strongly normalizing terms in λ_{dB} are not necessarily strongly normalizing in $\lambda\sigma$) [64]. However, the confluence property holds in $\lambda\sigma$ for ground expressions (i.e. expressions without meta-variables) [1], and for *semi-open expressions* (i.e. expressions with meta-variables of terms but no meta-variables of substitutions) [81]. In addition, the $\lambda\sigma$-calculus is weakly terminating on typed expressions [1], and this property even holds if we consider meta-variables [38, 70].

2.1.3 A confluent variant of $\lambda\sigma$: the $\lambda\sigma_{\Uparrow}$-Calculus

The composition operator was introduced in $\lambda\sigma$ to solve a critical pair, and so, to gain local confluence. Moreover, composition of substitutions is responsible of the following non-left-linear rule in $\lambda\sigma$: $\mathbf{1}[S] \cdot (\uparrow \circ S) \xrightarrow{\text{(SCons)}} S$. Informally, if we interpret S as a list, $\mathbf{1}$ as the head function

and $\uparrow$ as the tail function, then this rule corresponds to the surjective-pairing rule of the λ-calculus. As conjectured in [1], and latter proved in [20], (SCons) destroys confluence in $\lambda\sigma$.

The rule (SCons) appears in $\lambda\sigma$ when the identity substitution goes through an abstraction. In this case we have the following critical pair between the rules (Lambda) and (Id):

$$\begin{array}{ccc} & (\lambda M)[id] & \\ \text{(Lambda;Idl)} \swarrow & & \searrow \text{(Id)} \\ \lambda M[\mathbf{1}\cdot\uparrow] & & \lambda M \end{array}$$

This critical pair suggests the following contraction rule: $\mathbf{1}\cdot\uparrow \xrightarrow{\text{(VarShift)}} id$. Finally, (SCons) is introduced to solve a critical pair between the rules (VarShift) and (Map).

Curien, Hardin and Lévy [20] propose the *$\lambda\sigma_{\Uparrow}$-calculus*. In order to avoid the rules (VarShift) and (SCons) in $\lambda\sigma_{\Uparrow}$, substitutions go through abstractions by using a special substitution operator $\Uparrow(.)$, usually called *lift*. Thus, (Lambda) is written in $\lambda\sigma_{\Uparrow}$ as $(\lambda M)[S] \xrightarrow{\text{(Lambda)}} \lambda M[\Uparrow(S)]$. From an extensional point of view, the substitution $\Uparrow(S)$ of $\lambda\sigma_{\Uparrow}$ is equivalent to the substitution $\mathbf{1}\cdot(S\circ\uparrow)$ of $\lambda\sigma$, but this equivalence is not in the theory of $\lambda\sigma_{\Uparrow}$. However, in ground terms it holds $M[\Uparrow(S)] =_{\lambda\sigma_{\Uparrow}} M[\mathbf{1}\cdot(S\circ\uparrow)]$.

The set of well-formed expressions in $\lambda\sigma_{\Uparrow}$ is defined by the following grammar:

$$\begin{array}{lllll} \textbf{Terms} & M, N & ::= & \mathbf{1} \mid \lambda M \mid (M\ N) \mid M[S] \\ \textbf{Substitutions} & S, T & ::= & id \mid \Uparrow(S) \mid\ \uparrow\ \mid M\cdot S \mid S\circ T \end{array}$$

The $\lambda\sigma_{\Uparrow}$-calculus is given by the rewrite system of Fig. 2.3.

In $\lambda\sigma_{\Uparrow}$, the critical pair between (Lambda) and (Id) is solved with the rule (LiftId). In contrast to $\lambda\sigma$, the $\lambda\sigma_{\Uparrow}$-calculus is a left-linear rewrite system.

Proposition 2.3 (Confluence) *The $\lambda\sigma_{\Uparrow}$-calculus is confluent (on open expressions).*

Proof. See [20]. □

2.1.4 Preservation of Strong Normalization: the $\lambda\upsilon$-Calculus

A calculus *preserves strong normalization* (PSN) if its set of strongly normalizing terms contains the set of strongly normalizing terms of λ-calculus.

Melliès [64] has shown that neither $\lambda\sigma$ nor $\lambda\sigma_{\Uparrow}$ preserve strong normalization. In fact, he proves that a minimal $\lambda\sigma$-calculus formed by (Beta), (Application), (Lambda), (Clos) and (Map) may not terminate on strongly normalizing λ_{dB}-terms. The counter-example does not depend on the notation used to represent variables (de Bruijn's indices or names) as shown by Bloo [7]. The composition of substitutions, and in particular the rule (Map), seems to play an essential role in the counter-example.

Lescanne [58] has remarked that composition is not necessary to retain confluence on ground terms in calculi of explicit substitutions. He proposed the *$\lambda\upsilon$-calculus*. Since this calculus does not use composition of substitutions, Melliès' counter-example does not apply in $\lambda\upsilon$.

The $\lambda\upsilon$-calculus uses the lift operator to deal with substitutions applied to abstractions, and an operator to express singleton substitutions: ./. This operator was introduced originally by Ehrhard

$(\lambda M\ N)$	$\longrightarrow$	$M[N \cdot id]$	(Beta)
$(M\ N)[S]$	$\longrightarrow$	$(M[S]\ N[S])$	(Application)
$(\lambda M)[S]$	$\longrightarrow$	$\lambda M[\Uparrow(S)]$	(Lambda)
$M[S][T]$	$\longrightarrow$	$M[S \circ T]$	(Clos)
$\mathbf{1}[M \cdot S]$	$\longrightarrow$	M	(VarCons)
$\mathbf{1}[\Uparrow(S)]$	$\longrightarrow$	$\mathbf{1}$	(VarLift1)
$\mathbf{1}[\Uparrow(S) \circ T]$	$\longrightarrow$	$\mathbf{1}[T]$	(VarLift2)
$M[id]$	$\longrightarrow$	M	(Id)
$(S \circ S') \circ T$	$\longrightarrow$	$S \circ (S' \circ T)$	(Ass)
$(M \cdot S) \circ T$	$\longrightarrow$	$M[T] \cdot (S \circ T)$	(Map)
$\Uparrow(S) \circ \Uparrow(T)$	$\longrightarrow$	$\Uparrow(S \circ T)$	(Lift1)
$\Uparrow(S) \circ (\Uparrow(T) \circ S')$	$\longrightarrow$	$\Uparrow(S \circ T) \circ S'$	(Lift2)
$\Uparrow(S) \circ (M \cdot T)$	$\longrightarrow$	$M \cdot (S \circ T)$	(LiftCons)
$id \circ S$	$\longrightarrow$	S	(Idl)
$S \circ id$	$\longrightarrow$	S	(Idr)
$\Uparrow(id)$	$\longrightarrow$	id	(LiftId)
$\uparrow \circ (M \cdot S)$	$\longrightarrow$	S	(ShiftCons)
$\uparrow \circ \Uparrow(S)$	$\longrightarrow$	$S \circ \uparrow$	(ShiftLift1)
$\uparrow \circ (\Uparrow(S) \circ T)$	$\longrightarrow$	$S \circ (\uparrow \circ T)$	(ShiftLift2)

Figure 2.3: The rewrite system $\lambda\sigma_{\Uparrow}$

$(\lambda M\ N)$	$\longrightarrow$	$M[N/]$	(Beta)
$(M\ N)[S]$	$\longrightarrow$	$(M[S]\ N[S])$	(Application)
$(\lambda M)[S]$	$\longrightarrow$	$\lambda M[\Uparrow(S)]$	(Lambda)
$\underline{1}[M/]$	$\longrightarrow$	M	(FVar)
$\underline{n+1}[M/]$	$\longrightarrow$	$\underline{n}$	(RVar)
$\underline{1}[\Uparrow(S)]$	$\longrightarrow$	$\underline{1}$	(FVarLift)
$\underline{n+1}[\Uparrow(S)]$	$\longrightarrow$	$\underline{n}[S][\uparrow]$	(RVarLift)
$\underline{n}[\uparrow]$	$\longrightarrow$	$\underline{n+1}$	(VarShift)

Figure 2.4: The rewrite system λv

[27], and used by Rios [81] to define the $\lambda\tau$-calculus. The substitution $M/$ can be seen extensionally equivalent to the substitution $M \cdot id$ of $\lambda\sigma$.

The set of well-formed expressions in λv is defined by the following grammar:

Naturals	n	::=	$1 \mid n+1$
Terms	M, N	::=	$\underline{n} \mid \lambda M \mid (M\ N) \mid M[S]$
Substitutions	S	::=	$M/ \mid \uparrow \mid \Uparrow(S)$

The λv-calculus is given by the rewrite system of Fig. 2.4.

Although, λv is confluent on ground expressions [58], it is not confluent, not even locally confluent, on open expressions. To see why, consider e.g. the term $(\lambda M\ N)[S]$. It reduces via (Beta) to $M[N/][S]$, but also via (Application;Lambda;Beta) to $M[\Uparrow(S)][N[S]/]$. This critical pair cannot be joined by the rewrite system. However, the substitution lemma in [59] shows that it is joinable on ground terms.

Proposition 2.4 (Preservation of Strong Normalization) *If M is strongly normalizing in λ_{dB}, then M is a strongly normalizing in λv.*

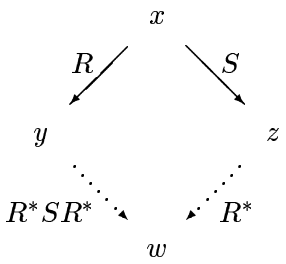
Proof. See [6]. □

We finish this first section with some properties under abstract relations and rewrite systems which happen to be useful to prove confluence and termination properties in calculi of the $\lambda\sigma$-family.

2.1.5 Useful Properties of Abstract Relations and Rewrite Systems

Assume that R and S are relations defined on the set X.

Definition 2.5 (Yokouchi-Hikita's Commutation) *We say that R* YH-commutes *over S if and only if for any $x, y, z \in X$ such that $x \xrightarrow{R} y$ and $x \xrightarrow{S} z$, there exists $w \in X$ such that $y \xrightarrow{R^* S R^*} w$ and $z \xrightarrow{R^*} w$, i.e. the following diagram holds:*

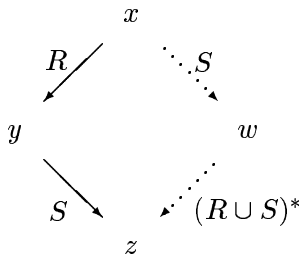


Lemma 2.6 (Yokouchi-Hikita's Lemma) *Let R and S be two relations such that: 1. R is confluent and terminating, 2. S is strongly confluent and 3. R YH-commutes over S, then the relation R^*SR^* is confluent.*

Proof. See [20]. □

The Yokouchi-Hikita's lemma [90] has been used in [20, 76, 67, 70] to prove confluence in left-linear calculi of explicit substitutions.

Definition 2.7 (Bachmair-Dershowitz's Commutation) *We say that R* BD-commutes *over S if and only if for any $x, y, z \in X$ such that $x \xrightarrow{R} y$ and $y \xrightarrow{S} z$, there exists $w \in X$ such that $x \xrightarrow{S} w$ and $w \xrightarrow{(R \cup S)^*} z$, i.e. the following diagram holds:*



Lemma 2.8 (Bachmair-Dershowitz's Lemma) *Let R and S be two relations such that: 1. R and S are terminating and 2. R BD-commutes over S, then the relation $R \cup S$ is terminating.*

Proof. See [22]. □

The Bachmair-Dershowitz' lemma also holds to prove preservation of strong normalization as shown by Lemma 2.10. We need the following consequence of BD-commutation.

Lemma 2.9 *If R BD-commutes over S, then for any x, y, z such that $x \xrightarrow{R^*} y \xrightarrow{S} z$, there exists w such that $x \xrightarrow{S} w \xrightarrow{(R \cup S)^*} z$.*

Proof. We reason by induction on the length of the reduction $x \xrightarrow{R^*} y$. □

Lemma 2.10 *Let R and S be two relations such that: 1. R is terminating and 2. R BD-commutes over S. For any $x \in X$, if x is strongly normalizing in S, then x is strongly normalizing in $R \cup S$.*

Proof. Let x be in X such that x is strongly normalizing in S. We define the set of S-successors of x as $\mathcal{S}_x^X = \{y \in X / x \xrightarrow{S^*} y\}$, and the order relation $\succ$ over $\mathcal{S}_x^X$ as: for all $y, z \in \mathcal{S}_x^X$, $y \succ z$ if and only if $y \xrightarrow{S^+} z$. Since x is strongly normalizing in S, the order $\succ$ is a well-founded relation. We prove by noetherian induction over $\succ$ that for any $y \in \mathcal{S}_x^X$, y is strongly normalizing in $R \cup S$.

Assume that there is a reduction $y \xrightarrow{(R \cup S)^*} \ldots$ If there are no S-steps in the reduction, then it has the form $y \xrightarrow{R^*} \ldots$ This reduction is terminating by hypothesis 1. Otherwise, there is at least one S-step and we can write the reduction as $y \xrightarrow{R^*} \xrightarrow{S} z \xrightarrow{(R \cup S)^*} \ldots$ In this case we use Lemma 2.9 to show that there exists a reduction $y \xrightarrow{S} w \xrightarrow{(R \cup S)^*} z \xrightarrow{(R \cup S)^*} \ldots$ By definition, $w \in \mathcal{S}_x^X$ and $y \succ w$, so by induction hypothesis, w is strongly normalizing in $R \cup S$. Therefore, z is strongly normalizing in $R \cup S$ too, and we conclude that the reduction $y \xrightarrow{R^*} \xrightarrow{S} z \xrightarrow{(R \cup S)^*} \ldots$ is finite.

Thus, $x \in \mathcal{S}_x^X$ is strongly normalizing in $R \cup S$. □

Bachmair-Dershowitz's like lemmas are used in [30, 67, 70] to prove termination in calculi of explicit substitutions.

The Critical Pair lemma [48] says that a rewrite system is locally confluent if its critical pairs are joinable. In the general case, local ground confluence cannot be mechanically verified [51]. However, the critical pair lemma holds for ground expressions [51]: a rewrite system is *locally ground confluent* if its critical pairs are *ground joinable*. In addition, the critical pair lemma also holds for many-sorted systems provided that for every rule $l \longrightarrow r$, the sort of l and the sort of r are the same (in this case we say that the rewrite system is sort-compatible) [85].

We merge the above results for local ground confluence and for sort-compatible systems, in a general presentation of the critical pair lemma.

In the following, we consider that R is a sort-compatible rewrite system defined on a set of expressions X. We denote by $sorts(X)$ the set of sorts of X, and we assume that $s \subseteq sorts(X)$.

Definition 2.11 (Ground Expressions on s) *The set of* s-ground expressions *of* X *is the subset of* X *formed by expressions which does not contain variables of sorts in* s.

Definition 2.12 (Local Ground Confluence on s) *We say that* R is locally s-ground confluent *if* R *is locally confluent on* s*-ground expressions.*

Definition 2.13 (Ground Joinability on s) *For* x *and* y s*-ground expressions, we say that* x and y are s-ground joinable in R, *if there exists a* s*-ground expression* z *such that* $x \xrightarrow{R^*} z$ *and* $y \xrightarrow{R^*} z$ *through paths of* s*-ground expressions.*

Lemma 2.14 (General Critical Pair Lemma) *R is locally s-ground confluent if its critical pairs are s-ground joinable.*

Proof. The proof follows straightforwardly the proofs in [85, 51]. Note that if two expressions are joinable on X, then they are in particular s-ground joinable. Hence, it suffices to concentrate on those critical pairs that are not joinable on X. □

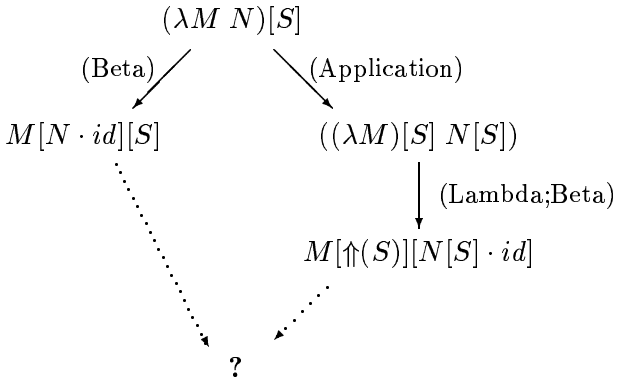
When s is the empty set and $sorts(X)$ is a singleton, the General Critical Pair (GCP) lemma corresponds to the usual critical pair lemma [48]; when s and $sorts(X)$ are both singletons, GCP lemma corresponds to the critical pair lemma on ground expressions [51]; and when s is the empty set and the cardinality of $sorts(X)$ is greater to one, GCP lemma corresponds to the critical pair lemma for sort-compatible systems [85].

2.2 Solving a Conjecture about Confluence and PSN: the λ_ζ-Calculus

It seems that there is a choice: confluence (and only weak normalization) or preservation of strong normalization (and only ground confluence). In this section we propose a calculus of explicit substitutions that enjoys both properties at the same time. It encodes, in a confluent first-order rewrite system, a reduction strategy of substitutions which is proved to preserve strong normalization.

2.2.1 A Harmful Critical Pair

In any calculi having the rules (Beta), (Application) and (Lambda) —e.g. $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$ and λv— we have the following critical pair:



We have seen that in λv the problem does not arise since the above critical pair is joinable on ground terms. In order to achieve local confluence on open expressions, for instance in $\lambda\sigma$ or $\lambda\sigma_{\Uparrow}$, a composition operator has been introduced, allowing to close the critical pair on the term $M[N[S] \cdot S]$. This operator being apparently responsible for non termination, we have taken another approach: we cut one of the branches of the critical pair, i.e. we forbid one of the two reductions.

To cut the right branch we must avoid the distribution of substitutions inside a β-redex. Thus, it is necessary to change the rule (Application) by a rule like (c.f. [44]):

$$((\underline{n}\ M_1) \dots M_m)[S] \quad \longrightarrow \quad ((\underline{n}[S]\ M_1[S]) \dots M_m[S])$$

Indeed, with this rule the distribution of substitutions to applications is only possible if the head term is a variable. Thus, in the above critical pair, the substitution S is not propagated inside $(\lambda M\ N)$ because the header of the term is not a variable. Therefore, only one reduction rule applies and there is no critical pair. It is less clear for us how to cut the left branch of the graph.

$(\lambda M\ N)$	$\longrightarrow$	$M[N/]$	(Beta)
$(\lambda M \bullet N)$	$\longrightarrow$	$M[N/]$	(Beta$^\bullet$)
$(\lambda M)[S]$	$\longrightarrow$	$\lambda M[\Uparrow(S)]$	(Lambda)
$\underline{1}[M/]$	$\longrightarrow$	M	(FVar)
$\underline{n+1}[M/]$	$\longrightarrow$	$\underline{n}$	(RVar)
$\underline{1}[\Uparrow(S)]$	$\longrightarrow$	$\underline{1}$	(FVarLift)
$\underline{n+1}[\Uparrow(S)]$	$\longrightarrow$	$\underline{n}[S][\uparrow]$	(RVarLift)
$\underline{n}[\uparrow]$	$\longrightarrow$	$\underline{n+1}$	(VarShift)
$(M \odot N)[S]$	$\longrightarrow$	$(M[S]\ N[S])$	(UnMark)
$(M\ N)[S]$	$\longrightarrow$	$(M \bullet N)[S]$	(Mark)
$(\underline{n} \bullet M)$	$\longrightarrow$	$(\underline{n} \odot M)$	(ReMark)
$((M\ N) \bullet M')$	$\longrightarrow$	$((M \bullet N) \bullet M')$	(Mark$^\bullet$)
$((M \odot N) \bullet M')$	$\longrightarrow$	$((M \odot N) \odot M')$	(Mark$^\odot$)

Figure 2.5: The rewrite system λ_ζ

The above rule is not a first-order rule. In order to have the header of applications at the top-level of expressions, we must change the structure of applications from $((M\ M_1)\dots M_n)$ to $(M\ (M_1,\dots,M_n))$ where $(M_1,\dots,M_n)$ is a list of terms. This notation for applications is used in the $\bar{\lambda}\sigma$-calculus proposed by Herbelin in [43]. In this calculus, the rule (Application) takes the form:

$$(\underline{n}\ L)[S] \quad \longrightarrow \quad (\underline{n}[S]\ L[S]) \quad \text{(Application-List)}$$

where L is a list of terms.

2.2.2 The λ_ζ-Calculus

The rule (Application-List) avoids the harmful critical pair by cutting the right branch of the graph. We propose a first-order rewrite system which implements an (Application-List)'s like rule by using *the application operator of λ-calculus*. This calculus is called λ_ζ [67].

The λ_ζ-calculus is similar to λv: it does not have composition of substitutions, it uses only singleton substitutions and it preserves strong normalization. Moreover, in contrast to λv, the λ_ζ-calculus is fully confluent (not only on ground expressions).

The set of well-formed expressions in λ_ζ is defined by the following grammar:

Naturals	n	::=	$1 \mid n+1$
Terms	M	::=	$\underline{n} \mid \lambda M \mid (M\ M') \mid M[S] \mid (M \bullet M') \mid (M \odot M')$
Substitutions	S	::=	$M/ \mid\ \uparrow \mid \Uparrow(S)$

The λ_ζ-calculus is given by the rewrite system of Fig. 2.5.

With respect to the syntax of λv, we have added two marked applications (or simply *marks*): $\bullet$ and $\odot$. We use these operators to introduce substitutions inside applications whit head variables. Notice that the rule (Application) of λv becomes (Mark) in λ_ζ.

In a term having the form $(M\ N)[S]$, we can see the use of marks as a hand-shaking process between S and the head term of M. To know if S can be distributed inside $(M\ N)$, a request —$\bullet$-mark— is sent to the head term of M via (Mark). The $\bullet$-mark is propagated recursively into the left part of the application using (Mark$^\bullet$). If the head term of M is a variable, then a positive answer —$\odot$-mark— is sent via (ReMark). The $\odot$-mark is propagated at the top level of the application via (Mark$^\odot$). The hand-shaking is completed when substitution S goes through the $\odot$-marks erasing them, by means of (UnMark). If M has an abstraction as the head term, then it is erased by (Beta$^\bullet$). Other rules of λ_ζ are used to reduce substitutions.

Example 2.15 *The following examples show the process of propagation of marks and substitutions in* λ_ζ.

1. $((\underline{n}\ M_1)\ M_2)[S] \xrightarrow{\lambda_\zeta{}^*} ((\underline{n} \bullet M_1) \bullet M_2)[S] \xrightarrow{\lambda_\zeta{}^*} ((\underline{n} \odot M_1) \odot M_2)[S] \xrightarrow{\lambda_\zeta{}^*}$
   $((\underline{n}[S]\ M_1[S])\ M_2[S]) \xrightarrow{\lambda_\zeta{}^*} \ldots.$

2. $((\lambda M\ M_1)\ M_2)[S] \xrightarrow{\lambda_\zeta{}^*} ((\lambda M \bullet M_1) \bullet M_2)[S] \xrightarrow{\lambda_\zeta{}^*} (M[M_1/] \bullet M_2)[S] \xrightarrow{\lambda_\zeta{}^*} \ldots.$

In order to formulate the main properties of λ_ζ, we need the following definitions and remarks.

Definition 2.16

1. *We call* $B^\bullet$ *the rewrite system formed by the rules* (Beta) *and* (Beta$^\bullet$), *and* ζ *the rewrite system formed by the rest of rules.*

2. *A term in the* λ_ζ*-calculus is* pure *if it is ground and it does not contain marks or substitutions.*

3. *A term* N *is* purifiable *if and only there exists a pure term* M *such that* $M \xrightarrow{\lambda_\zeta{}^*} N$.

Remark 2.17

1. *The set of* λ_{dB}*-terms is the set of pure terms.*

2. *The set of* β*-normal forms is the set of pure* λ_ζ*-normal forms.*

Note that the set of λ_ζ-normal forms is not included in the set of pure terms. For example, the λ_ζ-normal form $(\underline{1} \odot \underline{1})$ is not pure. However, we show in Lemma 2.20 that marks disappear in λ_ζ-normal forms of pure terms.

One-step semantics of β-reduction, i.e. simulation step-by-step of β-reductions, are not strictly implemented in λ_ζ. For instance, the reduction $\overbrace{(\lambda(\lambda\underline{1}\ \underline{2})\ \underline{1})}^{\beta\text{-redex}} \xrightarrow{\beta} (\lambda\underline{1}\ \underline{1})$ is not possible in λ_ζ, because $\overbrace{(\lambda(\lambda\underline{1}\ \underline{2})\ \underline{1})}^{\text{(Beta)-redex}}$ λ_ζ-reduces to $(\lambda\underline{1}\ \underline{2})[\underline{1}/]$, and we cannot perform the substitution until the redex $(\lambda\underline{1}\ \underline{2})$ has not been reduced. Thus, one-step β-reduction is not preserved unless we consider $(\lambda\underline{1}\ \underline{2})[\underline{1}/]$ as

$[\![\underline{n}]\!]_1$	$=$	2^n	$[\![\underline{n}]\!]_2$	$=$	n
$[\![\lambda M]\!]_1$	$=$	$1 + [\![M]\!]_1$	$[\![\lambda M]\!]_2$	$=$	1
$[\![(M\ M')]\!]_1$	$=$	$1 + [\![M]\!]_1 + [\![M']\!]_1$	$[\![(M\ M')]\!]_2$	$=$	$3 + [\![M]\!]_2 + [\![M']\!]_2$
$[\![(M \bullet M')]\!]_1$	$=$	$1 + [\![M]\!]_1 + [\![M']\!]_1$	$[\![(M \bullet M')]\!]_2$	$=$	$2 + [\![M]\!]_2 + [\![M']\!]_2$
$[\![(M \odot M')]\!]_1$	$=$	$1 + [\![M]\!]_1 + [\![M']\!]_1$	$[\![(M \odot M')]\!]_2$	$=$	$1 + [\![M]\!]_2 + [\![M']\!]_2$
$[\![M[S]]\!]_1$	$=$	$[\![M]\!]_1 [\![S]\!]_1$	$[\![M[S]]\!]_2$	$=$	$[\![M]\!]_2 + [\![S]\!]_2$
$[\![M/]\!]_1$	$=$	$[\![M]\!]_1$	$[\![M/]\!]_2$	$=$	1
$[\![\uparrow]\!]_1$	$=$	2	$[\![\uparrow]\!]_2$	$=$	2
$[\![\Uparrow(S)]\!]_1$	$=$	$[\![S]\!]_1$	$[\![\Uparrow(S)]\!]_2$	$=$	$2 + [\![S]\!]_2$

Figure 2.6: Interpretation for proving the termination of ζ

equivalent —modulo substitutions— to $(\lambda\underline{1}\ \underline{1})$. Nevertheless, in both calculi the following normal reduction is possible:

$$(\lambda \overbrace{(\lambda\underline{1}\ \underline{2})}^{\beta\text{-redex}}\ \underline{1}) \xrightarrow{*} \overbrace{(\lambda\underline{2}\ \underline{1})}^{\beta\text{-redex}} \xrightarrow{*} \underline{1}$$

This semantic of β-reduction is called big-step.

As far as we know, λ_ζ is the first calculus that enjoys both properties of confluence and preservation of strong normalization.

2.2.3 Properties

Now, we prove the main properties of the λ_ζ-calculus (for detailed proofs see [66]).

Basic Properties

Lemma 2.18 *The rewrite system ζ is terminating.*

Proof. We verify easily that the interpretation given in Fig. 2.6 defines a reduction order for ζ. □

Lemma 2.19 *The rewrite system ζ is confluent.*

Proof. We verify mechanically that ζ is locally confluent, for example using the system RRL [52]. We conclude with Lemma 2.18 and the Newman's lemma that that ζ is confluent. □

Lemma 2.20 *The λ_ζ-normal forms of purifiable terms are pure.*

Proof. We show that the set of purifiable terms can be described by the following grammar:

Purifiable terms	P	$::=$	$\underline{n} \mid \lambda P \mid (P\ P') \mid \overset{\bullet}{P}[S]$
Purifiable$^\bullet$ terms	$\overset{\bullet}{P}$	$::=$	$P \mid \overset{\odot}{P} \mid (\overset{\bullet}{P} \bullet P)$
Purifiable$^\odot$ terms	$\overset{\odot}{P}$	$::=$	$\underline{n} \mid (\overset{\odot}{P} \odot P)$
Purifiable substitutions	S	$::=$	$P/ \mid \uparrow \mid \Uparrow(S)$

and also, we check that this set is closed under λ_ζ-reductions.

Finally, we prove that λ_ζ-normal forms of ground term do not contain substitutions, thus λ_ζ-normal forms of purifiable terms are pure. □

Reduction Soundness

We define a function μ which maps ground λ_ζ-terms into λ_{dB}-terms. This function is strongly inspired from that defined in [81, 6].

$$\begin{array}{lcl}
\mu(M \bullet N) & = & (\mu(M)\ \mu(N)) \\
\mu(M \odot N) & = & (\mu(M)\ \mu(N)) \\
\mu(M\ N) & = & (\mu(M)\ \mu(N)) \\
\mu(\underline{n}) & = & \underline{n} \\
\mu(\lambda M) & = & \lambda\mu(M) \\
\mu(M[\Uparrow^n(N/)]) & = & \mu(M)\{n{+}1 := \mu(N)\} \\
\mu(M[\Uparrow^n(\uparrow)]) & = & \tau_n^1(\mu(M))
\end{array}$$

where $\Uparrow^n(S)$ is a notation for $\overbrace{\Uparrow(\ldots(\Uparrow(S)))}^{n\text{-times}}$, and $M\{n := N\}$, $\tau_n^1(.)$ as defined in Def. 2.1.

Remark 2.21 *If M is a pure term, then $\mu(M) = M$.*

Lemma 2.22 *Let M and N be ground λ_ζ-terms,*

1. *if $M \xrightarrow{B^\bullet} N$, then $\mu(M) \xrightarrow{\beta^*} \mu(N)$, and*
2. *if $M \xrightarrow{\zeta^*} N$, then $\mu(M) = \mu(N)$.*

Proof. We reason by case analysis on the λ_ζ-redex reduced in M. □

We have the following consequence of Lemma 2.22 and Def. 2.16.

Corollary 2.23 *If $M \xrightarrow{\lambda_\zeta{}^*} N$, then $\mu(M) \xrightarrow{\beta^*} \mu(N)$.*

Theorem 2.24 (Reduction Soundness) *If M and N are pure terms, and $M \xrightarrow{\lambda_\zeta{}^*} N$, then $M \xrightarrow{\beta^*} N$.*

Proof. If M and N are pure terms, then by Remark 2.21, $M = \mu(M)$ and $N = \mu(N)$. Finally, by Corollary 2.23 applied to hypothesis $M \xrightarrow{\lambda_\zeta{}^*} N$, we conclude $M = \mu(M) \xrightarrow{\beta^*} \mu(N) = N$. □

Reduction Completeness

We define reduction strategies in λ_ζ and λ_{dB}.

Definition 2.25 (Leftmost-Outermost Strategies)

1. *Let M be a λ_{dB}-term, we say that M reduces to N by the* leftmost-outermost strategy*, denoted by $M \xrightarrow[\mathrm{LO}]{\beta} N$, if and only if $M \xrightarrow{\beta} N$ using the leftmost-outermost β-redex of M.*

2. *Let M be a ground λ_ζ-term, we say that M reduces to N by a* leftmost-outermost λ_ζ-strategy*, denoted by $M \xrightarrow[\mathrm{LO}]{\lambda_\zeta} N$, if and only if $M \xrightarrow{B^\bullet} M'$ using the leftmost-outermost $B^\bullet$-redex of M, and N is the ζ-normal form of M'.*

The following lemma establishes the relation between leftmost-outermost β and λ_ζ-strategies.

Lemma 2.26 *Let M be a ζ-normal form of a purifiable term, if $M \xrightarrow[\mathrm{LO}]{\lambda_\zeta} N$ then $\mu(M) \xrightarrow[\mathrm{LO}]{\beta} \mu(N)$.*

Proof. By definition of leftmost-outermost λ_ζ-strategy, $M \xrightarrow{B^\bullet} M' \xrightarrow{\zeta^*} N$, where M reduces its leftmost-outermost $B^\bullet$-redex, and N is a ζ-normal form. By structural induction on M we show that $\mu(M) \xrightarrow{\beta} \mu(M')$, but also by Lemma 2.22, $\mu(M') = \mu(N)$. Therefore, $\mu(M) \xrightarrow[\mathrm{LO}]{\beta} \mu(M') = \mu(N)$. □

The following is a well-known result about the leftmost-outermost strategy in λ-calculus.

Lemma 2.27 *The leftmost-outermost strategy is a normal strategy, i.e. if M has β-normal form N, then $M \xrightarrow[\mathrm{LO}]{\beta^*} N$.*

Proof. See [4]. □

Theorem 2.28 (Reduction Completeness) *If M is a pure term, N is a β-normal form, and $M \xrightarrow{\beta^*} N$, then $M \xrightarrow{{\lambda_\zeta}^*} N$.*

Proof. Let M be a β-normalisable term. Assume that there exists an infinite reduction $M = M_0 \xrightarrow[\mathrm{LO}]{\lambda_\zeta} M_1 \xrightarrow[\mathrm{LO}]{\lambda_\zeta} \ldots$ Since M is a pure term, all M_i, $i \geq 0$, are ζ-normal forms of purifiable terms (Def. 2.16(3), Def. 2.25). By Lemma 2.26, $\mu(M_0) \xrightarrow[\mathrm{LO}]{\beta} \mu(M_1) \xrightarrow[\mathrm{LO}]{\beta} \ldots$ is an infinite reduction too. But $M = \mu(M_0)$ (Remark 2.21), so there is an infinite leftmost-outermost β-reduction of M and this is contradictory with Lemma 2.27.

Therefore, there is a reduction $M \xrightarrow[\mathrm{LO}]{{\lambda_\zeta}^*} N'$ where N' is a λ_ζ-normal form. By Lemma 2.20, N' is pure, thus, by Theorem 2.24, $M \xrightarrow{\beta^*} N'$. But N' is a β-normal form (Remark 2.17(2)) and λ_{dB} is confluent (c.f. [21, 4]), so $N = N'$. □

Confluence

The parallelization of $B^\bullet$, namely $B^\bullet_{\|}$, is defined as follows:

$$\frac{}{x \longrightarrow x}\,(\text{Refl}_{\|}) \qquad \frac{M \longrightarrow M'}{\lambda M \longrightarrow \lambda M'}\,(\text{Lambda}_{\|})$$

$$\frac{M \longrightarrow M' \qquad S \longrightarrow S'}{M[S] \longrightarrow M'[S']}\,(\text{Subs}_{\|}) \qquad \frac{S \longrightarrow S'}{\Uparrow(S) \longrightarrow \Uparrow(S')}\,(\text{Lift}_{\|})$$

$$\frac{M \longrightarrow M' \qquad N \longrightarrow N'}{(M\ N) \longrightarrow (M'\ N')}\,(\text{Application}_{\|}) \qquad \frac{M \longrightarrow M' \qquad N \longrightarrow N'}{(M \bullet N) \longrightarrow (M' \bullet N')}\,(\text{Application}^\bullet_{\|})$$

$$\frac{M \longrightarrow M' \qquad N \longrightarrow N'}{(M \odot N) \longrightarrow (M' \odot N')}\,(\text{Application}^\odot_{\|}) \qquad \frac{M \longrightarrow M'}{M/ \longrightarrow M'/}\,(\text{Cons}_{\|})$$

$$\frac{M \longrightarrow M' \qquad N \longrightarrow N'}{(\lambda M\ N) \longrightarrow M'[N'/]}\,(\text{Beta}_{\|}) \qquad \frac{M \longrightarrow M' \qquad N \longrightarrow N'}{(\lambda M \bullet N) \longrightarrow M'[N'/]}\,(\text{Beta}^\bullet_{\|})$$

Lemma 2.29 *The ζ-calculus YH-commutes over $B^\bullet_{\|}$, i.e. if x reduces in one ζ-step to y, and in one $B^\bullet_{\|}$-step to z, then there exists w such that $y \xrightarrow{\zeta^* B^\bullet_{\|} \zeta^*} w$ and $z \xrightarrow{\zeta^*} w$.*

Proof. By induction on the depth of the ζ-redex reduced in x. At the base case x is a ζ-redex:

- $x = (M\ N)[S] \xrightarrow{(\text{Mark})} (M \bullet N)[S] = y$ and $(M\ N)[S] \xrightarrow{B^\bullet_{\|}} (M'\ N')[S'] = z$, with $M \xrightarrow{B^\bullet_{\|}} M'$, $N \xrightarrow{B^\bullet_{\|}} N'$ and $S \xrightarrow{B^\bullet_{\|}} S'$. By definition of $B^\bullet_{\|}$, $(M \bullet N)[S] \xrightarrow{B^\bullet_{\|}} (M' \bullet N')[S']$. But also, $(M'\ N')[S'] \xrightarrow{(\text{Mark})} (M' \bullet N')[S']$.
- $x = (\lambda M\ N)[S] \xrightarrow{(\text{Mark})} (\lambda M \bullet N)[S]$ and $(\lambda M\ N)[S] \xrightarrow{B^\bullet_{\|}} M'[N'/][S'] = z$, with $M \xrightarrow{B^\bullet_{\|}} M'$, $N \xrightarrow{B^\bullet_{\|}} N'$ and $S \xrightarrow{B^\bullet_{\|}} S'$. But also, $(\lambda M \bullet N)[S] \xrightarrow{(\text{Beta}^\bullet_{\|})} M'[N'/][S']$.
- The other cases are similar to the previous ones.

□

Theorem 2.30 (Confluence) *The λ_ζ-calculus is confluent.*

Proof. The proof proceeds by taking X as the set of λ_ζ-terms, S as $B^\bullet_{\|}$ and R as the rewrite system ζ. We verify the hypotheses of Lemma 2.6, i.e.

1. ζ is terminating and confluent (Lemma 2.18, Lemma 2.19).
2. $B^\bullet_{\|}$ is strongly confluent, since $B^\bullet$ is a left-linear rewrite system without critical pairs (c.f. [48]).
3. ζ YH-commutes over $B^\bullet_{\|}$ (Lemma 2.29).

Therefore, $\zeta^* B^\bullet_{\|} \zeta^*$ is confluent. Note that $\lambda_\zeta \subseteq \zeta^* B^\bullet_{\|} \zeta^* \subseteq {\lambda_\zeta}^*$. Hence, if $x \xrightarrow{{\lambda_\zeta}^*} y$ and $x \xrightarrow{{\lambda_\zeta}^*} z$, then there exists w such that $y \xrightarrow{(\zeta^* B^\bullet_{\|} \zeta^*)^*} w$ and $z \xrightarrow{(\zeta^* B^\bullet_{\|} \zeta^*)^*} w$. So, $y \xrightarrow{{\lambda_\zeta}^*} w$ and $z \xrightarrow{{\lambda_\zeta}^*} w$. □

Preservation of Strong Normalization

We define the two calculi $\mathcal{M}$ and $\lambda v^\bullet$ as follows:

- the rewrite system $\mathcal{M}$ is formed by the rules (Mark), (ReMark), (Mark$^\bullet$) and (Mark$^\odot$),
- the rewrite system $\lambda v^\bullet$ is formed by the rules of λv in addition with the rules (Beta$^\bullet$), (UnMark) and

$$(M \bullet M')[S] \quad \longrightarrow \quad (M[S]\ M'[S]) \quad \text{(UnMark')}$$

Lemma 2.31 *The rewrite system $\lambda v^\bullet$ preserves strong normalization.*

Proof. We define a function which maps $\lambda v^\bullet$-terms into λv-terms:

$$\begin{array}{lcl}
\overline{\underline{n}} & = & \underline{n} \\
\overline{\lambda M} & = & \lambda\overline{M} \\
\overline{(M\ M')} & = & (\overline{M}\ \overline{M'}) \\
\overline{M[S]} & = & \overline{M}[\overline{S}] \\
\overline{(M \bullet M')} & = & (\overline{M}\ \overline{M'}) \\
\overline{(M \odot M')} & = & (\overline{M}\ \overline{M'}) \\
\overline{\uparrow} & = & \uparrow \\
\overline{\Uparrow(S)} & = & \Uparrow(\overline{S}) \\
\overline{M/} & = & \overline{M}/
\end{array}$$

It is easy to see that the map is compatible with reductions, i.e. if $M \xrightarrow{\lambda v^\bullet} N$ then $\overline{M} \xrightarrow{\lambda v} \overline{N}$. But, λv preserves strong normalization (Proposition 2.4), so we conclude that $\lambda v^\bullet$ preserves strong normalization too. □

Lemma 2.32 *The $\mathcal{M}$-calculus BD-commutes over $\lambda v^\bullet$.*

Proof. By case analysis in $\mathcal{M}$ and $\lambda v^\bullet$-redexes. □

Theorem 2.33 (Preservation of Strong Normalization) *If M is strongly normalizing in λ_{dB}, then M is strongly normalizing in λ_ζ.*

Proof. We take X as the set of ground λ_ζ-terms, S as $\lambda v^\bullet$ and R as $\mathcal{M}$. We verify the hypotheses of Lemma 2.8, i.e.:

1. $\mathcal{M}$ is terminating. Notice that $\mathcal{M} \in \zeta$, and by Lemma 2.18, ζ is terminating.
2. $\lambda v^\bullet$ preserves strong normalization (Lemma 2.31).
3. $\mathcal{M}$ BD-commutes over $\lambda v^\bullet$ (Lemma 2.32).

Therefore, $(\mathcal{M} \cup \lambda v^\bullet)$ preserves strong normalization. Since $\lambda_\zeta \subseteq (\mathcal{M} \cup \lambda v^\bullet)$, we conclude that λ_ζ preserves strong normalization. □

2.2.4 An Open Problem

The λ_ζ-calculus solves positively the conjecture about the existence of a first-order calculus of explicit substitutions which enjoys both confluence and preservation of strong normalization properties.

The main motivation of this thesis is to propose a formal model to represent incomplete proofs in proof checkers based on type theory. But, is the λ_ζ-calculus suitable to meet this goal? The fact that one-step β-reduction is not simulated in λ_ζ, seems to be a minor drawback. Big-step semantics of β-reduction suffice to reason about proof-terms.

In the λ_ζ-calculus, the confluence property holds without the introduction of composition of substitutions. The composition operation introduces simultaneous substitutions that happens to be useful for several purposes. For example, the modeling of closures of an abstract machine [40] or the pruning of search space in unification algorithms [25, 26, 55]. Also, this feature improves the substitution mechanism by allowing parallel substitutions of variables. An interesting discussion about composition of substitutions in λ-calculus can be found in [73].

Example 2.34 *In the unification algorithm, the equation $X[N/][S] =_? X[\Uparrow(S)][N[S]/]$, where X is a meta-variable, N is a ground term and S is a ground substitution, is a trivial equation when we add the composition operation: $X[N[S].S] =_? X[N[S].S]$. In λ_ζ (and any calculi without composition), it is only a flexible-flexible equation which is trivial for any ground instance of X.*

The re-introduction of composition and simultaneous substitutions in λ_ζ, while preserving its properties, is not simple. In fact, if we add the rule $M[S][T] \longrightarrow M[S \circ T]$, we have a critical pair with (UnMark) that leads to the rule $(M[S] \bullet N[S])[T] \longrightarrow (M[S \circ T]\ N[S \circ T])$. Now, it comes a harmful critical pair when we take M as $\lambda M'$. Nevertheless, we think that a constraint composition operation, just as composition of λ_d [30], is even possible.

Actually, we do not use the λ_ζ-calculus to represent incomplete proof-terms. Proof synthesis methods for higher-order logics are based on unification where composition and simultaneous substitutions are very useful features. However, we think that λ_ζ is a first step toward a calculus of practical interest. In this way, we state the following problem:

Problem 2.35 *Does there exist a calculus of explicit substitutions, confluent (on open terms), that preserves strong normalization, and that accepts general composition of substitutions (in particular, simultaneous substitutions) ?*

2.3 A Left-Linear Variant of $\lambda\sigma$: the $\lambda_\mathcal{L}$-Calculus

In practice, confluence on semi-open expressions, weak normalization and composition of substitutions, are sufficient features for many applications of λ-calculi with explicit substitutions and meta-variables. The $\lambda\sigma$-calculus has the above features. However, the non-left-linear rule of $\lambda\sigma$ (see Section 2.1.3) —$\mathbf{1}[S] \cdot (\uparrow \circ S) \xrightarrow{\text{(SCons)}} S$— is impractical for many reasons. We will show in Chapter 3 (c.f. [69]) that $\lambda\sigma$ with dependent types may lose the subject reduction property due to (SCons). Independently, Nadathur [74] has remarked that this non-left-linear rule is difficult to handle in implementations. In fact, he shows that (SCons) is admissible on semi-open expressions

$(\lambda M\ N)$	$\longrightarrow$	$M[N \cdot \uparrow^0]$	(Beta)
$(\lambda M)[S]$	$\longrightarrow$	$\lambda M[\mathbf{1} \cdot (S \circ \uparrow^{Suc(0)})]$	(Lambda)
$(M\ N)[S]$	$\longrightarrow$	$(M[S]\ N[S])$	(Application)
$M[S][T]$	$\longrightarrow$	$M[S \circ T]$	(Clos)
$\mathbf{1}[M \cdot S]$	$\longrightarrow$	M	(VarCons)
$M[\uparrow^0]$	$\longrightarrow$	M	(Id)
$(M \cdot S) \circ T$	$\longrightarrow$	$M[T] \cdot (S \circ T)$	(Map)
$\uparrow^0 \circ S$	$\longrightarrow$	S	(IdS)
$\uparrow^{Suc(n)} \circ (M \cdot S)$	$\longrightarrow$	$\uparrow^n \circ S$	(ShiftCons)
$\uparrow^{Suc(n)} \circ \uparrow^m$	$\longrightarrow$	$\uparrow^n \circ \uparrow^{Suc(m)}$	(ShiftShift)
$\mathbf{1} \cdot \uparrow^{Suc(0)}$	$\longrightarrow$	$\uparrow^0$	(Shift0)
$\mathbf{1}[\uparrow^n] \cdot \uparrow^{Suc(n)}$	$\longrightarrow$	$\uparrow^n$	(ShiftS)

Figure 2.7: The rewrite system $\lambda_{\mathcal{L}}$

when the following scheme of rule is considered: $\mathbf{1}[\uparrow^n] \cdot \uparrow^{n+1} \xrightarrow{\text{(SCons)}} \uparrow^n$, where $\uparrow^n$ is a notation for "$\overbrace{(\uparrow \circ (\ldots \circ \uparrow))}^{n\text{-times}}$".

To have a finitary presentation of the rule suggested by Nadathur, we introduce a primitive operator $\uparrow^n$. The obtained calculus, namely $\lambda_{\mathcal{L}}$, enjoys the same general features as $\lambda\sigma$, i.e. a *simple* and *finitary first-order presentation*, with *general composition of substitutions*, *confluent on semi-open expressions* and *weakly normalizing on typed terms*. However, in contrast to $\lambda\sigma$, the new calculus is left-linear, and in particular, it does not have the rule (SCons) of $\lambda\sigma$.

2.3.1 The $\lambda_{\mathcal{L}}$-Calculus

The set of well-formed expressions in $\lambda_{\mathcal{L}}$ is defined by the following grammar:[1]

Naturals	n	::=	$0 \mid Suc(n)$
Terms	M, N	::=	$\mathbf{1} \mid \lambda M \mid (M\ N) \mid M[S]$
Substitutions	S, T	::=	$\uparrow^n \mid M \cdot S \mid S \circ T$

The $\lambda_{\mathcal{L}}$-calculus is given by the rewrite system of Fig. 2.7. The $\mathcal{L}$-calculus is obtained by dropping (Beta) from $\lambda_{\mathcal{L}}$.

A first remark is that $\lambda_{\mathcal{L}}$ gathers, in its syntax, some notations that are frequently used to speak informally of $\lambda\sigma$. For example, the substitutions id and $\overbrace{(\uparrow \circ (\ldots \circ \uparrow))}^{n+1}$ of $\lambda\sigma$ are written respectively $\uparrow^0$ and $\uparrow^{Suc(n)}$ in $\lambda_{\mathcal{L}}$. Similarly, the de Bruijn's indices $\underline{1}$ and $\underline{n+1}$ are represented respectively by

[1] In previous manuscripts ([71, 69]) the name of the calculus was λ_ϕ, but we have changed to $\lambda_{\mathcal{L}}$ in order to avoid confusion with the $\lambda\phi$-calculus proposed by Lescanne in [58].

$\mathbf{1}$ and $\mathbf{1}[\uparrow^n]$ in $\lambda_{\mathcal{L}}$. In the $\lambda_{\mathcal{L}}$-calculus, the scheme of rule suggested by Nadathur is written as a first-order and finitary rule.

The $\lambda_{\mathcal{L}}$-calculus is not confluent, not even locally confluent, on general open expressions. However, just as $\lambda\sigma$, it is confluent on semi-open expressions, i.e. expressions with meta-variables of terms, but no meta-variables of substitutions or naturals.

The non-linearity of $\lambda_{\mathcal{L}}$ due to (ShiftS) is only apparent since the term with a double occurrence in this rule can be considered as a constant in the set of semi-open expressions. In particular, there are no reduction rules for natural numbers.

The $\lambda_{\mathcal{L}}$-calculus does not preserve strong normalization. In fact, Melliès' counter-example for $\lambda\sigma$ [64, 63] can be reproduced in $\lambda_{\mathcal{L}}$. Nevertheless, we show in Chapter 7 that, just like $\lambda\sigma$, $\lambda_{\mathcal{L}}$ is weakly normalizing on typed expressions.

For users of $\lambda\sigma$, it may seem odd that the rules (Ass) and (IdR)

$$\begin{array}{lcll} (S \circ T) \circ T' & \longrightarrow & S \circ (T \circ T') & \text{(Ass)} \\ S \circ \uparrow^0 & \longrightarrow & S & \text{(IdR)} \end{array}$$

do not appear in $\lambda_{\mathcal{L}}$. In fact as we show in next section, these rules are not necessary to retain the confluence property in semi-open expressions. However, the rules can be added to the system without destroying any property. These rules are interesting in implementations for efficiency reasons.

2.3.2 Properties

We start with some basic properties of $\lambda_{\mathcal{L}}$ and $\mathcal{L}$.

Lemma 2.36 *The $\mathcal{L}$-calculus is terminating.*

Proof. The proof we present here is entirely due to H. Zantema [93]. An alternative proof based on the normalization of $\lambda\sigma$ is proposed in [71].

The proof uses the *Semantic Labeling* method introduced in [92]. The idea is to transform the original system into a labeled, possibly infinite, system. The transformation has the property that the labeled system is terminating if and only if the original one is terminating.

The first step of the proof consists to identify the sorts of terms, substitutions and naturals in $\lambda_{\mathcal{L}}$. We obtain the one-sorted rewrite system

$$\begin{array}{l} (\lambda M) \circ S \longrightarrow \lambda(M \circ (Suc(0) \cdot (S \circ Suc(0)))) \\ (M\ N) \circ S \longrightarrow (M \circ S\ N \circ S) \\ (M \circ S) \circ T \longrightarrow M \circ (S \circ T) \\ Suc(0) \circ (M \cdot S) \longrightarrow M \\ (M \cdot S) \circ T \longrightarrow (M \circ T) \cdot (S \circ T) \\ M \circ 0 \longrightarrow M \\ 0 \circ S \longrightarrow S \\ Suc(n) \circ (M \cdot S) \longrightarrow n \circ S \\ Suc(n) \circ m \longrightarrow n \circ Suc(m) \\ Suc(0) \cdot Suc(0) \longrightarrow 0 \\ (Suc(0) \circ n) \cdot Suc(n) \longrightarrow n \end{array}$$

Clearly, termination of the one-sorted system implies termination of $\mathcal{L}$. Now we define an interpretation in some ordered set such that for each rewriting rule the interpretation of the left hand side is greater or equal to the right hand side. This interpretation is called a *quasi-model*, and the labeling transformation will depends on it. For $\mathcal{L}$ the quasi-model is defined on the positive natural numbers as follows:

$$\begin{array}{lcl} [\![0]\!] & = & 1 \\ [\![Suc(n)]\!] & = & [\![n]\!] \\ [\![\lambda M]\!] & = & [\![M]\!] \end{array}$$

$$\begin{array}{lcl} [\![(M\ N)]\!] & = & [\![M]\!] + [\![N]\!] \\ [\![S \circ T]\!] & = & [\![S]\!] \times [\![T]\!] \\ [\![M \cdot S]\!] & = & \max([\![M]\!], [\![S]\!]) \end{array}$$

We verify that it is a quasi-model:

- $[\![(\lambda M) \circ S]\!] = [\![M]\!] \times [\![S]\!] = [\![M]\!] \times \max(1, [\![S]\!] \times 1) = [\![\lambda(M \circ (Suc(0) \cdot (S \circ Suc(0))))]\!]$.
- $[\![(M\ N) \circ S]\!] = ([\![M]\!] + [\![N]\!]) \times [\![S]\!] = ([\![M]\!] \times [\![S]\!]) + ([\![N]\!] \times [\![S]\!]) = [\![(M \circ S\ N \circ S)]\!]$.
- $[\![(M \circ S) \circ T]\!] = [\![M]\!] \times [\![S]\!] \times [\![T]\!] = [\![M \circ (S \circ T)]\!]$.
- $[\![Suc(0) \circ (M \cdot S)]\!] = 1 \times \max([\![M]\!], [\![S]\!]) \geq [\![M]\!]$.
- $[\![(M \cdot S) \circ T]\!] = \max([\![M]\!], [\![S]\!]) \times [\![T]\!] = \max([\![M]\!] \times [\![T]\!], [\![S]\!] \times [\![T]\!]) = [\![(M \circ T) \cdot (S \circ T)]\!]$.
- $[\![M \circ 0]\!] = [\![M]\!] \times 1 = [\![M]\!]$.
- $[\![0 \circ S]\!] = 1 \times [\![S]\!] = [\![S]\!]$.
- $[\![Suc(n) \circ (M \cdot S)]\!] = [\![n]\!] \times \max([\![M]\!], [\![S]\!]) \geq [\![n]\!] \times [\![S]\!] = [\![n \circ S]\!]$.
- $[\![Suc(n) \circ m]\!] = [\![n]\!] \times [\![m]\!] = [\![n \circ Suc(m)]\!]$.
- $[\![Suc(0) \cdot Suc(0)]\!] = \max(1, 1) = 1 = [\![0]\!]$.
- $[\![(Suc(0) \circ n) \cdot Suc(n)]\!] = \max(1 \times [\![n]\!], [\![n]\!]) = [\![n]\!]$.

Only compositions are labeled: the expression $S \circ T$ becomes $S \circ_{([\![S]\!] \times [\![T]\!])} T$. According to Zantema's Theorem [92], it suffices to prove termination of the following infinite rewrite system:

$$\begin{array}{l} (\lambda M) \circ_{x \times y} S \longrightarrow \lambda(M \circ_{x \times y} (Suc(0) \cdot (S \circ_y Suc(0)))) \\ (M\ N) \circ_{(x+y) \times z} S \longrightarrow (M \circ_{x \times z} S\ N \circ_{y \times z} S) \\ (M \circ_{x \times y} S) \circ_{x \times y \times z} T \longrightarrow M \circ_{x \times y \times z} (S \circ_{y \times z} T) \\ Suc(0) \circ_{\max(x,y)} (M \cdot S) \longrightarrow M \\ (M \cdot S) \circ_{\max(x,y) \times z} T \longrightarrow (M \circ_{x \times z} T) \cdot (S \circ_{y \times z} T) \\ M \circ_x 0 \longrightarrow M \\ 0 \circ_x S \longrightarrow S \\ Suc(n) \circ_{z \times \max(x,y)} (M \cdot S) \longrightarrow n \circ_{z \times y} S \\ Suc(n) \circ_{x \times y} m \longrightarrow n \circ_{x \times y} Suc(m) \\ Suc(0) \cdot Suc(0) \longrightarrow 0 \\ (Suc(0) \circ_x n) \cdot Suc(n) \longrightarrow n \\ S \circ_i T \longrightarrow S \circ_j T \end{array}$$

for all strict positive naturals x, y, z, i and j, with $i > j$. Note that $(x + y) \times z > x \times z$, $(x+y) \times z > y \times z$, $(x+y) \times z > z$, $x \times y \times z \geq y \times z$, $\max(x, y) \times z \geq x \times z$, and $\max(x, y) \times z \geq y \times z$. Termination of this labeled system is proved by a recursive path order where $o_i > o_j$ for $i > j$, o_i is bigger than any other symbol, and o_i has a left-right lexicographic status.

Hence, $\mathcal{L}$ is terminating. □

The rewrite system $\mathcal{L}$ has the following critical pairs:

- **(Id-Clos)**. $M[S] \xleftarrow{\mathcal{L}^+} M[S][\uparrow^0] \xrightarrow{\mathcal{L}^+} M[S \circ \uparrow^0]$.
- **(Clos-Clos)**. $M[(S \circ T) \circ T'] \xleftarrow{\mathcal{L}^+} M[S][T][T'] \xrightarrow{\mathcal{L}^+} M[S \circ (T \circ T')]$.
- **(Shift0-Map)**. $S \xleftarrow{\mathcal{L}^+} (\mathbf{1} \cdot \uparrow^{Suc(0)}) \circ S \xrightarrow{\mathcal{L}} \mathbf{1}[S] \cdot (\uparrow^{Suc(0)} \circ S)$.
- **(ShiftS-Map)**. $\uparrow^n \circ S \xleftarrow{\mathcal{L}} (\mathbf{1}[\uparrow^n] \cdot \uparrow^{Suc(n)}) \circ S \xrightarrow{\mathcal{L}^+} \mathbf{1}[\uparrow^n \circ S] \cdot (\uparrow^{Suc(n)} \circ S)$.
- **(Lambda-Clos)**. $\lambda M[\mathbf{1} \cdot ((S \circ \uparrow^{Suc(0)}) \circ (\mathbf{1} \cdot (T \circ \uparrow^{Suc(0)})))] \xleftarrow{\mathcal{L}^+} (\lambda M)[S][T] \xrightarrow{\mathcal{L}^+}$ $\lambda M[\mathbf{1} \cdot ((S \circ T) \circ \uparrow^{Suc(0)})]$.

If we consider (Beta), then we have additionally the following critical pair with (Application):

- **(Beta-Application)**. $M[N[S] \cdot S] \xleftarrow{{\lambda_{\mathcal{L}}}^+} (\lambda M\ N)[S] \xrightarrow{{\lambda_{\mathcal{L}}}^+} M[N[S] \cdot ((S \circ \uparrow^{Suc(0)}) \circ (N \cdot \uparrow^0))]$.

The following lemma shows that these critical pairs are joinable on semi-open expressions, i.e. expressions with meta-variables of terms, but no meta-variables of substitutions or naturals.

Lemma 2.37 *The critical pairs of $\lambda_{\mathcal{L}}$ are $\mathcal{L}$-joinable on semi-open expressions.*

Proof. For any critical pair we reduce substitutions to $\mathcal{L}$-normal forms, and finally we proceed by structural induction on $\mathcal{L}$-normal substitutions. □

Lemma 2.38 *The $\mathcal{L}$-calculus is confluent on semi-open expressions.*

Proof. The $\mathcal{L}$-calculus has three sorts of expressions: Naturals, Substitutions and Terms, but only meta-variables of terms are admitted. We verify that $\mathcal{L}$ is a sort compatible system, i.e. terms reduce to terms and substitutions reduce to substitutions. Lemma 2.37 says that critical pairs of $\mathcal{L}$ are joinable on semi-open expressions, thus by Lemma 2.14, $\mathcal{L}$ is locally confluent on semi-open expressions. We conclude with Newman's lemma and Lemma 2.36 that $\mathcal{L}$ is confluent on semi-open expressions. □

Corollary 2.39 *Normal forms (of semi-open expressions) in $\mathcal{L}$ always exist and they are unique. We denote by $(x)\downarrow_{\mathcal{L}}$ the $\mathcal{L}$-normal form of x.*

Now, we address the confluence property of $\lambda_{\mathcal{L}}$.

The $B_{\|}$ rewrite system is defined as the parallelization of (Beta), i.e.

$$\frac{}{x \longrightarrow x}\,(\text{Refl}_{\|}) \qquad \frac{M \longrightarrow N}{\lambda M \longrightarrow \lambda N}\,(\text{Lambda}_{\|})$$

$$\frac{M \longrightarrow M' \qquad N \longrightarrow N'}{(M\ N) \longrightarrow (M'\ N')}\,(\text{Application}_{\|}) \qquad \frac{M \longrightarrow N \qquad S \longrightarrow T}{M[S] \longrightarrow N[T]}\,(\text{Clos}_{\|})$$

$$\frac{M \longrightarrow N \qquad S \longrightarrow T}{M \cdot S \longrightarrow N \cdot T}\,(\text{Cons}_{\|}) \qquad \frac{S \longrightarrow S' \qquad T \longrightarrow T'}{S \circ T \longrightarrow S' \circ T'}\,(\text{Comp}_{\|})$$

$$\frac{M \longrightarrow M' \qquad N \longrightarrow N'}{(\lambda M\ N) \longrightarrow M'[N' \cdot \uparrow^0]}\,(\text{Beta}_{\|})$$

Lemma 2.40 *On semi-open expressions, $\mathcal{L}$ YH-commutes over $B_{\|}$, i.e. if x reduces in one $\mathcal{L}$-step to y, and in one $B_{\|}$-step to z, then there exists w such that $y \xrightarrow{\mathcal{L}^* B_{\|} \mathcal{L}^*} w$ and $z \xrightarrow{\mathcal{L}^*} w$.*

Proof. By induction on the depth of the $\mathcal{L}$-redex reduced in x. At the base case x is a $\mathcal{L}$-redex:

- $x = (M\ N)[S] \xrightarrow{(\text{Application})} (M[S]\ N[S]) = y$ and $(M\ N)[S] \xrightarrow{B_{\|}} (M'\ N')[S'] = z$, with $M \xrightarrow{B_{\|}} M'$, $N \xrightarrow{B_{\|}} N'$ and $S \xrightarrow{B_{\|}} S'$. By definition of $B_{\|}$, $(M[S]\ N[S]) \xrightarrow{B_{\|}}$ $(M'[S']\ N'[S'])$. But also, $(M'\ N')[S'] \xrightarrow{(\text{Application})} (M'[S']\ N'[S'])$.
- $x = (\lambda M\ N)[S] \xrightarrow{(\text{Application})} ((\lambda M)[S]\ N[S]) = y$ and $(\lambda M\ N)[S] \xrightarrow{B_{\|}} M'[N' \cdot \uparrow^0][S']$ $= z$, with $M \xrightarrow{B_{\|}} M'$, $N \xrightarrow{B_{\|}} N'$ and $S \xrightarrow{B_{\|}} S'$. Then, $y = ((\lambda M)[S]\ N[S]) \xrightarrow{(\text{Lambda})}$ $(\lambda M[\mathbf{1} \cdot (S \circ \uparrow^{Suc(0)})]\ N[S]) \xrightarrow{B_{\|}} M'[\mathbf{1} \cdot (S' \circ \uparrow^{Suc(0)})][N'[S'] \cdot \uparrow^0] \xrightarrow{\mathcal{L}^*}$ $M'[N'[(S')\downarrow_{\mathcal{L}}] \cdot (((S')\downarrow_{\mathcal{L}} \circ \uparrow^{Suc(0)}) \circ (N'[(S')\downarrow_{\mathcal{L}}] \cdot \uparrow^0))] \xrightarrow{\mathcal{L}^*} M'[N'[(S')\downarrow_{\mathcal{L}}] \cdot (S')\downarrow_{\mathcal{L}}]$. But also, $M'[N' \cdot \uparrow^0][S'] \xrightarrow{\mathcal{L}^*} M'[N'[(S')\downarrow_{\mathcal{L}}] \cdot (S')\downarrow_{\mathcal{L}}]$. This case is the only interesting one.
- $x = (\lambda M)[S] \xrightarrow{(\text{Lambda})} \lambda M[\mathbf{1} \cdot (S \circ \uparrow^{Suc(0)})] = y$ and $x = (\lambda M)[S] \xrightarrow{B_{\|}} (\lambda M')[S'] = z$, with $M \xrightarrow{B_{\|}} M'$ and $S \xrightarrow{B_{\|}} S'$. By definition of $B_{\|}$, $\lambda M[\mathbf{1} \cdot (S \circ \uparrow^{Suc(0)})] \xrightarrow{B_{\|}}$ $\lambda M'[\mathbf{1} \cdot (S' \circ \uparrow^{Suc(0)})]$. But also, $(\lambda M')[S'] \xrightarrow{(\text{Lambda})} \lambda M'[\mathbf{1} \cdot (S' \circ \uparrow^{Suc(0)})]$.
- The other cases are similar to the previous one.

□

Theorem 2.41 (Confluence) *The $\lambda_{\mathcal{L}}$-calculus is confluent on semi-open expressions.*

Proof. We verify that $\mathcal{L}$ and $B_{\|}$ satisfy the conditions of the Yokouchi-Hikita's lemma (Def. 2.5 and Lemma 2.6) on semi-open expressions, i.e.

1. $\mathcal{L}$ is terminating and confluent (Lemma 2.36 and Lemma 2.38).
2. $B_{\|}$ is strongly confluent, since (Beta) by itself is a left linear system with no critical pairs (c.f. [48]).

	$\lambda\sigma$	$\lambda_{\mathcal{L}}$	$\lambda_{\mathcal{S}_e}$	$\lambda\sigma_{\Uparrow}$	λ_ζ	$\lambda\upsilon$	λ_d	λx	$\lambda\chi$
Confluence	Mv	Mv	Mv	$\ddot\smile$	$\ddot\smile$	Gnd	Gnd	Gnd	Gnd
Normalization	Wk	Wk	?	Wk	PSN	PSN	PSN	PSN	PSN
Composition	$\ddot\smile$	$\ddot\smile$	$\overset{**}{\frown}$	$\ddot\smile$	$\overset{**}{\frown}$	$\overset{**}{\frown}$	$\ddot\sim^{\star}$	$\overset{**}{\frown}$	$\overset{**}{\frown}$
Finitary 1st-order	$\ddot\smile$	$\ddot\smile$	$\overset{**}{\frown}$	$\ddot\smile$	$\ddot\smile$	$\ddot\smile$	$\ddot\smile$	$\overset{**}{\frown}$	$\overset{**}{\frown}$
Variables	dB	dB	dB	dB	dB	dB	dB	Nm	Lv
Number of rules	13	12	$13^{\dagger}$	19	13	8	19	6	$10^{\dagger}$
β-reduction	$\ddot\smile$	$\ddot\smile$	$\ddot\smile$	$\ddot\smile$	$\ddot\sim^{\ddagger}$	$\ddot\smile$	$\ddot\smile$	$\ddot\smile$	$\ddot\smile$
Reference	[1]	[70]	[49]	[20]	[67]	[58]	[53]	[10]	[60]

$\ddot\smile$: The general property holds.
$\overset{**}{\frown}$: The property does not hold.
$\ddot\sim$: The property holds with restrictions.
Mv : Confluence on semi-open expressions, i.e. only with meta-variables of terms.
Gnd : Confluence on ground expressions.
Wk : Weak normalization on typed terms.
PSN : Preservation of strong normalization.
dB : De Bruijn indices notation of variables.
Nm : Variable names.
Lv : De Bruijn levels notation with variable names.
? : The PSN property of $\lambda_{\mathcal{S}_e}$ is a conjecture.
$\star$: Restricted composition. In particular, the λ_d-calculus does not allow simultaneous substitutions.
$\dagger$: Number of schemes. These calculi are not finitary.
$\ddagger$: Big-step semantic of β-reduction. The λ_ζ-calculus does not simulate each step of β-reduction.

Figure 2.8: Some calculi of explicit substitutions

3. $\mathcal{L}$ YH-commutes over $B_{\|}$ (Lemma 2.40).

Therefore, $\mathcal{L}^* B_{\|} \mathcal{L}^*$ is confluent on semi-open expressions.

Note that $\lambda_{\mathcal{L}} \subseteq \mathcal{L}^* B_{\|} \mathcal{L}^* \subseteq {\lambda_{\mathcal{L}}}^*$. Let x be a semi-open expressions, if $x \xrightarrow{{\lambda_{\mathcal{L}}}^*} y$ and $x \xrightarrow{{\lambda_{\mathcal{L}}}^*} z$, then there exists w such that $y \xrightarrow{(\mathcal{L}^* B_{\|} \mathcal{L}^*)^*} w$ and $z \xrightarrow{(\mathcal{L}^* B_{\|} \mathcal{L}^*)^*} w$. So, $y \xrightarrow{{\lambda_{\mathcal{L}}}^*} w$ and $z \xrightarrow{{\lambda_{\mathcal{L}}}^*} w$. □

2.4 Related Works and Summary

The λ-calculus uses an external and atomic operation to compute the substitutions of variables by terms. Calculi of explicit substitutions improve the substitution mechanism by allowing substitutions to be part of the formal language by means of special constructors and reduction rules. There are several versions of calculi of explicit substitutions. Figure 2.8 summarizes the main characteristics of some of them. All these calculi implement the β-reduction by means of a lazy mechanism of reduction of substitutions.

The λ-calculus with explicit substitutions has been proposed as a framework for higher-order unification [25, 26, 11, 55] and for representation of incomplete proofs [62, 68]. In these approaches, terms with holes are represented by open terms, i.e. terms with meta-variables. In order to reach these goals, most of the calculi of explicit substitutions have a strong drawback: non-confluence on terms with meta-variables.

We distinguish two possible research areas (not necessarily disjoint):

- to design a calculus that preserves all the properties of the λ-calculus, notably confluence and strong normalization on well-typed terms, and
- to design a calculus that enjoys the properties sufficient to represent incomplete proof-terms and to describe their construction mechanism.

For a couple of years, the incompatibility between confluence (on open terms) and preservation of strong normalization, was conjectured for the calculi of explicit substitutions. We solve this conjecture by proposing the λ_ζ-calculus which, as far as we know, is the first calculus of explicit substitutions that enjoys both properties of confluence and preservation of strong normalization at the same time. However, the λ_ζ-calculus lacks a composition operator. This operator was introduced in some calculi in order to gain local confluence. But also, it happens to be useful for other purposes, for instance the modeling of closures of an abstract machine [40] or the pruning of search space in unification algorithms [25, 26, 55]. Actually, we do not use the λ_ζ-calculus to represent incomplete proof-terms. However, we think that λ_ζ is a first step in the first direction mentioned above.

On the other hand, the $\lambda\sigma$-calculus [1] is confluent on expressions with meta-variables of terms [81] and it is weakly normalizing on typed expressions [38, 70]. These two properties are sufficient to decide the equivalence of terms with meta-variables. In addition, the $\lambda\sigma$-calculus uses general composition of substitutions and simultaneous substitutions. However, $\lambda\sigma$ is not left-linear, and this feature raises technical problems in some frameworks [74, 55, 69]. Thus, we propose a variant of $\lambda\sigma$, namely $\lambda_\mathcal{L}$. The new calculus enjoys the same general properties as $\lambda\sigma$, but in contrast to it, $\lambda_\mathcal{L}$ is left-linear in the sort of terms and substitutions.

Another left-linear variant of $\lambda\sigma$ is the $\lambda\sigma_{\Uparrow}$-calculus [20]. The $\lambda\sigma_{\Uparrow}$-calculus is fully confluent on open expressions, not only with meta-variables of terms but also with meta-variables of substitutions. However, on open expressions, $\lambda\sigma_{\Uparrow}$ is incompatible with η-conversions due to the fact that substitutions id and $\mathbf{1}\cdot\uparrow$ are not $\lambda\sigma_{\Uparrow}$-convertible. To see why, consider the term $\lambda((\lambda M)[\uparrow]\ \mathbf{1})$. It is η-convertible to λM, but also

$$\lambda((\lambda M)[\uparrow]\ \mathbf{1}) \xrightarrow{\text{(Lambda)}} \lambda(\lambda M[\Uparrow(\uparrow)]\ \mathbf{1}) \xrightarrow{\text{(Beta)}} \lambda M[\Uparrow(\uparrow)][\mathbf{1}\cdot id] \xrightarrow{\sigma_{\Uparrow}{}^*} \lambda M[\mathbf{1}\cdot\uparrow]$$

The terms λM and $\lambda M[\mathbf{1}\cdot\uparrow]$ are not $\lambda\sigma_{\Uparrow}$-convertible on open expressions, but they are on ground expressions.

The above problem does not happen in $\lambda_\mathcal{L}$. The extensional version of $\lambda_\mathcal{L}$-calculus, i.e. $\lambda_\mathcal{L}$ with a η-rule, is confluent on ground terms as shown in [54], and we conjecture that it is also on semi-open expressions.

Both calculi $\lambda_\mathcal{L}$ and $\lambda\sigma$ share the same description of normal forms; so, the higher-order unification algorithm via explicit substitutions proposed in [25] can be expressed in $\lambda_\mathcal{L}$ with minor modifications.

Chapter 3

The System $\mathrm{CC}_{\mathcal{L}}$

The goal of this chapter is to propose a calculus of explicit substitutions for dependent type theories. In particular, we present the system $\mathrm{CC}_{\mathcal{L}}$ which is a formulation of the $\lambda_{\mathcal{L}}$-calculus in the calculus of constructions. The choice of the $\lambda_{\mathcal{L}}$-calculus is motivated by the properties that we want to preserve: type uniqueness, subject reduction, weak normalization and Church-Rosser, all of them on semi-open expressions. As we have suggested in Chapter 1 and we will see in Chapter 4, these properties are sufficient to raise our main goals: representation of proofs with place-holders and incremental construction of proofs.

Type systems for calculi of explicit substitutions with open terms present two difficulties: the typing of meta-variables and the typing of substitutions. The intended use of meta-variables in λ-calculus is to represent holes. As we have pointed out in Section 1.4.1, the soundness of the process of filling holes is guaranteed by a commutation property between instantiation of meta-variables and typing. In the simply-typed $\lambda\sigma$-calculus this property holds when each meta-variable is typed in a unique context by a unique type [25, 62].

On the other hand, a substitution being a list of terms in λ_{dB}, it seems natural to type an explicit substitution by a list of types [1].

We follow the above approaches: *to type meta-variables by a unique context and a unique type, and to type substitutions by a list of types*, to formulate three type extensions of $\lambda_{\mathcal{L}}$: simple types, dependent types ($\lambda\Pi_{\mathcal{L}}$), and the calculus of constructions ($\mathrm{CC}_{\mathcal{L}}$). The proof of the main properties that we state in this chapter will be given in the Part 2 of this thesis.

3.1 Simple Types

The simply-typed extension of $\lambda_{\mathcal{L}}$ is strongly inspired in that of $\lambda\sigma$ [1]. However, we use a little different notation for contexts (lists of types), and for application of substitutions.

Thus, contrary to the $\lambda\sigma$-tradition, —but just like in usual mathematical notation— we denote the application of a substitution S to a term M by $[S]M$ (and not $M[S]$), the composition of substitutions S and T by $T \circ S$ (and not $S \circ T$), and the simultaneous substitutions by $S \cdot M$ (and not $M \cdot S$). In the same way, we use reversed lists to represents contexts. An advantage of this notation will be evident when we introduce dependent types: just as in programming languages, a variable can be used after its declaration.

$(\lambda_A M\ N)$	$\longrightarrow$	$[\uparrow^0 \cdot N]M$	(Beta)
$[S](\lambda_A M)$	$\longrightarrow$	$\lambda_A[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}]M$	(Lambda)
$[S](M\ N)$	$\longrightarrow$	$([S]M\ [S]N)$	(Application)
$[T][S]M$	$\longrightarrow$	$[T \circ S]M$	(Clos)
$[S \cdot M]\mathbf{1}$	$\longrightarrow$	M	(VarCons)
$[\uparrow^0]M$	$\longrightarrow$	M	(Id)
$T \circ (S \cdot M)$	$\longrightarrow$	$(T \circ S) \cdot [T]M$	(Map)
$S \circ \uparrow^0$	$\longrightarrow$	S	(IdS)
$(S \cdot M) \circ \uparrow^{Suc(n)}$	$\longrightarrow$	$S \circ \uparrow^n$	(ShiftCons)
$\uparrow^m \circ \uparrow^{Suc(n)}$	$\longrightarrow$	$\uparrow^{Suc(m)} \circ \uparrow^n$	(ShiftShift)
$\uparrow^{Suc(0)} \cdot \mathbf{1}$	$\longrightarrow$	$\uparrow^0$	(Shift0)
$\uparrow^{Suc(n)} \cdot [\uparrow^n]\mathbf{1}$	$\longrightarrow$	$\uparrow^n$	(ShiftS)

Figure 3.1: The rewrite system $\lambda_{\mathcal{L}}$ in Church Style

3.1.1 The Simply-Typed Version of $\lambda_{\mathcal{L}}$

We consider a simply-typed theory, where types are generated from a set of basic types $a, b, \ldots$ and the arrow "$\rightarrow$" type constructor.

$$\textbf{Types} \quad A, B \quad ::= \quad a, b, \ldots \mid A \rightarrow B$$

Typed terms in $\lambda_{\mathcal{L}}$ differ from untyped ones only in abstractions. In order to have a type uniqueness property, we prefer a *Church style* notation where types of binder variables appear explicitly in the syntax.

$$\begin{array}{lllll} \textbf{Terms} & M, N & ::= & \mathbf{1} \mid \lambda_A.M \mid (M\ N) \mid [S]M \\ \textbf{Substitutions} & S, T & ::= & \uparrow^n \mid S \cdot M \mid T \circ S \end{array}$$

The $\lambda_{\mathcal{L}}$-calculus is modified according to this new syntax of abstractions as shown in Fig. 3.1. Moreover, it is not difficult to see that properties of Section 2.3 are preserved.

In typed λ-calculus, free variables are declared in structures called *contexts*. Typing assertions depend on contexts. In λ-calculus with de Bruijn's indices, contexts are just lists of types where the i-th element is the type of the i-th free variable. Contexts are defined inductively as follows:

$$\textbf{Contexts} \quad \Gamma \quad ::= \quad nil \mid \Gamma.A$$

Typing assertions have one of the following forms:

- $\Gamma \vdash M : A$, the term M has type A in the context Γ.

- $\Gamma \vdash S \triangleright \Delta$, the substitution S has type Δ in the context Γ.

The typing rules of the simply-typed version of $\lambda_{\mathcal{L}}$ are:

$$\frac{}{\Gamma.A \vdash \mathbf{1} : A}\,(\text{Var}) \qquad \frac{\Gamma.A \vdash M : B}{\Gamma \vdash \lambda_A.M : A{\to}B}\,(\text{Abs})$$

$$\frac{\Gamma \vdash M : A{\to}B \quad \Gamma \vdash N : A}{\Gamma \vdash (M\ N) : B}\,(\text{Appl}) \qquad \frac{\Gamma \vdash S \triangleright \Delta \quad \Delta \vdash M : A}{\Gamma \vdash [S]M : A}\,(\text{Clos})$$

$$\frac{}{\Gamma \vdash \uparrow^0 \triangleright \Gamma}\,(\text{Id}) \qquad \frac{\Gamma \vdash \uparrow^n \triangleright \Delta}{\Gamma.A \vdash \uparrow^{Suc(n)} \triangleright \Delta}\,(\text{Shift})$$

$$\frac{\Gamma \vdash S \triangleright \Delta' \quad \Delta' \vdash T \triangleright \Delta}{\Gamma \vdash S \circ T \triangleright \Delta}\,(\text{Comp}) \qquad \frac{\Gamma \vdash S \triangleright \Delta \quad \Gamma \vdash M : A}{\Gamma \vdash S \cdot M \triangleright \Delta.A}\,(\text{Cons})$$

In the same way as in $\lambda\sigma$ (c.f [25]), each meta-variable is typed in a unique context by a unique type:

$$\frac{}{\Gamma_X \vdash X : A_X}\,(\text{Meta}_X)$$

Example 3.1 *In the following example we take the index* $\underline{2}$ *as a notation for* $[\uparrow^{Suc(0)}]\mathbf{1}$.

1. *This is a typing derivation of* $nil.A \vdash \lambda_B.(X\ \underline{2}) : B{\to}C$.

$$\dfrac{\dfrac{\dfrac{}{nil.A.B \vdash X : A{\to}C}\,(\text{Meta}_X) \quad \dfrac{\dfrac{\dfrac{}{nil.A \vdash \uparrow^0 \triangleright nil.A}\,(\text{Id})}{nil.A.B \vdash \uparrow^{Suc(0)} \triangleright nil.A}\,(\text{Shift}) \quad \dfrac{}{nil.A \vdash \mathbf{1} : A}\,(\text{Var})}{nil.A.B \vdash \underline{2} : A}\,(\text{Clos})}{nil.A.B \vdash (X\ \underline{2}) : C}\,(\text{Appl})}{nil.A \vdash \lambda_B.(X\ \underline{2}) : B{\to}C}\,(\text{Abs})$$

2. *The term* $(\lambda_A.X\ X)$ *is not well-typed in any context. Notice that in the following derivation:*

$$\dfrac{\dfrac{\dfrac{\dots}{\Gamma.A \vdash X : A}}{\Gamma \vdash \lambda_A.X : A{\to}A}\,(\text{Abs}) \qquad \dfrac{\dots}{\Gamma \vdash X : A}}{\Gamma \vdash (\lambda_A.X\ X) : A{\to}A}\,(\text{Appl})$$

the meta-variable X *would be typed in two different contexts* $\Gamma.A$ *and* Γ.

3. *Let* X *be a meta-variable such that* $\Gamma \vdash X : A$. *We have the valid typing judgment:*

$$\Gamma \vdash (\lambda_A.\lambda_B.\underline{2}\ X) : B{\to}A \tag{3.1}$$

We get by $\lambda_{\mathcal{L}}$*-reduction:*

$$(\lambda_A.\lambda_B.\underline{2}\ X) \xrightarrow{(\text{Beta})} [\uparrow^0 \cdot X](\lambda_B.\underline{2}) \xrightarrow{\lambda_{\mathcal{L}}{}^*} \lambda_B.[\uparrow^{Suc(0)}]X$$

Also, we can verify that

$$\Gamma \vdash \lambda_B.[\uparrow^{Suc(0)}]X : B{\to}A \tag{3.2}$$

In the simply-typed version of $\lambda_{\mathcal{L}}$, just as in that of $\lambda\sigma$, instantiation of meta-variables and typing commute. This commutation property guarantees the soundness of instantiation of meta-variables in the unification algorithm [25, 26, 55], and in refinement steps of incomplete proofs [62, 68].

If x is an arbitrary expression, i.e. x is a term or a substitution, X is a meta-variable, and N a term, then $x\{X/N\}$ is a notation for the term x in which all the occurrences of X has been replaced by N, without taking care of possible captures of free variables. Formally,

Definition 3.2 (Instantiation) *An* instantiation *is a pair of terms (X, N), denoted by $\{X/N\}$, where X is a meta-variable and N is an arbitrary term.*

The application of an instantiation $\{X/N\}$ to an expression x, *denoted by $x\{X/N\}$ is inductively defined on x as follows:*

$$\begin{array}{lcl}
\mathbf{1}\{X/N\} & = & \mathbf{1} \\
([S]M)\{X/N\} & = & [S\{X/N\}]M\{X/N\} \\
(\lambda_A.M)\{X/N\} & = & \lambda_A.M\{X/N\} \\
(M_1\ M_2)\{X/N\} & = & (M_1\{X/N\}\ M_2\{X/N\}) \\
X\{X/N\} & = & N \\
Y\{X/N\} & = & Y \quad \textit{if } X \neq Y \\
\uparrow^n\{X/N\} & = & \uparrow^n \\
(T \circ S)\{X/N\} & = & T\{X/N\} \circ S\{X/N\} \\
(S \cdot M)\{X/N\} & = & S\{X/N\} \cdot M\{X/N\}
\end{array}$$

Lemma 3.3 (Instantiation Soundness) *Let X be a meta-variable and N a term such that $\Gamma_X \vdash N : A_X$,*

1. *if $\Delta \vdash M : B$, then $\Delta \vdash M\{X/N\} : B$, and*
2. *if $\Delta \vdash S \triangleright \Delta'$, then $\Delta \vdash S\{X/N\} \triangleright \Delta'$.*

Proof. By induction on typing derivation. □

The next property justifies the use of $\lambda_{\mathcal{L}}$ to build proof-terms in λ_{dB}.

Definition 3.4 *A term M in $\lambda_{\mathcal{L}}$ is said to be* pure *if M is a $\mathcal{L}$-normal form of a ground term.*

Pure terms in $\lambda_{\mathcal{L}}$ are isomorphic to terms in λ_{dB} by taking $\mathbf{1} \equiv \underline{1}$ and $[\uparrow^{Suc(n)}]\mathbf{1} \equiv \underline{n+1}$. Thus, for the sake of simplicity, we identify pure terms in $\lambda_{\mathcal{L}}$ with terms in λ_{dB}.

We recall the simply-typed version of λ_{dB}.

$$\frac{}{\Gamma.A \vdash_{\mathrm{dB}} \underline{1} : A}\ (\mathrm{Var}_1) \qquad \frac{\Gamma \vdash_{\mathrm{dB}} \underline{n} : B}{\Gamma.A \vdash_{\mathrm{dB}} \underline{n+1} : B}\ (\mathrm{Var}_{n+1})$$

$$\frac{\Gamma.A \vdash_{\mathrm{dB}} M : B}{\Gamma \vdash_{\mathrm{dB}} \lambda_A.M : A \to B}\ (\mathrm{Abs}) \qquad \frac{\Gamma \vdash_{\mathrm{dB}} M : A \to B \quad \Gamma \vdash_{\mathrm{dB}} N : A}{\Gamma \vdash_{\mathrm{dB}} (M\ N) : B}\ (\mathrm{Appl})$$

Theorem 3.5 (Typing Soundness with Respect to λ_{dB}) *Let M be a pure term in $\lambda_{\mathcal{L}}$, $\Gamma \vdash M : A$ if and only if $\Gamma \vdash_{\mathrm{dB}} M : A$.*

Proof. First, we show by induction on n that $\Gamma \vdash \underline{n} : A$ if and only if $\Gamma \vdash_{\mathrm{dB}} \underline{n} : A$. The other cases are proved by induction on typing derivation. □

3.1.2 Properties

The simply-typed version of the $\lambda_{\mathcal{L}}$-calculus enjoys almost the same properties as the simply-typed λ-calculus, e.g. type uniqueness, subject reduction, weak normalization and confluence, but not strong normalization.

The following lemma simplifies the proofs to come.

Lemma 3.6 (Inversion)

1. *If* $\Gamma \vdash M : A$, *then*
   - *if* $M = \mathbf{1}$, *then* $\Gamma = \Gamma'.A$;
   - *if* $M = X$ *(a meta-variable), then* $\Gamma = \Gamma_X$ *and* $A = A_X$;
   - *if* $M = \lambda_{A'}.M'$, *then* $\Gamma.A' \vdash M' : B'$ *and* $A = A' \to B'$;
   - *if* $M = (M'\ N')$, *then* $\Gamma \vdash M' : B \to A$ *and* $\Gamma \vdash N' : B$, *and*
   - *if* $M = [T]M'$, *then* $\Gamma \vdash T \triangleright \Delta'$ *and* $\Delta' \vdash M' : A$.
2. *If* $\Gamma \vdash S \triangleright \Delta$, *then*
   - *if* $S = \uparrow^0$, *then* $\Gamma = \Delta$, *and*
   - *if* $S = \uparrow^{Suc(n)}$, *then* $\Gamma = \Gamma'.A'$ *and* $\Gamma' \vdash \uparrow^n \triangleright \Delta$, *and*
   - *if* $S = T \cdot M'$, *then* $\Gamma \vdash M' : A'$, $\Gamma \vdash T \triangleright \Delta'$ *and* $\Delta = \Delta'.A'$, *and*
   - *if* $S = T \circ S'$, *then* $\Gamma \vdash T \triangleright \Delta'$, $\Delta' \vdash S' \triangleright \Delta$.

Proof. Note that the typing system is syntax directed, i.e. there is one rule for each constructor of terms and substitutions. □

Lemma 3.7 (Type Uniqueness)

1. *If* $\Gamma \vdash M : A_1$ *and* $\Gamma \vdash M : A_2$, *then* $A_1 = A_2$, *and*
2. *if* $\Gamma \vdash S \triangleright \Delta_1$ *and* $\Gamma \vdash S \triangleright \Delta_2$, *then* $\Delta_1 = \Delta_2$.

Proof. By simultaneous structural induction on M and S. □

Example 3.1(3) suggests that typing is preserved under $\lambda_{\mathcal{L}}$-reductions. This property is known as *subject reduction*. In order to prove the subject reduction property, we prove that each $\lambda_{\mathcal{L}}$-rewrite rule preserves typing.

Lemma 3.8 *The $\lambda_{\mathcal{L}}$-rewrite system preserves typing.*

Proof. We verify the property for each rewrite rule.

- (Beta). Let $\Gamma \vdash (\lambda_B.M\ N) : A$, we show $\Gamma \vdash [\uparrow^0 \cdot N]M : A$. We have the following hypotheses:
  1. (a) $\Gamma \vdash \lambda_B.M : B \to A$ and (b) $\Gamma \vdash N : B$, by Lemma 3.6 applied to hypothesis $\Gamma \vdash (\lambda_B.M\ N) : A$.

2. $\Gamma.B \vdash M : A$, by Lemma 3.6 applied to (1-a).
3. $\Gamma \vdash \uparrow^0 \triangleright \Gamma$, by (Id).
4. $\Gamma \vdash \uparrow^0 \cdot N \triangleright \Gamma.B$, by (Cons) applied to (1-b) and (3).

We conclude with (Clos) applied to (2) and (4) that $\Gamma \vdash [\uparrow^0 \cdot N]M : A$.

- (Lambda). Let $\Gamma \vdash [S](\lambda_B.M) : A$, we show $\Gamma \vdash \lambda_B.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}]M : A$. We have the following hypotheses:

  1. (a) $\Gamma \vdash S \triangleright \Delta$ and (b) $\Delta \vdash \lambda_B.M : A$, by Lemma 3.6 applied to hypothesis $\Gamma \vdash [S](\lambda_B.M) : A$.
  2. (a) $\Delta.B \vdash M : A'$ and (b) $A = B \rightarrow A'$, by Lemma 3.6 applied to (1-b).
  3. $\Gamma \vdash \uparrow^0 \triangleright \Gamma$, by (Id).
  4. $\Gamma.B \vdash \uparrow^{Suc(0)} \triangleright \Gamma$, by (Shift) applied to (3).
  5. $\Gamma.B \vdash \mathbf{1} : B$, by (Var).
  6. $\Gamma.B \vdash \uparrow^{Suc(0)} \circ S \triangleright \Delta$, by (Comp) applied to (1-a) and (4).
  7. $\Gamma.B \vdash (\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1} \triangleright \Delta.B$, by (Cons) applied to (5) and (6).
  8. $\Gamma.B \vdash [(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}]M : A'$, by (Clos) applied to (2-a) and (7).

  We conclude with (Abs) applied to (8) that $\Gamma \vdash \lambda_B.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}]M : B \rightarrow A'$.

- The other cases are similar.

□

Theorem 3.9 (Subject Reduction) *Let x and y be such that $x \xrightarrow{\lambda_{\mathcal{L}}{}^*} y$,*

- *if x is a term and $\Gamma \vdash x : A$, then $\Gamma \vdash y : A$, and*
- *if x is a substitution and $\Gamma \vdash x \triangleright \Delta$, then $\Gamma \vdash y \triangleright \Delta$.*

Proof. We show that typing is preserved for one-step reductions (i.e. $\xrightarrow{\lambda_{\mathcal{L}}}$), and then it is also for the reflexive and transitive closure (i.e. $\xrightarrow{\lambda_{\mathcal{L}}{}^*}$). Let $x \xrightarrow{\lambda_{\mathcal{L}}} y$ be a one-step reduction, we proceed by induction on the depth of the redex reduced in x. At the initial case x is reduced at the top level. In this case we conclude with Lemma 3.8. At the induction step we resolve with Lemma 3.6 and induction hypothesis. □

Strong normalization on typed terms does not hold for $\lambda_{\mathcal{L}}$. Nevertheless, in Section 7.1 we show that the simply-typed version of $\lambda_{\mathcal{L}}$ is weakly normalizing.

3.2 Dependent Types and the Calculus of Constructions

The Dependent Type theory presented in [41], namely $\lambda\Pi$, extends the Curry-Howard isomorphism to universal quantifiers. In this theory, the concepts of terms and types are identified, but terms are stratified in several levels called *sorts*. In order to represent sorts, the constants of terms *Type* and

Kind[1] are added to the syntax. The arrow type is generalized by a new binding structure called *product*. The extension of $\lambda\Pi$ with polymorphism and constructions of types is called the *Calculus of Constructions* [14, 15].

Well-formed contexts and terms in the λ_{dB}-calculus with products are defined by the following grammar:

Contexts	Γ	$::=$	$nil \mid \Gamma.M$
Terms	M, N	$::=$	$Kind \mid Type \mid \underline{n} \mid \Pi_M.N \mid \lambda_M.N \mid (M\ N)$

A term of the form $\Pi_M.N$ is called a *product type*, and it is a binding structure.

In contrast to the simply-typed case, in a λ-calculus with products, terms and types are in the same syntactical category. The stratification of terms by means of the sorts is necessary to avoid circular typing judgments as "$Type : Type$", which leads to the Girard's paradox. So, the term *Kind* is used as the valid type of the term *Type*.

The relation $\xrightarrow{\beta}$ is defined as the contextual closure of the rule

$$(\lambda'_M.M\ N) \longrightarrow M\{1 := N\}$$

Since types may depend on terms, when a substitution —or an updating operation of indices— is distributed into a term, all its subterms may be affected by the substitution, including the type of the bound variables in abstractions and products. Thus, we generalize the functions $\{n := N\}$ and τ_i^n defined in Section 2.1.1 in the following way:

$$\begin{aligned}
(M\ M')\{n := N\} &= (M\{n := N\}\ M'\{n := N\}) \\
(\lambda_M.M')\{n := N\} &= \lambda_{M\{n:=N\}}.M'\{n{+}1 := N\} \\
(\Pi_M.M')\{n := N\} &= \Pi_{M\{n:=N\}}.M'\{n{+}1 := N\} \\
\underline{m}\{n := N\} &= \begin{cases} \underline{m-1} & \text{if } m > n \\ \tau_0^{n-1}(M) & \text{if } m = n \\ \underline{m} & \text{if } m < n \end{cases}
\end{aligned}$$

and

$$\begin{aligned}
\tau_i^n(M\ N) &= (\tau_i^n(M)\ \tau_i^n(N)) \\
\tau_i^n(\lambda_M.N) &= \lambda_{\tau_i^n(M)}.\tau_{i+1}^n(N) \\
\tau_i^n(\Pi_M.N) &= \Pi_{\tau_i^n(M)}.\tau_{i+1}^n(N) \\
\tau_i^n(\underline{m}) &= \begin{cases} \underline{m+n} & \text{if } m > i \\ \underline{m} & \text{if } m \leq i \end{cases}
\end{aligned}$$

We use $M{\rightarrow}N$ as an abbreviation for $\Pi_M.\tau_0^1(N)$. In a λ-calculus with names, $M{\rightarrow}N$ is an abbreviation for $\Pi x{:}M.N$ where x does not appear free in N.

As we have said in Section 1.3, not any well-formed context is valid and so typing assertions for contexts and terms are mutually dependent.

We consider typing assertions having one of the following forms:

- $\vdash \Gamma$, the context Γ is valid.
- $\Gamma \vdash M : N$, the term M has type N in Γ.

[1] These names are not standard, other couples of names used in the literature are: $(Set, Type)$, $(Prop, Type)$ and $(*, \Box)$.

We recall that terms are stratified by means of the sorts as follows:

1. if $\Gamma \vdash M : \mathit{Kind}$, then we say that M *is a kind.*
2. if $\Gamma \vdash M : \mathit{Type}$, then we say that M *is a type.*
3. if $\Gamma \vdash M : N$ and $\Gamma \vdash N : \mathit{Kind}$, then we say that M *is a constructor.*
4. if $\Gamma \vdash M : N$ and $\Gamma \vdash N : \mathit{Type}$, then we say that M *is an object.*

Notation: We use the lowercase letter s to range over the set of sorts $\{\mathit{Kind}, \mathit{Type}\}$.

Valid Contexts

$$\frac{}{\vdash_{\mathrm{dB}} \mathit{nil}}\,(\text{Empty}) \qquad\qquad \frac{\Gamma \vdash_{\mathrm{dB}} M : s}{\vdash_{\mathrm{dB}} \Gamma.M}\,(\text{Var-Decl})$$

The $\lambda\Pi_{\mathrm{dB}}$ system, which is a formulation of $\lambda\Pi$ using de Bruijn's indices, is defined by the following typing rules:

Valid Terms

$$\frac{\vdash_{\mathrm{dB}} \Gamma}{\Gamma \vdash_{\mathrm{dB}} \mathit{Type} : \mathit{Kind}}\,(\text{Type}) \qquad \frac{\Gamma \vdash_{\mathrm{dB}} M : \mathit{Type} \quad \Gamma.M \vdash_{\mathrm{dB}} N : s}{\Gamma \vdash_{\mathrm{dB}} \Pi_M.N : s}\,(\text{Prod})$$

$$\frac{\vdash_{\mathrm{dB}} \Gamma.M}{\Gamma.M \vdash_{\mathrm{dB}} \mathbf{1} : \tau_0^1(M)}\,(\mathrm{Var}_1) \qquad \frac{\vdash_{\mathrm{dB}} \Gamma.M \quad \Gamma \vdash_{\mathrm{dB}} \underline{n} : N}{\Gamma.M \vdash_{\mathrm{dB}} \underline{n+1} : \tau_0^1(N)}\,(\mathrm{Var}_{n+1})$$

$$\frac{\Gamma \vdash_{\mathrm{dB}} M : \Pi_{M_1}.M_2 \quad \Gamma \vdash_{\mathrm{dB}} N : M_1}{\Gamma \vdash_{\mathrm{dB}} (M\ N) : M_2\{1 := N\}}\,(\text{Appl})$$

$$\frac{\Gamma \vdash_{\mathrm{dB}} M_1 : \mathit{Type} \quad \Gamma.M_1 \vdash_{\mathrm{dB}} M_2 : N \quad \Gamma \vdash_{\mathrm{dB}} \Pi_{M_1}.N : s}{\Gamma \vdash_{\mathrm{dB}} \lambda_{M_1}.M_2 : \Pi_{M_1}.N}\,(\text{Abs})$$

$$\frac{\Gamma \vdash_{\mathrm{dB}} M : M_1 \quad \Gamma \vdash_{\mathrm{dB}} M_2 : s \quad M_1 =_{\lambda_{\mathrm{dB}}} M_2}{\Gamma \vdash_{\mathrm{dB}} M : M_2}\,(\text{Conv})$$

The system CC_{dB}, which is a formulation of the Calculus of Constructions with de Bruijn's indices, is obtained by replacing the rule (Prod) and (Abs) of $\lambda\Pi_{\mathrm{dB}}$ by

$$\frac{\Gamma.M \vdash_{\mathrm{dB}} N : s}{\Gamma \vdash_{\mathrm{dB}} \Pi_M.N : s}\,(\text{Prod}) \qquad \frac{\Gamma.M_1 \vdash_{\mathrm{dB}} M_2 : N \quad \Gamma \vdash_{\mathrm{dB}} \Pi_{M_1}.N : s}{\Gamma \vdash_{\mathrm{dB}} \lambda_{M_1}.M_2 : \Pi_{M_1}.N}\,(\text{Abs})$$

The major difference between the versions of the systems presented in Section 1.3 —based on names— with those presented here —based on de Bruijn's notation–, are the updating operation of the indices. For instance, in the premise of (Var_1) the indices of M are relative to Γ, but in the conclusion, the index $\mathbf{1}$ is typed in the context $\Gamma.M$, thus the indices representing free variables in M must be shifted by one.

Now, we describe a typed version of the $\lambda_{\mathcal{L}}$-calculus with dependent types, polymorphism and constructions of types. Extensions to the simply-typed version of $\lambda\sigma$ lead to unexpected problems. First, it is well-known that the simply-typed $\lambda\sigma$-calculus does not enjoy the same properties as the simply-typed λ-calculus. For instance, as we have said in Section 2.1.2, $\lambda\sigma$ does not preserve strong normalization. Thus, we cannot expect to have this property in more complex type theories based on $\lambda\sigma$. Furthermore, in dependent type theories —as $\lambda\Pi$ or the Calculus of Constructions— the notions of well-typed expressions with meta-variables and well-typed substitutions are not simple due to mutual dependences between types and terms.

3.2.1 Typing of Meta-variables

As we have explained, in order to deal with renaming issues, we use a calculus based on the de Bruijn indices notation. In this way, variables are declared by means of a list of types where the i-th element is the type of the i-th free variable.

On the other hand, since the instantiation of meta-variables does not take care of renaming problems, we prefer a context with names, called *signature*, to declare meta-variables. A *meta-variable declaration* has the form $X{:}_{\Gamma} M$, where Γ and M are, respectively, a context and a type associated to the meta-variable X.

Definition 3.10 (Contexts and Signatures) *Well formed* contexts *and* signatures *are defined by the following grammar:*

$$\begin{array}{llll} \textbf{Contexts} & \Gamma & ::= & nil \mid \Gamma.M \\ \textbf{Signatures} & \Sigma & ::= & nil \mid \Sigma.\, X{:}_{\Gamma} M \end{array}$$

We consider typing assertions having one of the following forms:

- $\vdash \Sigma :\cdot \Gamma$, the context Γ is valid in the signature Σ.
- $\Sigma :\cdot \Gamma \vdash M : N$, the term M has type N in $\Sigma :\cdot \Gamma$.
- $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, the substitution S has as type the context Δ in $\Sigma :\cdot \Gamma$.

In dependent type theories, meta-variables may appear in types and in contexts. Hence, we need scoping rules for meta-variables and variables.

- In the typing assertion $\vdash \Sigma_1.\, X{:}_{\Gamma} M.\, \Sigma_2 :\cdot \Delta$, the context Γ and the term M can use all the meta-variables declared in Σ_1, the indices in M are relative to Γ, and the context Δ can use all the meta-variables declared in the signature $\Sigma_1.\, X{:}_{\Gamma} M.\, \Sigma_2$.
- In the typing assertions $\Sigma :\cdot \Gamma \vdash M : N$ and $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, the terms M, N, the substituion S and the context Δ can use all the meta-variables declared in Σ, and the indices in M, N and S are relative to Γ.

Notation: We use $\vdash \Sigma$, $\vdash \Gamma$, $\Gamma \vdash M : N$, and $\Gamma \vdash S \triangleright \Delta$ as notations for $\vdash \Sigma :\cdot nil$, $\vdash nil :\cdot \Gamma$, $nil :\cdot \Gamma \vdash M : N$, and $nil :\cdot \Gamma \vdash S \triangleright \Delta$, respectively.

3.2.2 Typing of Substitutions

Now we address the question about typing of substitutions in dependent type theories.

Let us consider the following example due to Geuvers and Bloo [34]. Take the context[2]

$$\Gamma = \mathit{nil}.\ \mathit{nat}{:}\mathit{Type}.\ T{:}\mathit{nat}{\rightarrow}\mathit{Type}.\ z{:}\mathit{nat}$$

We verify in $\lambda\Pi_{\mathrm{dB}}$ (and in CC_{dB}) that

$$\Gamma \vdash \lambda x{:}\mathit{nat}.\lambda f{:}((T\ x){\rightarrow}\mathit{nat}).\lambda y{:}(T\ x).(f\ y) : \Pi x{:}\mathit{nat}.((T\ x){\rightarrow}\mathit{nat}){\rightarrow}((T\ x).\mathit{nat}) \tag{3.3}$$

and that

$$\Gamma \vdash z : \mathit{nat} \tag{3.4}$$

Thus, applying (Appl) to Eq. 3.3 and Eq. 3.4 we get

$$\Gamma \vdash (\lambda x{:}\mathit{nat}.\lambda f{:}((T\ x){\rightarrow}\mathit{nat}).\lambda y{:}(T\ x).(f\ y)\ z) : ((T\ z){\rightarrow}\mathit{nat}){\rightarrow}((T\ z){\rightarrow}\mathit{nat}) \tag{3.5}$$

Assume that we reduce the outermost redex by using an explicit substitution calculus, i.e

$$(\lambda x{:}\mathit{nat}.\lambda f{:}((T\ x){\rightarrow}\mathit{nat}).\lambda y{:}(T\ x).(f\ y)\ z) \xrightarrow{\text{(Beta)}} [x := z](\lambda f{:}((T\ x){\rightarrow}\mathit{nat}).\lambda y{:}(T\ x).(f\ y))$$

When we go trough the first abstraction and distribute the substitution, but without crossing the second abstraction, we get

$$[x := z](\lambda f{:}((T\ x){\rightarrow}\mathit{nat}).\lambda y{:}(T\ x).(f\ y)) \longrightarrow \lambda f{:}((T\ z){\rightarrow}\mathit{nat}).[x := z \cdot f := f](\lambda y{:}(T\ x).(f\ y))$$

According to the rule (Abs) we must type the term $[x := z \cdot f := f](\lambda y{:}(T\ x).(f\ y))$ in the context $\Gamma.\ f{:}(T\ z){\rightarrow}\mathit{nat}$. In order to type this term, we consider the typing rules for closures and simultaneous substitutions for the simply-typed version of $\lambda_{\mathcal{L}}$:

$$\frac{\Gamma \vdash S \triangleright \Delta \quad \Delta \vdash M : A}{\Gamma \vdash [S]M : A}\ (\mathrm{Clos}) \qquad \frac{\Gamma \vdash S \triangleright \Delta \quad \Gamma \vdash M : A}{\Gamma \vdash S \cdot M \triangleright \Delta.A}\ (\mathrm{Cons})$$

Thus, we must first type the substitution $[x := z \cdot f := f]$ in the context $\Gamma.\ f{:}(T\ z){\rightarrow}\mathit{nat}$, i.e.

$$\Gamma \vdash [x := z \cdot f := f] \triangleright \Gamma.\ x{:}\mathit{nat}.\ f{:}(T\ z){\rightarrow}\mathit{nat}$$

and then to type $\lambda y{:}(T\ x).(f\ y)$ in the obtained context $\Gamma.\ x{:}\mathit{nat}.\ f{:}(T\ z){\rightarrow}\mathit{nat}$. However, this term is ill-typed since y has type $(T\ x)$, f has type $(T\ z){\rightarrow}\mathit{nat}$ and $(T\ x)$ is not convertible to $(T\ z)$.

Of course the dependent-typed versions of rules (Clos) and (Cons) must take into account that types and terms are mutually dependent. If we analyze the rule (Appl) in Section 3.2, we can see that a more appropriate typing rule for closures in a dependent type theory has the form

$$\frac{\Gamma \vdash S \triangleright \Delta \quad \Delta \vdash M : N}{\Gamma \vdash [S]M : [S]N}\ (\Pi\text{-Clos})$$

This is because when the substitution S modifies the term M, it modifies its type too.

[2] For readability, when discussing examples we use named variables and not de Bruijn's indices. Nevertheless, our formal development uses a de Bruijn nameless notation of variables.

The above example also shows that in order to have a correct typing of the term

$$[x := z \cdot f := f](\lambda y{:}(T\ x).(f\ y))$$

in the context $\Gamma.\ f{:}(T\ z){\to}nat$, the expected typing judgment for substitution $[x := z \cdot f := f]$ is

$$\Gamma.\ f{:}(T\ z){\to}nat \vdash [x := z \cdot f := f] \triangleright \Gamma.\ x{:}nat.\ f{:}(T\ x){\to}nat$$

i.e. the type declared for f must be the type before the application of the substitution $[x := z]$.

We could imagine a typing rule of the form

$$\frac{\Gamma \vdash S \triangleright \Delta \quad \Gamma \vdash M : N}{\Gamma \vdash S \cdot M \triangleright \Delta.[S^{-1}]N}\ (\Pi\text{-Cons'})$$

where S^{-1} denotes the "inversion" of S, but since it is not always possible to invert a substitution, we prefer the following presentation of the rule:

$$\frac{\Gamma \vdash S \triangleright \Delta \quad \Gamma \vdash M : [S]N}{\Gamma \vdash S \cdot M \triangleright \Delta.N}\ (\Pi\text{-Cons"})$$

However note that when we use the rule (Π-Cons") in a bottom-up manner, e.g. in type inference algorithms, the type that is given to M, by the inductive application of the procedure, is $[S]N$ (up to conversion). To find $\Delta.N$, which is the type of $S \cdot M$, it is necessary to use a higher-order unification procedure!

Another drawback of (Π-Cons") is that it is not sound with respect to the usual typing properties. In particular, a substitution can be typed with two contexts that are not convertible, i.e. types are not unique modulo conversion. For example consider the context:

$$\Gamma = nil.\ nat{:}Type.\ T{:}nat{\to}Type.\ l{:}(\Pi n{:}nat.(T\ n)).\ 0{:}nat$$

and the valid typing judgment

$$\Gamma \vdash (l\ 0) : [n := 0](T\ n) \tag{3.6}$$

Since $[n := 0](T\ n)$ is convertible to $[n := 0](T\ 0)$, we also have:

$$\Gamma \vdash (l\ 0) : [n := 0](T\ 0) \tag{3.7}$$

Using (Π-Cons") with $\Gamma \vdash [n := 0] \triangleright \Gamma.\ n{:}nat$ and 3.6, we get

$$\Gamma \vdash [n := 0 \cdot y := (l\ 0)] \triangleright \Gamma.\ n{:}nat.\ y{:}(T\ 0)$$

and with $\Gamma \vdash [n := 0] \triangleright \Gamma.\ n{:}nat$ and 3.7:

$$\Gamma \vdash [n := 0 \cdot y := (l\ 0)] \triangleright \Gamma.\ n{:}nat.\ y{:}(T\ n)$$

Clearly, $(T\ 0)$ and $(T\ n)$ are not convertible.

To solve these problems, we use type annotations in substitutions, in a similar way as in the Church style of λ-calculus —as opposed to Curry style— binder variables in abstractions are annotated. The final version of (Π-Cons) has the form

$$\frac{\Gamma \vdash S \triangleright \Delta \quad \Gamma \vdash M : [S]N}{\Gamma \vdash S \cdot M_{:N} \triangleright \Delta.N}\ (\Pi\text{-Cons})$$

$(\lambda_{M'}.M\ N)$	$\longrightarrow$	$[\uparrow^0 \cdot N_{:M'}]M$	(Beta)
$[S](\lambda_M.N)$	$\longrightarrow$	$\lambda_{[S]M}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M}]N$	(Lambda)
$[S](\Pi_M.N)$	$\longrightarrow$	$\Pi_{[S]M}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M}]N$	(Pi)
$[S](M\ N)$	$\longrightarrow$	$([S]M\ [S]N)$	(Application)
$[T][S]M$	$\longrightarrow$	$[T \circ S]M$	(Clos)
$[S \cdot M_{:N}]\mathbf{1}$	$\longrightarrow$	M	(VarCons)
$[\uparrow^0]M$	$\longrightarrow$	M	(Id)
$T \circ (S \cdot M_{:N})$	$\longrightarrow$	$(T \circ S) \cdot [T]M_{:N}$	(Map)
$S \circ \uparrow^0$	$\longrightarrow$	S	(IdS)
$(S \cdot M_{:N}) \circ \uparrow^{Suc(n)}$	$\longrightarrow$	$S \circ \uparrow^n$	(ShiftCons)
$\uparrow^m \circ \uparrow^{Suc(n)}$	$\longrightarrow$	$\uparrow^{Suc(m)} \circ \uparrow^n$	(ShiftShift)
$\uparrow^{Suc(0)} \cdot \mathbf{1}_{:M}$	$\longrightarrow$	$\uparrow^0$	(Shift0)
$\uparrow^{Suc(n)} \cdot [\uparrow^n]\mathbf{1}_{:M}$	$\longrightarrow$	$\uparrow^n$	(ShiftS)
$[S]\mathit{Type}$	$\longrightarrow$	Type	(Type)

Figure 3.2: The $\lambda_{\mathcal{L}\Pi}$-rewrite system

The annotations in substitutions act as reminders of types, and they are introduced and maintained by the calculus of substitutions.

A different way proposed by Bloo in [8] is to introduce substitutions in context, and to give typing rules to deal with these extended contexts. This solution is similar to type systems with definitions [86, 9], where closures are typeable but substitutions are not considered as typeable objects.

We propose a type system where substitutions are first-class typeable expressions. We think that this feature is suitable in a λ-calculus with meta-variables. Before presenting the formal definition of the system with dependent types and explicit substitutions, we give some technical details about the calculus of substitutions with annotations. In Example 3.16 of Section 3.3 we redo completely the example developed at the beginning of this section by using type annotations.

3.2.3 A Calculus of Pre-Expressions

The set of well-formed expressions in $\lambda_{\mathcal{L}\Pi}$ is defined by the following grammar:

Naturals	n	::=	$0 \mid Suc(n)$
Pre-terms	M, N	::=	$\mathit{Kind} \mid \mathit{Type} \mid \mathbf{1} \mid \Pi_M.N \mid \lambda_M.N \mid (M\ N) \mid [S]M$
Pre-substitutions	S, T	::=	$\uparrow^n \mid S \cdot M_{:N} \mid T \circ S$

The $\lambda_{\mathcal{L}\Pi}$-calculus is given by the rewrite system of Fig. 3.2.

Semi-open expressions in $\lambda_{\mathcal{L}\Pi}$ are called *pre-expressions* to emphasize the fact that they contain type annotations in substitutions, but they are not necessarily well-typed expressions.

The $\mathcal{L}_\Pi$*-calculus* is obtained by dropping (Beta) from $\lambda_{\mathcal{L}\Pi}$.

Proposition 3.11 (Termination of $\mathcal{L}_\Pi$) *The $\mathcal{L}_\Pi$-calculus is terminating.*

Proof. The proof we present here is entirely due to H. Zantema [93], and it follows closely the proof of Lemma 2.36 in the previous chapter.

Termination of $\mathcal{L}_\Pi$ follows from termination of this one-sorted rewrite system:

$$
\begin{array}{l}
S \circ (\lambda_M.N) \longrightarrow \lambda_{[S]M}.(((Suc(0) \circ S) \cdot Suc(0)_{:M}) \circ N) \\
S \circ (\Pi_M.N) \longrightarrow \Pi_{[S]M}.(((Suc(0) \circ S) \cdot Suc(0)_{:M}) \circ N) \\
S \circ (M\ N) \longrightarrow (S \circ M\ S \circ N) \\
T \circ (S \circ M) \longrightarrow (T \circ S) \circ M \\
(S \cdot M_{:N}) \circ Suc(0) \longrightarrow M \\
0 \circ M \longrightarrow M \\
T \circ (S \cdot M_{:N}) \longrightarrow (T \circ S) \cdot (T \circ M)_{:N} \\
S \circ 0 \longrightarrow S \\
(S \cdot M_{:N}) \circ Suc(n) \longrightarrow S \circ n \\
m \circ Suc(n) \longrightarrow Suc(m) \circ n \\
Suc(0) \cdot Suc(0)_{:M} \longrightarrow 0 \\
Suc(n) \cdot (n \circ Suc(0))_{:M} \longrightarrow n \\
[S]\mathit{Type} \longrightarrow \mathit{Type}
\end{array}
$$

For $\mathcal{L}_\Pi$ the quasi-model is defined on the positive natural as follows:

$$
\begin{array}{lcl}
[\![\mathit{Type}]\!] & = & 1 \\
[\![0]\!] & = & 1 \\
[\![Suc(n)]\!] & = & [\![n]\!] \\
[\![\Pi_M.N]\!] & = & [\![M]\!] + [\![N]\!] \\
[\![\lambda_M.N]\!] & = & [\![M]\!] + [\![N]\!] \\
[\![(M\ N)]\!] & = & [\![M]\!] + [\![N]\!] \\
[\![T \circ S]\!] & = & [\![T]\!] \times [\![S]\!] \\
[\![S \cdot M_{:N}]\!] & = & \max([\![S]\!], [\![M]\!])
\end{array}
$$

In the same way as in Lemma 2.36, we verify that it is a quasi-model, and thus, according to Zantema's Theorem [92], it suffices to prove termination of the following infinite rewriting system:

$$
\begin{array}{l}
S \circ_{z\times(x+y)} (\lambda_M.N) \longrightarrow \lambda_{S\circ_{z\times x} M}.(((Suc(0) \circ_z S) \cdot Suc(0)_{:M}) \circ_{z\times y} N) \\
S \circ_{z\times(x+y)} (\Pi_M.N) \longrightarrow \Pi_{S\circ_{z\times x} M}.(((Suc(0) \circ_z S) \cdot Suc(0)_{:M}) \circ_{z\times y} N) \\
S \circ_{z\times(x+y)} (M\ N) \longrightarrow (S \circ_{z\times x} M)\ (S \circ_{z\times y} N) \\
T \circ_{z\times y\times x} (S \circ_{y\times x} M) \longrightarrow (T \circ_{z\times y} S) \circ_{z\times y\times x} M \\
(S \cdot M_{:N}) \circ_{\max(y,x)} Suc(0) \longrightarrow M \\
T \circ_{z\times\max(y,x)} (S \cdot M_{:N}) \longrightarrow (T \circ_{z\times y} S) \cdot (T \circ_{z\times x} M)_{:N} \\
0 \circ_x M \longrightarrow M \\
S \circ_x 0 \longrightarrow S \\
(S \cdot M_{:N}) \circ_{\max(y,x)\times z} Suc(n) \longrightarrow S \circ_{y\times z} n \\
m \circ_{y\times x} Suc(n) \longrightarrow Suc(m) \circ_{y\times x} n \\
Suc(0) \cdot Suc(0)_{:N} \longrightarrow 0 \\
Suc(n) \cdot (n \circ_x Suc(0))_{:N} \longrightarrow n \\
S \circ_x \mathit{Type} \longrightarrow \mathit{Type} \\
T \circ_i S \longrightarrow T \circ_j S
\end{array}
$$

for all strict positive naturals x, y, z, i and j, with $i > j$. Termination of this labeled system is proved by a recursive path order where $o_i > o_j$ for $i > j$, o_i is bigger than any other symbol, and o_i has a right-left lexicographic status.

Hence, $\mathcal{L}_\Pi$ is terminating. □

Corollary 3.12 *$\mathcal{L}_\Pi$-normal forms always exist.*

Annotated substitutions raise a technical problem. The $\lambda_{\mathcal{L}\Pi}$-calculus is not confluent on pre-expressions. The problem exists even if we only consider local confluence on ground terms. In fact, the following critical pair is not joinable in the general case, e.g. assume N and N' to be different ground $\lambda_{\mathcal{L}\Pi}$-normal forms:

$$\begin{array}{ccc} & (S \cdot M_{:N'}) \circ (\uparrow^{Suc(0)} \cdot \mathbf{1}_{:N}) & \\ \text{(Shift0;IdS)} \swarrow & & \searrow \text{(Map;VarCons;ShiftCons;IdS)} \\ S \cdot M_{:N'} & & S \cdot M_{:N} \end{array}$$

The same problem of non-confluence on pre-expressions arises in $\lambda\sigma$ with annotated substitutions, and in $\lambda\sigma_{\Uparrow}$ when the lift operator is annotated with typing information.

This problem is similar to that pointed out by Nederpelt for the λ-calculus extended with the η-rule. In that case, the confluence property holds on terms without type annotations (λ-calculus in Curry style), but does not on pre-terms (λ-calculus in Church style). Geuvers proposes in [32] a method to prove confluence for $\beta\eta$-reduction on typed λ-terms written in Church style. Geuvers' technique is based on confluence of the calculus without type annotations (i.e. in Curry style), and in a positive formulation of the Nederpelt's counter-example.

In Chapter 5 we show how to adapt Geuvers' technique to the $\lambda_{\mathcal{L}\Pi}$-calculus.

3.3 The Systems $\lambda\Pi_{\mathcal{L}}$ and $\mathrm{CC}_{\mathcal{L}}$

We present two dependent-typed versions of $\lambda_{\mathcal{L}\Pi}$: the system $\lambda\Pi_{\mathcal{L}}$ which is a formulation of $\lambda\Pi$ [41], and the system $\mathrm{CC}_{\mathcal{L}}$ which is a formulation of the Calculus of Constructions [14, 15]. These systems support explicit substitutions and typed meta-variables.

We recall that well-formed contexts and signatures are defined by the following grammar:

$$\begin{array}{lcll} \textbf{Contexts} & \Gamma & ::= & nil \mid \Gamma.M \\ \textbf{Signatures} & \Sigma & ::= & nil \mid \Sigma.\, X{:}_{\Gamma} M \end{array}$$

and typing assertions have one of the following forms:

- $\vdash \Sigma :\cdot nil$, the signature Σ is valid.
- $\vdash \Sigma :\cdot \Gamma$, the context Γ is valid in the signature Σ.
- $\Sigma :\cdot \Gamma \vdash M : N$, the term M has type N in $\Sigma :\cdot \Gamma$.
- $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, the substitution S has as type the context Δ in $\Sigma :\cdot \Gamma$.

Notation:

- We use $\vdash \Sigma$, $\vdash \Gamma$, $\Gamma \vdash M : N$, and $\Gamma \vdash S \triangleright \Delta$ as notations for $\vdash \Sigma :\cdot nil$, $\vdash nil :\cdot \Gamma$, $nil :\cdot \Gamma \vdash M : N$, and $nil :\cdot \Gamma \vdash S \triangleright \Delta$, respectively.
- We use the lowercase letter s to range over the set of sorts $\{Kind, Type\}$.
- The expression "$X \notin \Sigma$" means that the meta-variable X is not declared in Σ, i.e. there is no context Γ and term M such that $X :_\Gamma M \in \Sigma$.

Typing rules for signatures, contexts, terms and substitutions are all mutually dependent. We first give the rules to valid signatures and contexts.

Valid Signatures and Contexts

$$\frac{}{\vdash nil :\cdot nil}\ (\text{Empty}) \qquad \frac{\Sigma :\cdot \Gamma \vdash M : s}{\vdash \Sigma :\cdot \Gamma.M}\ (\text{Var-Decl})$$

$$\frac{\vdash \Sigma :\cdot \Gamma \quad X \notin \Sigma}{\vdash \Sigma.\ X :_\Gamma Kind}\ (\text{Metavar-Decl}_1) \qquad \frac{\Sigma :\cdot \Gamma \vdash M : s \quad X \notin \Sigma}{\vdash \Sigma.\ X :_\Gamma M}\ (\text{Metavar-Decl}_2)$$

Definition 3.13 *The* system $\lambda\Pi_{\mathcal{L}}$ *is defined by the following deductions rules:*

Valid Terms

$$\frac{\vdash \Sigma :\cdot \Gamma}{\Sigma :\cdot \Gamma \vdash Type : Kind}\ (\text{Type}) \qquad \frac{\Sigma :\cdot \Gamma \vdash M : Type \quad \Sigma :\cdot \Gamma.M \vdash N : s}{\Sigma :\cdot \Gamma \vdash \Pi_M.N : s}\ (\text{Prod})$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Delta \vdash M : Kind}{\Sigma :\cdot \Gamma \vdash [S]M : Kind}\ (\text{Clos-Kind}) \qquad \frac{\vdash \Sigma :\cdot \Gamma.M}{\Sigma :\cdot \Gamma.M \vdash \mathbf{1} : [\uparrow^{Suc(0)}]M}\ (\text{Var})$$

$$\frac{\Sigma :\cdot \Gamma \vdash M : \Pi_{M_1}.M_2 \quad \Sigma :\cdot \Gamma \vdash N : M_1}{\Sigma :\cdot \Gamma \vdash (M\ N) : [\uparrow^0 \cdot N_{:M_1}]M_2}\ (\text{Appl})$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Delta \vdash M : N \quad \Sigma :\cdot \Delta \vdash N : s}{\Sigma :\cdot \Gamma \vdash [S]M : [S]N}\ (\text{Clos})$$

$$\frac{\vdash \Sigma :\cdot \Gamma \quad X :_\Delta M \in \Sigma \quad \Delta =_{\lambda_{\mathcal{L}\Pi}} \Gamma}{\Sigma :\cdot \Gamma \vdash X : M}\ (\text{Metavar})$$

$$\frac{\Sigma :\cdot \Gamma \vdash M_1 : Type \quad \Sigma :\cdot \Gamma.M_1 \vdash M_2 : N \quad \Sigma :\cdot \Gamma \vdash \Pi_{M_1}.N : s}{\Sigma :\cdot \Gamma \vdash \lambda_{M_1}.M_2 : \Pi_{M_1}.N}\ (\text{Abs})$$

$$\frac{\Sigma :\cdot \Gamma \vdash M : M_1 \quad \Sigma :\cdot \Gamma \vdash M_2 : s \quad M_1 =_{\lambda_{\mathcal{L}\Pi}} M_2}{\Sigma :\cdot \Gamma \vdash M : M_2}\ (\text{Conv})$$

Valid Substitutions

$$\frac{\vdash \Sigma :\cdot \Gamma}{\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma}\ (\mathrm{Id}) \qquad \frac{\vdash \Sigma :\cdot \Gamma.M \quad \Sigma :\cdot \Gamma \vdash \uparrow^n \triangleright \Delta}{\Sigma :\cdot \Gamma.M \vdash \uparrow^{Suc(n)} \triangleright \Delta}\ (\mathrm{Shift})$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta' \quad \Sigma :\cdot \Delta' \vdash T \triangleright \Delta}{\Sigma :\cdot \Gamma \vdash S \circ T \triangleright \Delta}\ (\mathrm{Comp})$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Gamma \vdash M : [S]N \quad \Sigma :\cdot \Delta \vdash N : \mathit{Type}}{\Sigma :\cdot \Gamma \vdash S \cdot M_{:N} \triangleright \Delta.N}\ (\mathrm{Cons})$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \vdash \Sigma :\cdot \Delta' \quad \Delta =_{\lambda_{\mathcal{L}\Pi}} \Delta'}{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta'}\ (\mathrm{Conv\text{-}Subs})$$

Now, we extend the system $\lambda\Pi_{\mathcal{L}}$ with polymorphism and constructions of types.

Definition 3.14 *The* system $\mathrm{CC}_{\mathcal{L}}$ *is defined by the same deduction rules as* $\lambda\Pi_{\mathcal{L}}$ *except for (Prod), (Abs) and (Cons) which are re-defined as follows:*

$$\frac{\Sigma :\cdot \Gamma.M \vdash N : s}{\Sigma :\cdot \Gamma \vdash \Pi_M.N : s}\ (\mathrm{Prod})$$

$$\frac{\Sigma :\cdot \Gamma.M_1 \vdash M_2 : N \quad \Sigma :\cdot \Gamma \vdash \Pi_{M_1}.N : s}{\Sigma :\cdot \Gamma \vdash \lambda_{M_1}.M_2 : \Pi_{M_1}.N}\ (\mathrm{Abs})$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Gamma \vdash M : [S]N \quad \Sigma :\cdot \Delta \vdash N : s}{\Sigma :\cdot \Gamma \vdash S \cdot M_{:N} \triangleright \Delta.N}\ (\mathrm{Cons})$$

Remark 3.15 *Since there is no typing rule for Kind, for any signature and context the term Kind is not typeable, and so Kind does not occur as sub-term of a well-typed expression.*

Note that $\mathrm{CC}_{\mathcal{L}}$ and $\lambda\Pi_{\mathcal{L}}$ are both defined on the same calculus: the $\lambda_{\mathcal{L}\Pi}$-calculus. In other words, expressions in $\mathrm{CC}_{\mathcal{L}}$ and $\lambda\Pi_{\mathcal{L}}$ share the same abstract syntax.

Example 3.16 *Let* $\Gamma =$ *nil. nat:Type. T:nat*$\rightarrow$ *Type. z:nat, we verify in* $\lambda\Pi_{\mathcal{L}}$ *and* $\mathrm{CC}_{\mathcal{L}}$ *that*

$$\Gamma \vdash (\lambda x{:}nat.\lambda f{:}((T\ x)\rightarrow nat).\lambda y{:}(T\ x).(f\ y)\ z) : ((T\ z)\rightarrow nat)\rightarrow((T\ z)\rightarrow nat) \tag{3.8}$$

Reducing the (Beta)*-redex and distributing the substitution inside the abstraction, we get*

$$\begin{array}{ll} (\lambda x{:}nat.\lambda f{:}((T\ x)\rightarrow nat).\lambda y{:}(T\ x).(f\ y)\ z) & \xrightarrow{(\mathrm{Beta})} \\ [\uparrow^0 \cdot\, x := z_{:nat}](\lambda f{:}((T\ x)\rightarrow nat).\lambda y{:}(T\ x).(f\ y)) & \xrightarrow{\mathcal{L}_{\Pi}{}^*} \\ \lambda f{:}((T\ z)\rightarrow nat).[\uparrow^{Suc(0)} \cdot\, x := z_{:nat} \cdot\, f := f_{:(T\ x)\rightarrow nat}](\lambda y{:}(T\ x).(f\ y)) & \end{array}$$

According to the rule (Abs), the term $[\uparrow^{Suc(0)} \cdot\, x := z_{:nat} \cdot\, f := f_{:(T\ x)\rightarrow nat}](\lambda y{:}(T\ x).(f\ y))$ *must be typed in the context* $\Gamma.\ f{:}(T\ z)\rightarrow nat$*. We remark that the type annotation for* f *in the substitution*

corresponds to the type of f before the distribution of the substitution inside the abstraction (see the rule (Lambda) in Fig. 3.2).

We have also

$$\Gamma.\ f{:}(T\ z){\to}nat \vdash [\uparrow^{Suc(0)} \cdot x := z_{:nat} \cdot f := f_{:(T\ x){\to}nat}] \triangleright \Gamma.\ x{:}nat.\ f{:}(T\ x){\to}nat \tag{3.9}$$

Note that the typing rules for substitutions install the right context of variables. For example the variable declaration $f : (T\ z){\to}nat$ has been replaced by $f : (T\ x){\to}nat$ in the context where $\lambda y{:}(T\ x).(f\ y)$ will be typed.

Finally we verify

$$\Gamma.\ x{:}nat.\ f{:}(T\ x){\to}nat \vdash \lambda y{:}(T\ x).(f\ y) : (T\ x){\to}nat \tag{3.10}$$

hence, by the rule (Clos):

$$\Gamma.\ f{:}(T\ z){\to}nat \vdash [\uparrow^{Suc(0)} \cdot x := z_{:nat} \cdot f := f_{:(T\ x){\to}nat}](\lambda y{:}(T\ x).(f\ y)) : (T\ z){\to}nat \tag{3.11}$$

and by the rule (Abs):

$$\begin{array}{ll}\Gamma \vdash & \lambda f{:}((T\ z){\to}nat).[\uparrow^{Suc(0)} \cdot x := z_{:nat} \cdot f := f_{:(T\ x){\to}nat}](\lambda y{:}(T\ x).(f\ y)) : \\ & ((T\ z){\to}nat){\to}((T\ z){\to}nat)\end{array}$$

The above example is due to Geuvers and Bloo [34], and it happens to be a counter-example for subject reduction in calculi of explicit substitutions with dependent types where substitutions do not keep track of typing information. The use of annotated substitutions in $\lambda_{\mathcal{L}\Pi}$ preserves the right type when a substitution goes through an abstraction or a product. In fact, as we will show, subject reduction holds in both systems $\lambda\Pi_{\mathcal{L}}$ and $\mathrm{CC}_{\mathcal{L}}$.

In contrast to the simply-typed version of $\lambda_{\mathcal{L}}$, in $\mathrm{CC}_{\mathcal{L}}$ and $\lambda\Pi_{\mathcal{L}}$ the meta-variables can occur in contexts and signatures. Thus, we extend the notation $x\{X/M\}$ (where x is an arbitrary term or substitution) to contexts and signatures, i.e. $\Gamma\{X/M\}$ and $\Sigma\{X/M\}$, in the obvious way.

Definition 3.17 (Instantiation)

$$\begin{array}{lcl}
nil\{X/M\} & = & nil \\
(\Sigma.\ Y{:}_{\Gamma}N)\{X/M\} & = & \Sigma\{X/M\}.\ Y{:}_{\Gamma\{X/M\}}N\{X/M\} \\
(\Gamma.N)\{X/M\} & = & \Gamma\{X/M\}.N\{X/M\} \\
x\{X/M\} & = & x \quad \textit{if } x \in \{Kind, Type\} \textit{ or } x = \mathbf{1} \textit{ or } x = \uparrow^n \\
([S]M)\{X/M\} & = & [S\{X/M\}]M\{X/M\} \\
(\lambda_M.N)\{X/M\} & = & \lambda_{M\{X/M\}}.N\{X/M\} \\
(\Pi_M.N)\{X/M\} & = & \Pi_{M\{X/M\}}.N\{X/M\} \\
(M\ N)\{X/M\} & = & (M\{X/M\}\ N\{X/M\}) \\
X\{X/M\} & = & M \\
Y\{X/M\} & = & Y \quad \textit{if } X \neq Y \\
(T \circ S)\{X/M\} & = & T\{X/M\} \circ S\{X/M\} \\
(S \cdot M_{:N})\{X/M\} & = & S\{X/M\} \cdot M\{X/M\}_{:N\{X/M\}}
\end{array}$$

The following lemma states the conditions that guarantee the soundness of instantiation of meta-variables in $\mathrm{CC}_{\mathcal{L}}$ and $\lambda\Pi_{\mathcal{L}}$.

Lemma 3.18 (Instantiation Soundness) *Let M be a term such that $\Sigma' :\cdot \Gamma \vdash M : M_1$, and Σ a signature having the form $\Sigma'.\ X :_{\Gamma} M_1.\ \Sigma''$,*

1. *if $\vdash \Sigma :\cdot \Delta$, then $\vdash \Sigma'.\ \Sigma''\{X/M\} :\cdot \Delta\{X/M\}$,*
2. *if $\Sigma :\cdot \Delta \vdash M' : M_2$, then $\Sigma'.\ \Sigma''\{X/M\} :\cdot \Delta\{X/M\} \vdash M'\{X/M\} : M_2\{X/M\}$, and*
3. *if $\Sigma :\cdot \Delta \vdash S \triangleright \Delta'$, then $\Sigma'.\ \Sigma''\{X/M\} :\cdot \Delta\{X/M\} \vdash S\{X/M\} \triangleright \Delta'\{X/M\}$.*

Proof. By induction on typing derivation. □

Next property justifies the use of $\mathrm{CC}_{\mathcal{L}}$ (resp. $\lambda\Pi_{\mathcal{L}}$) to build proof-terms in CC_{dB} (resp. $\lambda\Pi_{\mathrm{dB}}$).

Definition 3.19 *A term M in $\lambda_{\mathcal{L}\Pi}$ is* pure *if M is a $\mathcal{L}_\Pi$-normal form of a ground term. A context Γ is pure if it is composed only of pure terms.*

Pure terms in $\lambda_{\mathcal{L}\Pi}$ are isomorphic to terms in λ_{dB} (with products and sorts) by taking $\mathbf{1} \equiv \underline{1}$ and $[\uparrow^{Suc(n)}]\mathbf{1} \equiv \underline{n+1}$. Thus, for the sake of simplicity, we identify pure terms in $\lambda_{\mathcal{L}\Pi}$ with terms in λ_{dB}.

Theorem 3.20 (Typing Soundness with Respect to λ_{dB}) *Let M, N be pure terms in $\lambda_{\mathcal{L}\Pi}$ and Γ a pure context, $\Gamma \vdash M : N$ in $\mathrm{CC}_{\mathcal{L}}$ (resp. $\lambda\Pi_{\mathcal{L}}$) if and only if $\Gamma \vdash_{\mathrm{dB}} M : N$ in CC_{dB} (resp. $\lambda\Pi_{\mathrm{dB}}$).*

Proof. First, we show by induction on n that $\Gamma \vdash \underline{n} : N$ in $\mathrm{CC}_{\mathcal{L}}$ (resp. $\lambda\Pi_{\mathcal{L}}$) if and only if $\Gamma \vdash_{\mathrm{dB}} \underline{n} : N$ in CC_{dB} (resp. $\lambda\Pi_{\mathrm{dB}}$). The other cases are proved by induction on typing derivation. □

We will show in Part 2 that $\mathrm{CC}_{\mathcal{L}}$ enjoys the following properties (see [69] for a discussion about the proofs of the system $\lambda\Pi_{\mathcal{L}}$):

Proposition (Sort Soundness)

1. *If $\Sigma :\cdot \Gamma \vdash M : N$, then either $N = Kind$, or $\Sigma :\cdot \Gamma \vdash N : s$, and*
2. *if $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, then $\vdash \Sigma :\cdot \Delta$.*

Proposition (Type Uniqueness) *Let Γ_1 and Γ_2 be such that $\Gamma_1 =_{\lambda_{\mathcal{L}\Pi}} \Gamma_2$,*

1. *if $\Sigma :\cdot \Gamma_1 \vdash M : N_1$ and $\Sigma :\cdot \Gamma_2 \vdash M : N_2$, then $N_1 =_{\lambda_{\mathcal{L}\Pi}} N_2$, and*
2. *if $\Sigma :\cdot \Gamma_1 \vdash S \triangleright \Delta_1$ and $\Sigma :\cdot \Gamma_2 \vdash S \triangleright \Delta_2$, then $\Delta_1 =_{\lambda_{\mathcal{L}\Pi}} \Delta_2$.*

Proposition (Subject Reduction) *If $x \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} y$, then*

1. *if x is a term and $\Sigma :\cdot \Gamma \vdash x : N$, then $\Sigma :\cdot \Gamma \vdash y : N$, and*
2. *if x is a substitution and $\Sigma :\cdot \Gamma \vdash x \triangleright \Delta$, then $\Sigma :\cdot \Gamma \vdash y \triangleright \Delta$.*

Proposition (Typing Soundness)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, $\Sigma :\cdot \Gamma \vdash M' : N'$ *and* $M =_{\lambda_{\mathcal{L}\Pi}} M'$, *then there exists a path of well-typed terms to convert* M *and* M', *and*

2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, $\Sigma :\cdot \Gamma \vdash S' \triangleright \Delta$ *and* $S =_{\lambda_{\mathcal{L}\Pi}} S'$, *then there exists a path of well-typed substitutions to convert* S *and* S'.

In contrast to $\lambda\Pi$ and the Calculus of Constructions, the systems $\lambda\Pi_{\mathcal{L}}$ and $\mathrm{CC}_{\mathcal{L}}$ do not preserve strong normalization. In fact, the counter-example of the simply-typed version of $\lambda_{\mathcal{L}}$ can be reproduced in the dependent-typed versions of $\lambda_{\mathcal{L}\Pi}$. Nevertheless, we prove in Chapter 7 the weak normalization property of $\lambda\Pi_{\mathcal{L}}$ and $\mathrm{CC}_{\mathcal{L}}$.

Proposition (Weak Normalization)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, *then* M *is weakly normalizing, and*

2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* S *is weakly normalizing.*

Therefore, M *and* S *have at least one* $\lambda_{\mathcal{L}\Pi}$*-normal form.*

Proposition (Church-Rosser) *Let* x *and* y *be such that* $x =_{\lambda_{\mathcal{L}\Pi}} y$. *Then,* x *and* y *are* $\lambda_{\mathcal{L}\Pi}$*-joinable, i.e. there exists* w *such that* $x \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} w$ *and* $y \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} w$, *if*

1. x *is a term,* $\Sigma :\cdot \Gamma_1 \vdash x : N_1$ *and* $\Sigma :\cdot \Gamma_2 \vdash y : N_2$, *or*

2. x *is a substitution,* $\Sigma :\cdot \Gamma_1 \vdash x \triangleright \Delta_1$, $\Sigma :\cdot \Gamma_2 \vdash y \triangleright \Delta_2$ *and* $\Delta_1 =_{\lambda_{\mathcal{L}\Pi}} \Delta_2$.

3.3.1 The Case of (SCons) in $\lambda\sigma$

When we consider annotated substitutions, the system $\lambda\sigma$ may lose the subject reduction property in a dependent type theory due to the non left-linear rule (SCons).

Take the context

$$\Gamma = \mathit{nil}.\ \mathit{nat}{:}\mathit{Type}.\ T{:}\mathit{nat}{\to}\mathit{Type}.\ l{:}(\Pi n{:}\mathit{nat}.(T\ n)).\ 0{:}\mathit{nat}.\ m{:}(T\ 0){\to}\mathit{nat}$$

and the substitution

$$S = (x := 0_{:\mathit{nat}} \cdot y := (l\ 0)_{:(T\ 0)})$$

We verify that the following typing judgments are valid:

$$\Gamma \vdash S \triangleright \Gamma.\ x{:}\mathit{nat}.\ y{:}(T\ 0) \tag{3.12}$$

$$\Gamma \vdash (S \circ \uparrow) \cdot [S]\mathbf{1}_{:(T\ x)} \triangleright \Gamma.\ x{:}\mathit{nat}.\ y{:}(T\ x) \tag{3.13}$$

But also,

$$(S \circ \uparrow) \cdot [S]\mathbf{1}_{:(T\ x)} \xrightarrow{\text{(SCons)}} S$$

However, since $(T\ 0)$ and $(T\ x)$ are not convertible,

$$\Gamma \not\vdash S \triangleright \Gamma.\ x{:}\mathit{nat}.\ y{:}(T\ x)$$

The problem here is not with the typing system but with the substitution calculus. Non-left-linear rules —like (SCons)— are not only harmful for typing, but also they are usually responsible of non-confluence problems [56]. This remark was our main motivation for developing the $\lambda_{\mathcal{L}}$-calculus. As we have seen, the $\lambda_{\mathcal{L}}$-calculus is a left-linear variant of $\lambda\sigma$ and it does not have the rule (SCons).

3.4 Related Works and Summary

Some type theories extended with explicit substitutions have been proposed: The Simple Type Theory [1, 58, 25, 50, 18], the Second-Order Type Theory [1], the Martin Löf Type Theory [88], the Calculus of Constructions [82] and Pure Type Systems [8]. Except for the simply-typed version of $\lambda\sigma$ in [25], neither of them consider terms with meta-variables as first-class objects.

Sometimes, explicit substitutions are identified to the `let-in` constructor of functional ML-style programming languages. Both mechanisms allow to have delayed applications of substitutions to terms. For example, $\texttt{let}\ x := 0\ \texttt{in}\ \lambda y{:}A.x$ will be unfolded in $\lambda y{:}A.0$, in the same way as $[x := 0](\lambda y{:}A.x)$ reduces to $\lambda y{:}A.0$. In their simply-typed versions, explicit substitutions and `let-in` constructors have similar behaviors. However, in dependent type systems the relationship between both mechanisms is not immediate.

To illustrate that, let us take the typing rule for closures —explicit applications of substitutions to terms— in a dependent type system:

$$\frac{\Gamma \vdash S \triangleright \Delta \quad \Delta \vdash M : N}{\Gamma \vdash [S]M : [S]N}\ (\Pi\text{-Clos})$$

Consider the context:

$$\Gamma = \mathit{nil}.\ \mathit{nat}{:}\mathit{Type}.\ T{:}\mathit{nat}{\to}\mathit{Type}.\ l{:}(\Pi n{:}\mathit{nat}.(T\ n)).\ 0{:}\mathit{nat}.\ m{:}(T\ 0){\to}\mathit{nat}$$

Using the above typing rule, the term $[x := 0](m\ (l\ x))$ is ill-typed. This is because the information that the variable x will be substituted by 0 in the term $(m\ (l\ x))$ is not taken into account by the rule (Π-Clos). Therefore, the type of $(l\ x)$ is $(T\ x)$, not $(T\ 0)$ as expected by m. On the other hand, the same term can be written using the `let-in` notation as $\texttt{let}\ x := 0\ \texttt{in}\ (m\ (l\ x))$. This term is well-typed because x has the value 0 in $(m\ (l\ x))$, and thus $\texttt{let}\ x := 0\ \texttt{in}\ (m\ (l\ x))$ is typed as if it was $(m\ (l\ 0))$. However, to unfold definitions before typing is not sufficient when we admit meta-variables in λ-terms since substitutions and meta-variables may appear in normal forms. In this case we cannot escape to have a (Π-Clos)'s like rule.

The approach we have taken is to consider explicit substitutions different from the `let-in` mechanism. Explicit substitutions is a syntactic feature to allow substitutions to be part of the formal language by means of special constructors and reduction rules. In this way, the term $[x := 0](m\ (l\ x))$ is ill-typed, just as the term $(\lambda x{:}\mathit{nat}.(m\ (l\ x))\ 0)$ is. The `let-in` structure has a more complex behavior. It provides a mechanism for definitions in the language. Formal presentations of type systems with definitions are given in [86, 9].

In this chapter we have proposed three type systems: simple types ($\lambda_{\mathcal{L}}$), Dependent Types ($\lambda\Pi_{\mathcal{L}}$), and the Calculus of Constructions ($\mathrm{CC}_{\mathcal{L}}$), all of them with explicit substitutions and typed meta-variables. These type systems enjoy the usual typing properties: type uniqueness, subject reduction, weak normalization and confluence. Before discussing the proof of these properties, we conclude the first part of this thesis with the main motivations of our calculus: applications to incomplete proof representation and to proof synthesis.

Chapter 4

An Application to Proof Synthesis

4.1 Proof Synthesis

Automatic proof synthesis is at the basis of proof assistant systems. For the first order logic, and for the higher-order logic, general complete methods for search of proof-trees are based on resolution and unifications algorithms [84, 45].

In the proofs-as-terms paradigm, a method of proof synthesis is just a method to find a term of a given type. Dowek shows in [23, 24] that resolution and unification can be merged, and generalized to the type systems of the Barendregt's cube, in a term enumeration algorithm.

Since the set of λ-terms is enumerable, a complete method of proof synthesis in any constructive logic where type-checking is decidable consists to enumerate all the terms, and for each one to check if it has the right type. Of course this method is inpractical for implementations.

A smart enumeration of terms take typing information into account. For example, since well-typed terms have normal forms, we can enumerate normal forms only. Also, terms can be built incrementally according to their types.

The method we propose is strongly inspired by that of Proof Synthesis for the Cube of Type Systems presented in [23, 24]. However, we use meta-variables to represent holes in terms, and instantiation of meta-variables as the filling mechanism. In this way, terms can be build incrementally. As we have said in Chapter 1, explicit substitutions are necessary to deal correctly with typing and reduction requirements.

4.1.1 An Example with Simple Types

A possible proof-term of $A{\to}((A{\to}B){\to}B)$ in the empty context has the form $\lambda x{:}A.\lambda y{:}A{\to}B.Y_1$, where Y_1 is an arbitrary term of type B in the context $\Gamma = nil.\ x{:}A.\ y{:}A{\to}B$. Now, we can proceed recursively on Y_1. A possible term of type B in Γ has the form $(y\ Y_2)$, where Y_2 is an arbitrary term of type A in Γ; for instance x.

In our formalism, the terms Y_1 and Y_2 are meta-variables, and to find a proof-term of

$$A{\to}((A{\to}B){\to}B)$$

is merely to find an instantiation of a meta-variable Y, such that

$$\Gamma \vdash Y : A{\to}((A{\to}B){\to}B)$$

We verify that

$$Y\{Y/\lambda x{:}A.\lambda y{:}A{\rightarrow}B.Y_1\}\{Y_1/(y\ Y_2)\}\{Y_2/x\} = \lambda x{:}A.\lambda y{:}A{\rightarrow}B.(y\ x)$$

is a ground term in normal form of the right type.

4.1.2 An Example with Dependent Types

Let $\Gamma = nil.\ nat{:}Type.\ {\geq}{:}nat{\rightarrow}(nat{\rightarrow}Type).\ 0{:}nat$. We search an instantiation for a meta-variable Y such that

$$\Gamma \vdash Y : \Pi p{:}(\Pi h{:}nat.(h \geq 0)).(0 \geq 0)$$

A possible instantiation for Y has the form $\lambda p{:}(\Pi h{:}nat.(h \geq 0)).Y_1$, where

$$\Gamma.\ p{:}(\Pi h{:}nat.(h \geq 0)) \vdash Y_1 : (0 \geq 0)$$

We suspect that an instantiation for Y_1 has the form $(p\ Y_2)$ where

$$\Gamma.\ p{:}(\Pi h{:}nat.(h \geq 0)) \vdash Y_2 : nat$$

However, we cannot instantiate Y_1 with $(p\ Y_2)$ because Y_1 has type $(0 \geq 0)$, while $(p\ Y_2)$ has type $(Y_2 \geq 0)$. In order to guarantee the soundness of the construction, we must solve the constraint $(0 \geq 0) = (Y_2 \geq 0)$. In this case, the constraint is satisfied if we instantiate Y_2 with 0.

We verify that

$$Y\{Y/\lambda p{:}(\Pi h{:}nat.(h \geq 0)).Y_1\}\{Y_1/(p\ Y_2)\}\{Y_2/0\} = \lambda p{:}(\Pi h{:}nat.(h \geq 0)).(p\ 0)$$

is a ground term in normal form of the right type.

4.1.3 An Example with Polymorphism

Let $\Gamma = nil.\ nat{:}Type.\ 0{:}nat.\ pol{:}(\Pi t{:}Type.t)$. We search an instantiation for a meta-variable Y such that

$$\Gamma \vdash Y : nat$$

A possible instantiation for Y is $\{Y/0\}$. However, since *pol* is a type constructor, we can built another instantiation by instantiating Y with $(pol\ Y_1\ Y_2)$. In order to guarantee the soundness of the construction we must satisfy the constraints $Y_1 = \Pi y{:}H_1.K_1$, $[y := Y_{2:H_1}]K_1 = nat$, and

$$\Gamma \vdash Y_1 : Type$$

$$\Gamma \vdash Y_2 : H_1$$

$$\Gamma \vdash H_1 : Type$$

$$\Gamma.\ y{:}H_1 \vdash K_1 : Type$$

We can instantiate H_1 with *nat*. The new problem has the constraints $Y_1 = \Pi y{:}nat.K_1$, $[y := Y_{2:nat}]K_1 = nat$, and

$$\Gamma \vdash Y_1 : Type$$

$$\Gamma \vdash Y_2 : nat$$

$$\Gamma.\ y{:}nat \vdash K_1 : Type$$

We can instantiate K_1 with nat. The new problem has the constraints $Y_1 = \Pi y{:}nat.nat$, and

$$\Gamma \vdash Y_1 : Type$$

$$\Gamma \vdash Y_2 : nat$$

Now, we can instantiate Y_1 with $\Pi y{:}nat.nat$ to solve the first constraint. Finally, we can instantiate Y_2 with 0.

We verify that

$$Y\{Y/(pol\ Y_1\ Y_2)\}\{Y_1/\Pi y{:}nat.nat\}\{Y_2/0\} = (pol\ (\Pi y{:}nat.nat)\ 0)$$

is a ground term in normal form of type nat.

Notice that the arity of pol depends on its first argument. In fact, there are infinite ground instantiations for Y with the form

$$\{Y/(pol\ (\Pi y_1{:}nat.\ldots.\Pi y_k{:}nat.nat)\ \overbrace{0\ \ldots\ 0}^{k-\text{times}})\}$$

The above examples show that a complete term enumeration algorithm for dependent types must solve constraints of equivalence of terms, and in the case of polymorphism and constructions of types, it must deal with terms of variable arity (this problem is usually called splitting [23]).

4.2 A Type System with Constraints

Definition 4.1 (Constraints and Constrained Signatures) *A* constraint *is a triple* (M, N, Γ), *denoted by* $M =^?_\Gamma N$, *relating two terms* M, N *and one context* Γ. *A* constrained signature *is a list containing meta-variable declarations —just as in Def. 3.10— and constraint declarations. Formally, they are defined by the following grammar:*

$$\textbf{Constrained Signatures}\quad \Sigma_c \quad ::= \quad nil \mid \Sigma_c.\ X{:}_\Gamma M \mid \Sigma_c.\ M =^?_\Gamma N$$

Definition 4.2 (Equivalence Modulo Constraints) *Let* Σ_c *be a constrained signature, we define the relation* $\equiv_{\Sigma_c}$ *as the smallest equivalence relation compatible with structure such that*

- *if* $M =_{\lambda_{\mathcal{L}\Pi}} N$, *then* $M \equiv_{\Sigma_c} N$, *and*
- *if* $M =^?_\Gamma N \in \Sigma_c$, *then* $M \equiv_{\Sigma_c} N$.

Definition 4.3 *The type system with constraints, namely* CC_c, *is defined as* $\mathrm{CC}_\mathcal{L}$ *in Def. 3.14 (Section 3.3), but we denote typing judgements by* $\Sigma_c :\cdot \Gamma \vdash_c M : N$ *and* $\Sigma_c :\cdot \Gamma \vdash_c S \triangleright \Delta$, *we replace signatures by constrained signatures, we add the rule*

$$\frac{\Sigma_c :\cdot \Gamma \vdash M : N \quad \Sigma_c :\cdot \Gamma \vdash M' : N}{\vdash_c \Sigma_c.\ M =^?_\Gamma M'}\ (\text{Constraint})$$

and we replace the rule (Conv) *by*

$$\frac{\Sigma_c :\cdot \Gamma \vdash M : M_1 \quad \Sigma_c :\cdot \Gamma \vdash M_2 : s \quad M_1 \equiv_{\Sigma_c} M_2}{\Sigma_c :\cdot \Gamma \vdash M : M_2}\ (\text{Conv})$$

Definition 4.4 *We say that* Σ_c *is a* valid constrained signature *if it holds that* $\vdash_c \Sigma_c$.

Lemma 4.5 *If we denote the prefix relation of lists by* $\preceq$, *then the following typing rules are admissible in* CC_c.

$$\frac{\vdash_c \Sigma_c :\cdot \Delta \quad \Gamma \preceq \Delta}{\vdash_c \Sigma_c :\cdot \Gamma}\ (\text{Undeclare-Var}) \qquad \frac{\vdash_c \Sigma_c' \quad \Sigma_c \preceq \Sigma_c'}{\vdash_c \Sigma_c}\ (\text{Undeclare-Metavar})$$

Proof. By structural induction on Δ and Σ_c'. □

Notation: If Σ_c is a constrained signature, we denote by $(\Sigma_c)^{-c}$ the signature obtained by dropping the constraints of Σ_c.

Lemma 4.6 *Let* Σ_c *be a valid constrained signature, and* $\Sigma = (\Sigma_c)^{-c}$,

- 1. *if* $\vdash \Sigma :\cdot \Gamma$, *then* $\vdash_c \Sigma_c :\cdot \Gamma$,
  2. *if* $\Sigma :\cdot \Gamma \vdash M : N$, *then* $\Sigma_c :\cdot \Gamma \vdash_c M : N$, *and*
  3. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* $\Sigma_c :\cdot \Gamma \vdash_c S \triangleright \Delta$,
- *and if* Σ_c *does not have constraints, i.e.* $\Sigma = \Sigma_c$,
  1. *if* $\vdash_c \Sigma_c :\cdot \Gamma$, *then* $\vdash \Sigma_c :\cdot \Gamma$, *then*
  2. *if* $\Sigma_c :\cdot \Gamma \vdash_c M : N$, *then* $\Sigma_c :\cdot \Gamma \vdash M : N$, *and*
  3. *if* $\Sigma_c :\cdot \Gamma \vdash_c S \triangleright \Delta$, *then* $\Sigma_c :\cdot \Gamma \vdash S \triangleright \Delta$.

Proof. By simultaneous induction on typing derivations. □

Definition 4.7 (Normal Form of a Signature) *Let* Σ_c *be a constrained signature, the* normal form *of* Σ_c, *denoted by* $(\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}}$, *is obtained by the application of the following rules:*

$$\Sigma_c'.\ X :_\Gamma M.\ \Sigma_c'' \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma_c'.\ X :_\Gamma (M)\downarrow_{\lambda_{\mathcal{L}\Pi}}.\ \Sigma_c'' \qquad \text{if } (\Sigma_c')^{-c} :\cdot \Gamma \vdash X : M \text{ and } M \text{ is not a } \lambda_{\mathcal{L}\Pi}\text{-normal form}$$

$$\Sigma_c'.\ M =^?_\Gamma N.\ \Sigma_c'' \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma_c'.\ (M)\downarrow_{\lambda_{\mathcal{L}\Pi}} =^?_\Gamma N.\ \Sigma_c'' \qquad \text{if } (\Sigma_c')^{-c} :\cdot \Gamma \vdash M : M' \text{ and } M \text{ is not a } \lambda_{\mathcal{L}\Pi}\text{-normal form}$$

$$\Sigma_c'.\ M =^?_\Gamma N.\ \Sigma_c'' \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma_c'.\ M =^?_\Gamma (N)\downarrow_{\lambda_{\mathcal{L}\Pi}}.\ \Sigma_c'' \qquad \text{if } (\Sigma_c')^{-c} :\cdot \Gamma \vdash N : M' \text{ and } N \text{ is not a } \lambda_{\mathcal{L}\Pi}\text{-normal form}$$

$$\Sigma_c'.\ M =^?_\Gamma M.\ \Sigma_c'' \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma_c'.\ \Sigma_c''$$

Remark 4.8 *The above definition is right since the* $\lambda_{\mathcal{L}\Pi}$*-normal forms of well-typed expressions in* $\mathrm{CC}_{\mathcal{L}}$ *always exist and they are unique (for details see Theorem 7.62 and Corollary 8.3 in Chapter 7 and Chapter 8, respectively).*

Lemma 4.9 *Let* Σ_c *be a valid constrained signature,*

1. $\vdash_c \Sigma_c :\cdot \Gamma$ *if and only if* $\vdash_c (\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma$,
2. $\Sigma_c :\cdot \Gamma \vdash M : N$ *if and only if* $(\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma \vdash_c M : N$, *and*
3. $\Sigma_c :\cdot \Gamma \vdash S \triangleright \Delta$ *if and only if* $(\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma \vdash_c S \triangleright \Delta$.

Proof. By simultaneous induction on typing derivation in CC_c. □

Corollary 4.10 *If* $\vdash_c \Sigma_c$, *then* $\vdash_c (\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}}$.

We recall that an expression x is said to be *ground* if it has no occurrence of meta-variables.

Definition 4.11 (Failure Signature) *Let* Σ_c *be the normal form of a valid constrained signature, we say that* Σ_c *is a* failure signature *if it contains a constraint relating two ground terms in* $\lambda_{\mathcal{L}\Pi}$*-normal form which are not identical.*

Lemma 4.12 *Let* Σ_c *be the normal form of a valid constrained signature such that* $\Sigma_c \neq nil$ *and* Σ_c *is not a failure signature. Then,* Σ_c *has the form* $\Sigma'_c.\ X{:}_\Gamma N.\ \Sigma''_c$, *where*

- Σ'_c *does not have constraints,*
- $\vdash \Sigma'_c.\ X{:}_\Gamma N$, *and*
- N *is a* $\lambda_{\mathcal{L}\Pi}$*-normal form.*

Proof. Since Σ_c is not empty, then it has at least one element. Assume that the first element is a constraint $M =^?_\Gamma M'$. By hypothesis and Corollary 4.10, we have $\vdash_c \Sigma_c$. By Lemma 4.5, it happens that $\vdash_c nil.\ M =^?_\Gamma M'$. The last derivation of this judgment in CC_c has the premises $\Gamma \vdash_c M : N'$ and $\Gamma \vdash_c M' : N'$. Since M, M', N' are well-typed in CC_c without meta-variables, they are ground, and by Lemma 4.6, it holds that $\Gamma \vdash M : N'$ and $\Gamma \vdash M' : N'$. Since Σ_c is a signature in normal form, M and N' are not identical. But this is not possible because Σ_c is not a failure context.

Therefore, the first element of Σ_c is not a constraint, and so, Σ_c has the form $\Sigma'_c.\ X{:}_\Gamma N.\ \Sigma''_c$, where Σ'_c does not have constraints. By Lemma 4.5, we have $\vdash_c \Sigma'_c.\ X{:}_\Gamma N$, and thus, by Lemma 4.6, $\vdash \Sigma'_c.\ X{:}_\Gamma N$. Since Σ_c is a signature in normal form, N is a $\lambda_{\mathcal{L}\Pi}$-normal form. □

Definition 4.13 (Rigid Terms) *The set* $\mathcal{R}^n_i$ *of* atomic rigid terms of arity i and header n, *where* $i, n \geq 0$, *is defined as follows:*

$$\begin{array}{lcl} \mathcal{R}^0_0 & = & \{\mathit{Kind}, \mathit{Type}\} \\ \mathcal{R}^n_i & = & (\underline{n}\ M_1\ \ldots\ M_i) \quad \textit{where } n \geq 1 \end{array}$$

A term M *is* rigid, *denoted by* $M \in \mathcal{R}$, *if and only if* M *is an abstraction or* M *is a product, or* $M \in \mathcal{R}^n_i$ *for some* $i, n \geq 0$.

Definition 4.14 (Flexible Terms) *The term* M *is said to be* flexible of arity (i, n) and header X, *where* $i, n \geq 0$ *and* X *is a meta-variable, if and only if it has the form* $(M'\ M'_1\ \ldots\ M'_m)$ *with* $m \geq 0$, *and*

- *if* $i = 0$ *and* $n = 0$, *then* $M' = X$,

- *if* $i = 0$ *and* $n = Suc(n')$, *then* M' *has the form* $[\uparrow^n]X$, *and*
- *if* $i > 0$, *then* M' *has the form* $[\uparrow^n \cdot M_{i:N_i} \cdot \ldots \cdot M_{1:N_1}]X$.

In this case we write $M \in \mathcal{F}_X^{(i,n)}$.

Lemma 4.15 *If* $\Sigma :\cdot \Gamma \vdash M : N$ *and* N *is a* $\lambda_{\mathcal{L}\Pi}$*-normal form, then* N *has the form* $\Pi_{N_1} \ldots . \Pi_{N_i} . N'$, *where* $i \geq 0$, *and either* $N' \in \mathcal{R}_j^n$ *for some* $j, n \geq 0$, *or* $N' \in \mathcal{F}_X^{(j,n)}$ *for some* $j, n \geq 0$ *and meta-variable* X.

Proof. By the sort soundness property (Theorem 6.12 in Chapter 6), either $N = Kind$ or $\Sigma :\cdot \Gamma \vdash N : s$. The case $N = Kind$ is trivial because $Kind \in \mathcal{R}_0^0$. In the other case, we proceed by case analysis on N. □

4.3 Graftings, Derivations and Solutions

As we have seen in Section 4.1.2 and Section 4.1.3, instantiation of meta-variables in dependent type systems, may need the declaration of new meta-variables and constraints. Thus, we generalize the instantiation mechanism in a *grafting* mechanism.

Definition 4.16 (Graftings) *A* grafting θ_c *is a triple* (X, M, Σ_c), *denoted by* $\{X/_{\Sigma_c} M\}$, *where* X *is a meta-variable,* M *is a term and* Σ_c *is a constrained signature.*

Notation: If $\theta_c = \{X/_{\Sigma_c} M\}$, we denote by $(\theta_c)^{-c}$ the instantiation $\{X/M\}$.

Definition 4.17 (Valid Graftings) *Let* Σ_c *be a valid constrained signature, we say that* $\{X/_{\Sigma'_c} M\}$ *is a* valid grafting of Σ_c, *if and only if*

- *the meta-variables declared in* Σ'_c *are not in* Σ_c, *and*
- Σ_c *has the form* $\Sigma_{c_1} .\ X :_\Gamma N .\ \Sigma_{c_2}$ *where* $\Sigma_{c_1} .\ \Sigma'_c :\cdot \Gamma \vdash_c M : N$.

Definition 4.18 *Let* $\theta_c = \{X/_{\Sigma'_c} M\}$ *be a valid grafting of* $\Sigma_c = \Sigma_{c_1} .\ X :_\Gamma N .\ \Sigma_{c_2}$, *the* application of θ_c to Σ_c, *denoted by* $\Sigma_c\theta_c$, *is defined as:*

$$\Sigma_c\theta_c = \Sigma_{c_1} .\ \Sigma'_c .\ \Sigma_{c_2}\{X/M\}$$

where $\Sigma_{c_2}\{X/M\}$ *is defined as in Def. 3.17 with the additional clause to deal with constraints:*

$$(\Sigma_c .\ N =^?_\Gamma N')\{X/M\} \ = \ \Sigma_c\{X/M\} .\ N\{X/M\} =^?_{\Gamma\{X/M\}} N'\{X/M\}$$

Lemma 4.19 *Let* θ_c *be a valid grafting of a constrained signature* Σ_c, *and* $\theta = (\theta_c)^{-c}$,

- *if* $\vdash_c \Sigma_c :\cdot \Gamma$, *then* $\vdash_c \Sigma_c\theta_c :\cdot \Gamma\theta$,
- *if* $\Sigma_c :\cdot \Gamma \vdash_c M : N$, *then* $\Sigma_c\theta_c :\cdot \Gamma\theta \vdash_c M\theta : N\theta$, *and*
- *if* $\Sigma_c :\cdot \Gamma \vdash_c S \triangleright \Delta$, *then* $\Sigma_c\theta_c :\cdot \Gamma\theta \vdash_c S\theta \triangleright \Delta\theta$.

Proof. By induction on typing derivations. We use Def. 4.17, Def. 4.18, and Lemma 3.18. □

Definition 4.20 (Sequential Graftings) *Let Σ_c be a valid constrained signature, we say that a list ψ of graftings is a* sequential grafting of Σ_c *if and only if*

- *ψ is the empty list, or*
- *$\psi = \theta_c, \psi'$, where θ_c is a valid grafting of Σ_c, and ψ' is a sequential grafting of $\Sigma_c\theta_c$.*

The application of a sequential grafting ψ to Σ_c, *denoted by $\Sigma_c\psi$, consists in the sequential application of graftings in ψ to Σ_c. It is inductively defined as follows:*

$$\Sigma_c\psi \;=\; \begin{cases} \Sigma_c & \text{if } \psi \text{ is the empty list} \\ (\Sigma_c\theta_c)\psi' & \text{if } \psi = \theta_c, \psi' \end{cases}$$

The application of a sequential grafting ψ to a context Γ and to an expression x, denoted by $\Gamma\psi$ and $x\psi$ respectively, is defined as follows:

$$\Gamma\psi = \begin{cases} \Gamma & \text{if } \psi \text{ is the empty list} \\ (\Gamma\theta)\psi' & \text{if } \psi = \theta_c, \psi' \text{ and } \theta = (\theta_c)^{-c} \end{cases} \qquad x\psi = \begin{cases} x & \text{if } \psi \text{ is the empty list} \\ (x\theta)\psi' & \text{if } \psi = \theta_c, \psi' \text{ and } \theta = (\theta_c)^{-c} \end{cases}$$

We show some properties of valid graftings.

Lemma 4.21 *Let ψ be a sequential grafting of a constrained signature Σ_c,*

- *if $\vdash_c \Sigma_c :\cdot \Gamma$, then $\vdash_c \Sigma_c\psi :\cdot \Gamma\psi$,*
- *if $\Sigma_c :\cdot \Gamma \vdash_c N : N'$, then $\Sigma_c\psi :\cdot \Gamma\psi \vdash_c N\psi : N'\psi$, and*
- *if $\Sigma_c :\cdot \Gamma \vdash_c S \triangleright \Delta$, then $\Sigma_c\psi :\cdot \Gamma\psi \vdash_c S\psi \triangleright \Delta\psi$.*

Proof. We reason by induction on the length of the list ψ. In the induction step we use Lemma 4.19. □

Lemma 4.22 *Let Σ_c be a valid constrained signature, θ_c is a valid grafting of Σ_c if and only if θ_c is a valid grafting of $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LH}}}$.*

Proof. Let $\theta_c = \{X/_{\Sigma'_c} M\}$. By Lemma 4.9, $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LH}}}$ is a valid constrained signature. Assume that θ_c is a valid grafting of Σ_c, we verify that

1. Meta-variables declared in Σ'_c are not in $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LH}}}$. This is true since by Def. 4.7, Σ_c and $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LH}}}$ declare exactly the same meta-variables, and by hypothesis, meta-variables declared in Σ'_c are not in Σ_c,
2. $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LH}}}$ has the form $\Sigma_{c_1}.\, X{:}_\Gamma N.\, \Sigma_{c_2}$ where $\Sigma_{c_1}.\, \Sigma'_c :\cdot \Gamma \vdash_c M : N$. By hypothesis, Σ_c has the form $\Sigma'_{c_1}.\, X{:}_\Gamma N'.\, \Sigma_{c'_2}$, thus $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LH}}}$ has the form $\Sigma_{c_1}.\, X{:}_\Gamma N.\, \Sigma_{c_2}$, where
   (a) $\Sigma'_{c_1}.\, \Sigma'_c :\cdot \Gamma \vdash_c M : N'$,
   (b) $\Sigma_{c_1} = (\Sigma'_{c_1})\downarrow_{\lambda_{\mathcal{LH}}}$, and

(c) $N' \xrightarrow{\lambda_{\mathcal{LI}}} N$.

From (a) and (c), we have $\Sigma'_{c_1}.\ \Sigma'_c :\cdot \Gamma \vdash_c M : N$, and using Lemma 4.9 we conclude $(\Sigma'_{c_1})\downarrow_{\lambda_{\mathcal{LI}}}.\ \Sigma'_c :\cdot \Gamma \vdash_c M : N$.

The other sense is similar. □

Lemma 4.23 *If θ_c is a valid grafting of Σ_c, $((\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\theta_c)\downarrow_{\lambda_{\mathcal{LI}}} = (\Sigma_c\theta_c)\downarrow_{\lambda_{\mathcal{LI}}}$.*

Proof. Let $\theta_c = \{X/_{\Sigma'_c}M\}$. By Def. 4.17, Σ_c has the form $\Sigma_{c_1}.\ X{:}_\Gamma N.\ \Sigma_{c_2}$, and by Def. 4.18, $\Sigma_c\theta_c = \Sigma_{c_1}.\ \Sigma'_c.\ \Sigma_{c_2}\{X/M\}$. On the other hand, $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}$ has the form $(\Sigma_{c_1})\downarrow_{\lambda_{\mathcal{LI}}}.\ X{:}_\Gamma N'.\ \Sigma'_{c_2}$, where Σ'_{c_2} is a normal form. By Lemma 4.22, θ_c is a valid grafting of $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}$ too, and thus by Def. 4.17, $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\theta_c = (\Sigma_{c_1})\downarrow_{\lambda_{\mathcal{LI}}}.\ \Sigma'_c.\ \Sigma'_{c_2}\{X/M\}$.

Since $((\Sigma_{c_1})\downarrow_{\lambda_{\mathcal{LI}}}.\ \Sigma'_c)\downarrow_{\lambda_{\mathcal{LI}}} = (\Sigma_{c_1}.\ \Sigma'_c)\downarrow_{\lambda_{\mathcal{LI}}}$, it suffices to show that the normal form of $\Sigma'_{c_2}\{X/M\}$ is equal to the normal form of $\Sigma_{c_2}\{X/M\}$. We use Def. 4.7 and the fact that for arbitrary expressions x, y, if $x \xrightarrow{\lambda_{\mathcal{LI}}} y$, then $x\{X/M\} \xrightarrow{\lambda_{\mathcal{LI}}} y\{X/M\}$. □

Lemma 4.24 *Let Σ_c be a valid constrained signature, if ψ a sequential grafting of $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}$, then $((\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\psi)\downarrow_{\lambda_{\mathcal{LI}}} = (\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{LI}}}$.*

Proof. We proceed by induction on the length of ψ. The base case is trivial. Let $\psi = \theta_c, \psi'$,

$\Rightarrow$ $(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{LI}}} = ((\Sigma_c\theta_c)\psi')\downarrow_{\lambda_{\mathcal{LI}}}$, by Def. 4.20,

$\Rightarrow$ $((\Sigma_c\theta_c)\psi')\downarrow_{\lambda_{\mathcal{LI}}} = ((\Sigma_c\theta_c)\downarrow_{\lambda_{\mathcal{LI}}}\psi')\downarrow_{\lambda_{\mathcal{LI}}}$, by induction hypothesis,

$\Rightarrow$ $((\Sigma_c\theta_c)\downarrow_{\lambda_{\mathcal{LI}}}\psi')\downarrow_{\lambda_{\mathcal{LI}}} = (((\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\theta_c)\downarrow_{\lambda_{\mathcal{LI}}}\psi')\downarrow_{\lambda_{\mathcal{LI}}}$, by Lemma 4.23,

$\Rightarrow$ $(((\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\theta_c)\downarrow_{\lambda_{\mathcal{LI}}}\psi')\downarrow_{\lambda_{\mathcal{LI}}} = (((\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\theta_c)\psi')\downarrow_{\lambda_{\mathcal{LI}}}$, by induction hypothesis,

$\Rightarrow$ $(((\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\theta_c)\psi')\downarrow_{\lambda_{\mathcal{LI}}} = ((\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\psi)\downarrow_{\lambda_{\mathcal{LI}}}$, by Def. 4.20.

□

Lemma 4.25 *Let Σ_c be a valid constrained signature, ψ is a sequential grafting of Σ_c if and only if ψ is a sequential grafting of $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}$.*

Proof. By induction on the length of ψ. If ψ is the empty list, then the conclusion is trivial by Def. 4.20. Otherwise, $\psi = \theta_c, \psi'$

$\Leftrightarrow$ θ_c is a valid grafting of Σ_c and ψ' is a sequential grafting of $\Sigma_c\theta_c$, by Def. 4.20,

$\Leftrightarrow$ θ_c is a valid grafting of $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}$, by Lemma 4.22,

$\Leftrightarrow$ ψ' is a sequential grafting of $(\Sigma_c\theta_c)\downarrow_{\lambda_{\mathcal{LI}}}$, by induction hypothesis,

$\Leftrightarrow$ ψ' is a sequential grafting of $((\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\theta_c)\downarrow_{\lambda_{\mathcal{LI}}}$, by Lemma 4.23,

$\Leftrightarrow$ ψ' is a sequential grafting of $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}\theta_c$, by induction hypothesis,

$\Leftrightarrow$ ψ is a sequential grafting of $(\Sigma_c)\downarrow_{\lambda_{\mathcal{LI}}}$, by Def. 4.20.

□

Definition 4.26 (Derivation) *Let ψ be a sequential grafting of a valid constrained signature Σ_c, we say that ψ is a derivation of Σ_c if and only if $(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} = nil$.*

Lemma 4.27 *Failure signatures do not have derivations.*

Proof. Let Σ_c be a valid constrained signature. Note that if $M =^?_\Gamma N \in \Sigma_c$, where M and N are different ground terms in $\lambda_{\mathcal{L}\Pi}$-normal form, then for any sequential grafting ψ of Σ_c, $M =^?_\Gamma N \in \Sigma_c\psi$, and by Lemma 4.21, $\vdash_c \Sigma_c\psi$. But the normal form of a failure signature is a failure signature, thus $M =^?_\Gamma N \in (\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}}$, and $(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} \neq nil$. □

Definition 4.28 (Parallel Instantiation) *A* parallel instantiation *is a function Ψ from meta-variables to terms such that $\Psi(X) \neq X$ for a number finite of meta-variables X. The function Ψ extends straightforwardly to contexts and arbitrary expressions by*

$$\begin{array}{lcl}
\Psi(nil) & = & nil \\
\Psi(\Gamma.M) & = & \Psi(\Gamma).\Psi(M) \\
\Psi(x) & = & x \quad \textit{if } x \in \{Kind, Type\} \textit{ or } x = \mathbf{1} \textit{ or } x = \uparrow^n \\
\Psi([S]M) & = & [\Psi(S)]\Psi(M) \\
\Psi(\lambda_M.N) & = & \lambda_{\Psi(M)}.\Psi(N) \\
\Psi(\Pi_M.N) & = & \Pi_{\Psi(M)}.\Psi(N) \\
\Psi(M\ N) & = & (\Psi(M)\ \Psi(N)) \\
\Psi(T \circ S) & = & \Psi(T) \circ \Psi(S) \\
\Psi(S \cdot M_{:N}) & = & \Psi(S) \cdot \Psi(M)_{:\Psi(N)}
\end{array}$$

Definition 4.29 *The* restriction *of a parallel instantiation Ψ to a constrained signature Σ_c, denoted by $\Psi\backslash_{\Sigma_c}$, is defined as*

$$\begin{array}{lcll}
\Psi\backslash_{\Sigma_c}(X) & = & \Psi(X) & \textit{if } X \in \Sigma_c \\
\Psi\backslash_{\Sigma_c}(X) & = & X & \textit{otherwise}
\end{array}$$

Definition 4.30 (Solution) *Let Σ_c be a valid constrained signature, we say that a parallel instantiation Ψ is a* solution of Σ_c *if and only if*

- *for any constraint $M_1 =^?_\Gamma M_2 \in \Sigma_c$, we have $\Psi(\Gamma) \vdash \Psi(M_1) : N$, $\Psi(\Gamma) \vdash \Psi(M_2) : N$ and $\Psi(M_1) =_{\lambda_{\mathcal{L}\Pi}} \Psi(M_2)$, and*
- *for any meta-variable declaration $X{:}_\Gamma M \in \Sigma_c$, we have $\Psi(\Gamma) \vdash \Psi(X) : \Psi(M)$.*

In this case we say that Σ_c is a solvable *signature.*

Remark 4.31 *If Ψ is solution of Σ_c, then for any $X \in \Sigma_c$, $\Psi(X)$ is ground.*

The problem to know if a valid constrained signature is solvable is undecidable in the general case. However, we can verify easily the following property.

Remark 4.32 *Failure signatures are not solvable.*

Definition 4.33 (Composition) *Let ψ be a sequential grafting. The* composition *of ψ, denoted by $\widetilde{\psi}$, is the parallel instantiation defined inductively on ψ as follows*

$$\begin{array}{lcll} \widetilde{\psi}(Y) & = & Y & \text{if } \psi \text{ is the empty list} \\ \widetilde{\psi}(X) & = & \widetilde{\psi'}(M) & \text{if } \psi = \{X/_{\Sigma_c} M\}, \psi' \\ \widetilde{\psi}(Y) & = & \widetilde{\psi'}(Y) & \text{if } \psi = \{X/_{\Sigma_c} M\}, \psi' \text{ and } Y \neq X \end{array}$$

Proposition 4.34 *If ψ is a derivation of a valid constrained signature Σ_c, then $\widetilde{\psi}\backslash_{\Sigma_c}$ is a solution of Σ_c.*

Proof. First we prove by induction on the length of ψ that for any context Γ and expression x, $\Gamma\psi = \widetilde{\psi}(\Gamma)$ and $x\psi = \widetilde{\psi}(x)$. Now, since Σ_c is a valid constrained signature we have for any constraint $M_1 =^?_\Gamma M_2$ and meta-variable declaration $X:_\Delta N$ in Σ_c

$$\Sigma_c :\cdot \Gamma \vdash_c M_1 : M \tag{4.1}$$

$$\Sigma_c :\cdot \Gamma \vdash_c M_2 : M \tag{4.2}$$

$$\Sigma_c :\cdot \Delta \vdash_c X : N \tag{4.3}$$

Because ψ is a sequential grafting of Σ_c and by Lemma 4.21, we have

$$\Sigma_c\psi :\cdot \Gamma\psi \vdash_c M_1\psi : M\psi \tag{4.4}$$

$$\Sigma_c\psi :\cdot \Gamma\psi \vdash_c M_2\psi : M\psi \tag{4.5}$$

$$\Sigma_c\psi :\cdot \Delta\psi \vdash_c X\psi : N\psi \tag{4.6}$$

By Lemma 4.9,

$$(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma\psi \vdash_c M_1\psi : M\psi \tag{4.7}$$

$$(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Gamma\psi \vdash_c M_2\psi : M\psi \tag{4.8}$$

$$(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} :\cdot \Delta\psi \vdash_c X\psi : M\psi \tag{4.9}$$

Since ψ is a derivation of Σ_c, we have $(\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}} = nil$, but also $\Gamma\psi = \widetilde{\psi}(\Gamma)$, $\Delta\psi = \widetilde{\psi}(\Delta)$, $M_1\psi = \widetilde{\psi}(M_1)$, and $M_2\psi = \widetilde{\psi}(M_2)$. Therefore, $\widetilde{\psi}$ and $\widetilde{\psi}\backslash_{\Sigma_c}$ are solutions of Σ_c. □

4.4 Solving a Signature

In order to solve a valid constrained signature Σ_c, we can build incrementally a derivation ψ, and then we use Proposition 4.34 to find the solution $\widetilde{\psi}\backslash_{\Sigma_c}$. Informally we proceed as follows:

1. Let $\Sigma'_c = (\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}}$.
2. If $\Sigma'_c = nil$, then exit.
3. If Σ'_c is a failure signature, then fail since by Lemma 4.27, Σ_c does not have derivations.
4. Otherwise, take a meta-variable declaration $X:_\Gamma M$ in Σ'_c, such that M is a $\lambda_{\mathcal{L}\Pi}$-normal form. Such a meta-variable exists by Lemma 4.12. According to Lemma 4.15, there are the following non-disjoint cases for M:

- $M = \mathit{Kind}$. Try to instantiate X with Type.
- $M \in \{\mathit{Kind}, \mathit{Type}\}$. Try to instantiate X with $\Pi_H.K$, where H and K are meta-variables of the right type.
- $M = \Pi_N.N'$. Try to instantiate X with $\lambda_N.Y$, where Y is a meta-variable of the right type.
- M has as type a sort in $\{\mathit{Kind}, \mathit{Type}\}$. Try to instantiate X with $(\underline{n}\ X_1\ \ldots\ X_i)$, where $X_1, \ldots X_i$ are meta-variables of the right type, and $\underline{n}$ is a variable declared in Γ (i.e. $1 \leq n \leq \mathrm{Length}(\Gamma)$). Add also all the constraints necessary to guarantee that the instantiation is well-typed.

5. Continue recursively on the new signature.

The step 4 is the basic step of the method. It finds a valid grafting of the constrained signature that we want to solve.

Definition 4.35 (Elementary Graftings) *Let Σ_c be the normal form of a valid constrained signature such that $\Sigma_c \neq nil$ and Σ_c is not a failure signature. We choose a meta-variable X in Σ_c, such that*

1. $\Sigma'_c.\ X{:}_\Gamma M \preceq \Sigma_c$,
2. $\vdash \Sigma'_c.\ X{:}_\Gamma M$, *and*
3. M *is a* $\lambda_{\mathcal{L}\Pi}$*-normal form.*

Such a meta-variable exists by Lemma 4.12. We define the following graftings by case analysis on M (the cases are not disjoint):

- $M = \mathit{Kind}$. *We consider the grafting* $\{X/_{nil}\,\mathit{Type}\}$.
- $M \in \{\mathit{Kind}, \mathit{Type}\}$. *For any* $M' \in \{\mathit{Kind}, \mathit{Type}\}$, *we consider the grafting* $\{X/_{\Sigma''_c}\Pi_H.K\}$, *where* $H \notin \Sigma_c$, $K \notin \Sigma_c$, *and* $\Sigma''_c = nil.\ H{:}_\Gamma M'.\ K{:}_{\Gamma.H} M$.
- $M = \Pi_N.N'$. *We consider the grafting* $\{X/_{\Sigma''_c}\lambda_N.Y\}$, *where* $Y \notin \Sigma_c$, *and* $\Sigma''_c = nil.\ Y{:}_{\Gamma.N} N'$.
- $\Sigma'_c :\cdot \Gamma \vdash M : s$. *We need some definitions.*

  Definition 4.36 *The standard π-reduction is inductively defined as*

  $$\begin{array}{lcll} \pi([S](\Pi_M.N)) & = & \Pi_{[S]M}.\pi([(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M}]N) & \\ \pi([S]M) & = & [S]M & \text{if } M \text{ is not a product} \end{array}$$

  Definition 4.37 *Let $\chi = X_1, H_1, K_1, X_2, H_2, K_2, \ldots$ be an infinite list of distinct names of meta-variables. We define the set $\chi(\Gamma, M, N)^i$, $i \geq 0$, as formed by the triples (Σ'_c, M', N') satisfying the following conditions:*

  - *If $i = 0$, then $\Sigma'_c = nil$, $M' = M$, and $N' = N$.*
  - *If $i > 0$ and $(\Sigma''_c, M'', N'') \in \chi(\Gamma, M, N)^{i-1}$, then*

- *If* N'' *has the form* $\Pi_{N_1}.N_2$, *then*
  - $\Sigma'_c = \Sigma''_c.\ X_i{:}_\Gamma N_1$,
  - $M' = (M''\ X_i)$, *and*
  - $N' = \pi([\uparrow^0 \cdot X_{i:N_1}]N_2)$.
- *Otherwise, let* s' *be in* $\{Kind, Type\}$
  - $\Sigma'_c = \Sigma''_c.\ H_i{:}_\Gamma s'.\ K_i{:}_{\Gamma.H_i} s.\ X_i{:}_\Gamma H_i.\ N'' =^?_\Gamma \Pi_{H_i}.K_i$,
  - $M' = (M''\ X_i)$, *and*
  - $N' = [\uparrow^0 \cdot X_{i:H_i}]K_i$.

For $1 \leq n \leq \mathrm{Length}(\Gamma)$, *such that*

1. $\Sigma'_c :\cdot \Gamma \vdash \underline{n} : N$, *and*
2. $\Sigma'_c :\cdot \Gamma \vdash N : s$

we consider all the graftings

$$\{X/_{\Sigma''_c.\ M =^?_\Gamma N'} M'\}$$

where $(\Sigma''_c, M', N') \in \chi(\Gamma, \underline{n}, (N)\downarrow_{\lambda_{\mathcal{LP}}})^i$ *for some list* χ *of infinite distinct names of meta-variables not in* Σ_c, *and* $i \geq 0$.

All the graftings considered above form the set of elementary graftings *of the meta-variable* X *in* Σ_c.

Proposition 4.38 (Elementary Graftings) *Let* Σ_c *be the normal form of a valid constrained signature such that* $\Sigma_c \neq nil$ *and* Σ_c *is not a failure signature. If* X *is a meta-variable in* Σ_c *such that it is well-typed without constraints, then elementary graftings of* X *are valid graftings of* Σ_c.

Proof. By Lemma 4.12, Σ_c has the form $\Sigma_{c_1}.\ X{:}_\Gamma N.\ \Sigma_{c_2}$, where N is a $\lambda_{\mathcal{LP}}$-normal form. Using Lemma 4.5 and the typing rules, we can verify that

$$\vdash_c \Sigma_{c_1} :\cdot \Gamma \tag{4.10}$$

$$N = Kind \text{ or } \Sigma_{c_1} :\cdot \Gamma \vdash_c N : s \tag{4.11}$$

We reason by case analysis on N, and we consider all the elementary graftings of X.

- $N = Kind$ and $\theta_c = \{X/_{nil}\, Type\}$. Trivially, the meta-variables declared in *nil* are not in Σ_c, and using Eq. 4.10 with the rule (Type), we get $\Sigma_{c_1} :\cdot \Gamma \vdash_c Type : Kind$. Therefore, θ_c is a valid grafting of Σ_c.
- $N \in \{Kind, Type\}$. For any $N' \in \{Kind, Type\}$, we consider the grafting $\{X/_{\Sigma''_c}\Pi_H.K\}$, where $H \notin \Sigma_c$, $K \notin \Sigma_c$, and $\Sigma''_c = nil.\ H{:}_\Gamma N'.\ K{:}_{\Gamma.H} N$. By Def. 4.17, it suffices to verify that $\Sigma_{c_1}.\ \Sigma''_c :\cdot \Gamma \vdash_c \Pi_H.K : N$. We consider two cases according to N'.
  - $N' = Kind$. We have the derivation

$$\frac{\text{Eq. 4.10}}{\vdash_c \Sigma_{c_1}.\ H{:}_\Gamma Kind}\,(\text{Metavar-Decl}_1)$$

– $N' = \mathit{Type}$. We have the derivation

$$\dfrac{\dfrac{\text{Eq. 4.10}}{\Sigma_{c_1} :\cdot \Gamma \vdash_c \mathit{Type} : \mathit{Kind}}\,(\text{Type})}{\vdash_c \Sigma_{c_1}.\, H :_\Gamma \mathit{Type}}\,(\text{Metavar-Decl}_2)$$

In both cases, we have

$$\vdash_c \Sigma_{c_1}.\, H :_\Gamma N' \tag{4.12}$$

We continue the derivation as follows:

$$\dfrac{\dfrac{\text{Eq. 4.12}}{\Sigma_{c_1}.\, H :_\Gamma N' :\cdot \Gamma \vdash_c H : N'}\,(\text{Metavar})}{\vdash_c \Sigma_{c_1}.\, H :_\Gamma N' :\cdot \Gamma.H}\,(\text{Var-Decl})$$

Now, we consider two cases according to N.

– $N = \mathit{Kind}$. We have the derivation

$$\dfrac{\vdash_c \Sigma_{c_1}.\, H :_\Gamma N' :\cdot \Gamma.H}{\vdash_c \Sigma_{c_1}.\, H :_\Gamma N'.\, K :_{\Gamma.H} \mathit{Kind}}\,(\text{Metavar-Decl}_1)$$

– $N = \mathit{Type}$. We have the derivation

$$\dfrac{\dfrac{\vdash_c \Sigma_{c_1}.\, H :_\Gamma N' :\cdot \Gamma.H}{\Sigma_{c_1}.\, H :_\Gamma N' :\cdot \Gamma \vdash_c \mathit{Type} : \mathit{Kind}}\,(\text{Type})}{\vdash_c \Sigma_{c_1}.\, H :_\Gamma N'.\, K :_{\Gamma.H} \mathit{Type}}\,(\text{Metavar-Decl}_2)$$

In both cases, we have

$$\vdash_c \Sigma_{c_1}.\, H :_\Gamma N'.\, K :_{\Gamma.H} N \tag{4.13}$$

We continue the derivation as follows:

$$\dfrac{\dfrac{\text{Eq. 4.13}}{\Sigma_{c_1}.\, H :_\Gamma N'.\, K :_{\Gamma.H} N :\cdot \Gamma.H \vdash_c K : N}\,(\text{Metavar})}{\Sigma_{c_1}.\, H :_\Gamma N'.\, K :_{\Gamma.H} N :\cdot \Gamma \vdash_c \Pi_H K : N}\,(\text{Prod})$$

- $N = \Pi_{N_1}.N_2$. We consider the grafting $\{X/_{\Sigma''_c} \lambda_{N_1}.Y\}$, where $Y \notin \Sigma_c$, and $\Sigma''_c = \mathit{nil}.\, Y :_{\Gamma.N_1} N_2$. By Def. 4.17, it suffices to verify that $\Sigma_{c_1}.\, \Sigma''_c :\cdot \Gamma \vdash_c \lambda_{N_1}.Y : N$. As in the previous case we can derivate

$$\Sigma_{c_1}.\, \Sigma''_c :\cdot \Gamma.n_1 \vdash_c Y : N_2 \tag{4.14}$$

We conclude by applying the rule (Abs) to Eq. 4.14 and Eq. 4.11.

- $\Sigma_1 :\cdot \Gamma \vdash N : s$. For $1 \leq n \leq \text{Length}(\Gamma)$ such that

  1. $\Sigma_1 :\cdot \Gamma \vdash \underline{n} : M$, and
  2. $\Sigma_1 :\cdot \Gamma \vdash M : s$

we consider all the graftings

$$\{X/_{\Sigma_c''.\ N=^?_\Gamma N'} M'\}$$

where $(\Sigma_c'', M', N') \in \chi(\Gamma, \underline{n}, (M)\downarrow_{\lambda\mathcal{L}\Pi})^k$ for some list χ of infinite distinct names of meta-variables not in Σ_c, and $k \geq 0$. First we prove by induction on k that if $(\Sigma_c'', M', N') \in \chi(\Gamma, \underline{n}, (M)\downarrow_{\lambda\mathcal{L}\Pi})^k$, then

$$\Sigma_{c_1}.\ \Sigma_c'' :\cdot \Gamma \vdash_c M' : N' \tag{4.15}$$

and

$$\Sigma_{c_1}.\ \Sigma_c'' :\cdot \vdash_c N' : s \tag{4.16}$$

For that we show by induction on typing derivation that the standard π-reduction preserves typing. Now, we have

$$\frac{\Sigma_{c_1}.\ \Sigma_c'' :\cdot \Gamma \vdash_c N : s \quad \Sigma_{c_1}.\ \Sigma_c'' :\cdot \Gamma \vdash_c N' : s}{\vdash_c \Sigma_{c_1}.\ \Sigma_c''.\ N =^?_\Gamma N'}\ (\text{Constraint})$$

Finally, we conclude with the rule (Conv) that

$$\Sigma_{c_1}.\ \Sigma_c''.\ N =^?_\Gamma N' :\cdot \Gamma \vdash_c M' : N$$

□

Definition 4.39 (Search Tree) *Let Σ_c be a valid constrained signature, we build a* search tree *of Σ_c, where nodes are labeled by normal constrained signatures and edges by elementary graftings, in the following way:*

- *The root is labeled by $(\Sigma_c)\downarrow_{\lambda\mathcal{L}\Pi}$.*
- *Nodes labeled by the empty signature or by failure signatures are leaves.*
- *If a node is labeled by a signature Σ_c which is not the empty signature or a failure signature, we choose a meta-variable X in Σ_c such that it is well-typed without constraints and for each elementary grafting $\theta_c = \{X/_{\Sigma_c'} M\}$ of X, we grow an edge labeled by θ_c to a new node labeled by $(\Sigma_c\theta_c)\downarrow_{\lambda\mathcal{L}\Pi}$.*

Lemma 4.40 *Let $\Sigma_{c_0} \xrightarrow{\theta_{c_0}} \Sigma_{c_1} \longrightarrow \ldots \xrightarrow{\theta_{c_{i-1}}} \Sigma_{c_i}$, $i \geq 0$, be a path of a tree search of Σ_c such that $\Sigma_{c_0} = (\Sigma_c)\downarrow_{\lambda\mathcal{L}\Pi}$. The list of graftings $\psi = \theta_{c_0}, \ldots, \theta_{c_i}$ is a sequential grafting of Σ_c, and $\Sigma_{c_i} = (\Sigma_c\psi)\downarrow_{\lambda\mathcal{L}\Pi}$.*

Proof. By induction on i. The case base is trivial. Assume that $i > 0$ and take $\psi' = \theta_{c_1}, \ldots, \theta_{c_i}$. By construction, θ_{c_0} is an elementary grafting of X in Σ_{c_0}. Thus, by Proposition 4.38, θ_{c_0} is a valid grafting of Σ_{c_0} and $\Sigma_{c_1} = (\Sigma_{c_0}\theta_{c_0})\downarrow_{\lambda\mathcal{L}\Pi}$ is well-defined. By induction hypothesis, ψ' is a sequential grafting of $\Sigma_{c_0}\theta_{c_0}$, and $\Sigma_{c_i} = (\Sigma_{c_0}\theta_{c_0}\psi')\downarrow_{\lambda\mathcal{L}\Pi} = (\Sigma_{c_0}\psi)\downarrow_{\lambda\mathcal{L}\Pi}$. By Def. 4.20, ψ is a sequential grafting of $\Sigma_{c_0} = (\Sigma_c)\downarrow_{\lambda\mathcal{L}\Pi}$. Therefore, by Lemma 4.25, ψ is a sequential grafting of Σ_c, and by Lemma 4.24, $\Sigma_{c_i} = (\Sigma_c\psi)\downarrow_{\lambda\mathcal{L}\Pi}$. □

Theorem 4.41 (Soundness) *Let $(\Sigma_c)\downarrow_{\lambda\mathcal{L}\Pi} \xrightarrow{\psi} nil$ be a path of a tree search of Σ_c. The composition of ψ restricted to Σ_c, i.e. $\widetilde{\psi}\backslash_{\Sigma_c}$, is a solution of Σ_c.*

Proof. By Lemma 4.40, ψ is a sequential grafting of Σ_c, and $nil = (\Sigma_c\psi)\downarrow_{\lambda_{\mathcal{L}\Pi}}$. Therefore, by Def. 4.26, ψ is a derivation of Σ_c. We conclude with Proposition 4.34 that $\widetilde{\psi}\backslash_{\Sigma_c}$ is a solution of Σ_c. □

Conjecture 4.42 (Completeness) *Let Σ_c be a solvable constrained signature, there exists a path $(\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}} \xrightarrow{\psi} nil$ in any tree search of Σ_c.*

A semi-algorithm to solve a valid constrained signature is to enumerate the nodes of a search tree in order to find a leave labeled by the *nil* signature. The enumeration must deal with infinite paths in the tree, but also with infinite branching due to the fact that the set of elementary graftings of a meta-variable is potentially infinite.

4.5 Examples

4.5.1 In Dependent Types

Assume a context where we have the transitivity axiom: $\forall x \forall y \forall z.(x \leq y) \Rightarrow (y \leq z) \Rightarrow (x \leq z)$, proof-terms of $0 \leq 1$ and $1 \leq 2$, and we search a proof-terms of $0 \leq 2$. In order to apply the transitivity axiom, we need to find a term $\square$ such that $0 \leq \square$ and $\square \leq 2$. Of course, the immediate solution is $\square = 1$. All these steps are simulated by the proof synthesis method.

Formally, we let the context

$$\Gamma = \begin{array}{l} nil.\ nat{:}Type.\ \leq{:}nat{\to}(nat{\to}\ Type). \\ 0{:}nat.\ 1{:}nat.\ 2{:}nat.\ p_1{:}(0 \leq 1).\ p_2{:}(1 \leq 2). \\ trans{:}(\Pi x{:}nat.\Pi y{:}nat.\Pi z{:}nat.\Pi h_1{:}(x \leq y).\Pi h_2{:}(y \leq z).(x \leq z)) \end{array}$$

and we search a solution for the valid constrained signature

$$\Sigma_c = nil.\ X{:}_\Gamma(0 \leq 2)$$

Thus, we search a valid grafting for X. The first step is to apply the transitivity axiom. Let $M = \Pi x{:}nat.\Pi y{:}nat.\Pi z{:}nat.\Pi h_1{:}(x \leq y).\Pi h_2{:}(y \leq z).(x \leq z)$, we verify that $\Sigma_c = (\Sigma_c)\downarrow_{\lambda_{\mathcal{L}\Pi}} = (\Sigma_c)^{-c}$, $\vdash \Sigma_c$, $\Gamma \vdash (0 \leq 2) : Type$, and $\Gamma \vdash trans : M$. We take the elementary grafting

$$\{X/_{\Sigma_c''.\ (0\leq 2)=^?_\Gamma N'} M'\}$$

where

$$\Sigma_c'' = \begin{array}{l} nil.\ X_1{:}_\Gamma nat.\ X_2{:}_\Gamma[x := X_{1:nat}]nat.\ X_3{:}_\Gamma[y := X_{2:nat}][x := X_{1:nat}]nat. \\ X_4{:}_\Gamma[z := X_{3:nat}][y := X_{2:nat}][x := X_{1:nat}](x \leq y). \\ X_5{:}_\Gamma[h_1 := X_{4:(x\leq y)}][z := X_{3:nat}][y := X_{2:nat}][x := X_{1:nat}](y \leq z) \end{array} \quad (4.17)$$

$$M' = (trans\ X_1\ X_2\ X_3\ X_4\ X_5) \quad (4.18)$$

$$N' = [h_2 := X_{5:(y\leq z)}][h_1 := X_{4:(x\leq y)}][z := X_{3:nat}][y := X_{2:nat}][x := X_{1:nat}](x \leq z) \quad (4.19)$$

Note that $(\Sigma_c'', M', N') \in \chi(\Gamma, trans, M)^5$. We get the normal constrained signature

$$nil.\ X_1{:}_\Gamma nat.\ X_2{:}_\Gamma nat.\ X_3{:}_\Gamma nat.\ X_4{:}_\Gamma(X_1 \leq X_2).\ X_5{:}_\Gamma(X_2 \leq X_3).\ (0 \leq 2) =^?_\Gamma (X_1 \leq X_3)$$

Now, we need to find valid graftings for the new meta-variables. In order to instantiate X_1 with 0, we verify that $\vdash nil.\ X_1{:}_\Gamma nat$, $\Gamma \vdash nat : Type$, and $\Gamma \vdash 0 : nat$. We take the elementary grafting $\{X_1/_{nil.\ nat=^?_\Gamma nat}0\}$, where $(nil, 0, nat) \in \chi(\Gamma, 0, nat)^0$. We get the normal constrained signature

$$nil.\ X_2{:}_\Gamma nat.\ X_3{:}_\Gamma nat.\ X_4{:}_\Gamma(0 \leq X_2).\ X_5{:}_\Gamma(X_2 \leq X_3).\ (0 \leq 2) =^?_\Gamma (0 \leq X_3)$$

To instantiate X_3 with 2, we verify that $\vdash nil.\ X_2{:}_\Gamma nat.\ X_3{:}_\Gamma nat$, $nil.\ X_2{:}_\Gamma nat :\cdot \Gamma \vdash nat : Type$, and $nil.\ X_2{:}_\Gamma nat :\cdot \Gamma \vdash 2 : nat$. We take the elementary grafting $\{X_3/_{nil.\ nat=^?_\Gamma nat}2\}$, where $(nil, 2, nat) \in \chi(\Gamma, 2, nat)^0$. We get the normal constrained signature

$$nil.\ X_2{:}_\Gamma nat.\ X_4{:}_\Gamma(0 \leq X_2).\ X_5{:}_\Gamma(X_2 \leq 2)$$

Now, we can instantiate X_2 with 1 since we verify that $\vdash nil.\ X_2{:}_\Gamma nat$, $\Gamma \vdash nat : Type$, and $\Gamma \vdash 1 : nat$. We take the elementary grafting $\{X_2/_{nil.\ nat=^?_\Gamma nat}1\}$, where $(nil, 1, nat) \in \chi(\Gamma, 1, nat)^0$. We get the normal constrained signature

$$nil.\ X_4{:}_\Gamma(0 \leq 1).\ X_5{:}_\Gamma(1 \leq 2)$$

We finish by instantiating X_4 with p_1 and X_5 with p_2. We verify that $\vdash nil.\ X_4{:}_\Gamma(0 \leq 1)$, $\Gamma \vdash (0 \leq 1) : Type$, and $\Gamma \vdash p_1 : (0 \leq 1)$. We take the elementary grafting $\{X_4/_{nil.\ (0\leq 1)=^?_\Gamma(0\leq 1)}p_1\}$, where $(nil, p_1, (0 \leq 1)) \in \chi(\Gamma, p_1, (0 \leq 1))^0$. We get the normal constrained signature

$$nil.\ X_5{:}_\Gamma(1 \leq 2)$$

We verify that $\vdash nil.\ X_5{:}_\Gamma(1 \leq 2)$, $\Gamma \vdash (1 \leq 2) : Type$, and $\Gamma \vdash p_2 : (1 \leq 2)$. We take the elementary grafting $\{X_5/_{nil.\ (1\leq 2)=^?_\Gamma(1\leq 2)}p_2\}$, where $(nil, p_2, (1 \leq 2)) \in \chi(\Gamma, p_2, (1 \leq 2))^0$. We get the normal constrained signature nil.

A solution Ψ of Σ_c is obtained by composing the derivation that we have built, and restricting it to Σ_c:

$$\begin{array}{rcl} \Psi(X) & = & (trans\ 0\ 1\ 2\ p_1\ p_2) \\ \Psi(Y) & = & Y \quad \text{if } Y \neq X \end{array}$$

4.5.2 An Example with Polymorphism

The following example is taken from a similar one developed in [24].

Let the context

$$\begin{array}{rl} & nil.\ nat{:}Type.\ bool{:}Type.\ i{:}\Pi t{:}Type.Type. \\ \Gamma = & h{:}(i\ \Pi x{:}nat.bool).\ pol{:}(\Pi t{:}Type.\Pi y{:}(i\ t).t) \\ & 0{:}nat \end{array}$$

and the valid constrained signature

$$\Sigma_c = nil.\ X{:}_\Gamma bool$$

In order to solve Σ_c, we use the polymorphic axiom pol, and in the next steps it will be applied to the type $nat{\rightarrow}bool$.

First, we verify that $\Sigma_c = (\Sigma_c){\downarrow_{\lambda\mathcal{L}\Pi}} = (\Sigma_c)^{-c}$, $\vdash \Sigma_c$, $\Gamma \vdash bool : Type$, and $\Gamma \vdash pol : (\Pi t{:}Type.\Pi y{:}(i\ t).t)$. We take the elementary grafting $\{X/_{\Sigma''_c.\ bool=^?_\Gamma N'}M'\}$, where

$$\Sigma_c'' = \begin{array}{l} nil.\ X_1{:}_\Gamma\ Type.\ X_2{:}_\Gamma [t := X_{1:\ Type}](i\ t). \\ H_3{:}_\Gamma\ Type.\ K_3{:}_{\Gamma.\ x:H_3}\ Type.\ X_3{:}_\Gamma H_3. \\ [y := X_{2:(i\ t)}][t := X_{1:Type}]t =^?_\Gamma \Pi x{:}H_3.K_3 \end{array} \tag{4.20}$$

$$M' = (pol\ X_1\ X_2\ X_3) \tag{4.21}$$

$$N' = [x := X_{3:H_3}]K_3 \tag{4.22}$$

Notice that $(\Sigma_c'', M', N') \in \chi(\Gamma, pol, \Pi t{:}Type.\Pi y{:}(i\ t).t)^3$. We get the normal constrained signature

$$\begin{array}{l} nil.\ X_1{:}_\Gamma\ Type.\ X_2{:}_\Gamma (i\ X_1). \\ H_3{:}_\Gamma\ Type.\ K_3{:}_{\Gamma.\ x:H_3}\ Type.\ X_3{:}_\Gamma H_3. \\ X_1 =^?_\Gamma \Pi x{:}H_3.K_3 \end{array}$$

We verify that $\vdash nil.\ X_1{:}_\Gamma\ Type$. We take the elementary grafting

$$\{X_1/_{nil.\ H:\Gamma\ Type.\ K:_{\Gamma.\ x:H}\ Type}\Pi x{:}H.K\}$$

We get the normal constrained signature

$$\begin{array}{l} nil.\ H{:}_\Gamma\ Type.\ K{:}_{\Gamma.\ x:H}\ Type \\ X_2{:}_\Gamma (i\ \Pi x{:}H.K). \\ H_3{:}_\Gamma\ Type.\ K_3{:}_{\Gamma.\ x:H_3}\ Type.\ X_3{:}_\Gamma H_3. \\ \Pi x{:}H.K =^?_\Gamma \Pi x{:}H_3.K_3 \end{array}$$

Now, we instantiate H with nat and K with $bool$.

We verify that $\vdash nil.\ H{:}_\Gamma\ Type$, $\Gamma \vdash Type : Kind$, and $\Gamma \vdash nat : Type$. We take the elementary grafting $\{H/_{nil.\ Type=^?_\Gamma Type}nat\}$, where $(nil, nat, Type) \in \chi(\Gamma, nat, Type)^0$. We get the normal constrained signature

$$\begin{array}{l} nil.\ nat{:}_\Gamma\ Type.\ K{:}_{\Gamma.\ x:nat}\ Type \\ X_2{:}_\Gamma (i\ \Pi x{:}nat.K). \\ H_3{:}_\Gamma\ Type.\ K_3{:}_{\Gamma.\ x:H_3}\ Type.\ X_3{:}_\Gamma H_3. \\ \Pi x{:}nat.K =^?_\Gamma \Pi x{:}H_3.K_3 \end{array}$$

We verify that $\vdash nil.\ K{:}_{\Gamma.\ x:nat}\ Type$, $\Gamma.\ x{:}nat \vdash Type : Kind$, and $\Gamma.\ x{:}nat \vdash bool : Type$. We take the elementary grafting

$$\{K/_{nil.\ Type=^?_{\Gamma.\ x:nat}\ Type}bool\}$$

where $(nil, bool, Type) \in \chi((\Gamma.\ x{:}nat), bool, Type)^0$. We get the normal constrained signature

$$\begin{array}{l} nil.\ X_2{:}_\Gamma (i\ \Pi x{:}nat.bool). \\ H_3{:}_\Gamma\ Type.\ K_3{:}_{\Gamma.\ x:H_3}\ Type.\ X_3{:}_\Gamma H_3. \\ \Pi x{:}nat.bool =^?_\Gamma \Pi x{:}H_3.K_3 \end{array}$$

Now we can use the hypothesis h to instantiate X_2. We verify that $\vdash nil.\ X_2{:}_\Gamma (i\ \Pi x{:}nat.bool)$, $\Gamma \vdash \Pi x{:}nat.bool : Type$, and $\Gamma \vdash h : (i\ \Pi x{:}nat.bool)$. We take the elementary grafting

$$\{X_2/_{nil.\ (i\ \Pi x:nat.bool)=^?_\Gamma (i\ \Pi x:nat.bool)}h\}$$

where $(nil, h, (i\ \Pi x{:}nat.bool)) \in \chi(\Gamma, h, (i\ \Pi x{:}nat.bool))^0$. We get the normal constrained signature

$$nil.\ H_3{:}_\Gamma\ Type.\ K_3{:}_{\Gamma.\ x:H_3}\ Type.\ X_3{:}_\Gamma H_3.$$
$$\Pi x{:}nat.bool =^?_\Gamma \Pi x{:}H_3.K_3$$

We finish by instantiating H_3 with nat, K_3 with $bool$, and X_3 with 0.

We verify that $\vdash nil.\ H_3{:}_\Gamma\ Type$, $\Gamma \vdash Type : Kind$, and $\Gamma \vdash nat : Type$. We take the elementary grafting $\{H_3/_{nil.\ Type=^?_\Gamma Type}nat\}$, where $(nil, nat, Type) \in \chi(\Gamma, nat, Type)^0$. We get the normal constrained signature

$$nil.\ K_3{:}_{\Gamma.\ x:nat}\ Type.\ X_3{:}_\Gamma nat.$$
$$\Pi x{:}nat.bool =^?_\Gamma \Pi x{:}nat.K_3$$

We verify that $\vdash nil.\ K_3{:}_{\Gamma.\ x:nat}\ Type$, $\Gamma.\ x{:}nat \vdash Type : Kind$, and $\Gamma.\ x{:}nat \vdash bool : Type$. We take the elementary grafting

$$\{K_3/_{nil.\ Type=^?_{\Gamma.\ x:nat}Type}bool\}$$

where $(nil, bool, Type) \in \chi((\Gamma.\ x{:}nat), bool, Type)^0$. We get the normal constrained signature $nil.\ X_3{:}_\Gamma nat$.

We verify that $\vdash nil.\ X_3{:}_\Gamma nat$, $\Gamma \vdash nat : Type$, and $\Gamma \vdash 0 : nat$. We take the elementary grafting $\{X_3/_{nil.\ nat=^?_\Gamma nat}0\}$, where $(nil, 0, nat) \in \chi(\Gamma, 0, nat)^0$. We get the normal constrained signature nil.

We obtain a solution of Σ_c by composing the derivation that we have built, and restricting it to Σ_c:

$$\Psi(X) = (pol\ \Pi x{:}nat.bol\ h\ 0)$$
$$\Psi(Y) = Y \quad \text{if } Y \neq X$$

4.6 Efficiency Improvements

As usual in search problems in trees, efficiency improvements means early detecting of failure and success nodes. In this way, unnecessary explorations of nodes are avoided. In [24], efficiency issues for a method of proof synthesis in the Cube of Type Systems are discussed. We adapt some of these techniques to our method based on explicit substitutions.

4.6.1 Using the Rigid-Rigid Constraints

It is not difficult to see that the general form of a rigid term is invariant under $\lambda_{\mathcal{L}\Pi}$-reductions and instantiations of meta-variables. This remark motivates the following definition.

Definition 4.43 *Let M, N be terms in $\lambda_{\mathcal{L}\Pi}$, we say that M and N are* approximatively equals, *denoted by $M \approx N$, if and only if either*

- *M and N are in both in $\mathcal{R}^n_i$ for some $i \geq 0$, $n > 0$, or*
- *M and N are both abstractions, or*
- *M and N are both products, or*
- *$M = N = Type$, or*
- *$M = N = Kind$.*

We write $M \not\approx N$, if it is not true that $M \approx N$.

We can see that two rigid terms M,N that are not approximatively equal, are not $\lambda_{\mathcal{L}\Pi}$-convertible via instantiation of meta-variables. Thus, we re-define a failure signature as

Definition 4.44 *Let Σ_c be the normal form of a constrained signature, we say that Σ_c is a* failure signature *if it contains a constraint relating two terms $M, N \in \mathcal{R}$, such that $M \not\approx N$.*

We can prove the following properties of the $\lambda_{\mathcal{L}\Pi}$-calculus using the Geuvers' lemma (Theorem 5.14), which is a weak form of Church-Rosser (for details see Chapter 5).

- $\lambda_{M_1}.M_2 =_{\lambda_{\mathcal{L}\Pi}} \lambda_{N_1}.N_2$ if and only if $M_1 =_{\lambda_{\mathcal{L}\Pi}} N_1$ and $M_2 =_{\lambda_{\mathcal{L}\Pi}} N_2$,
- $\Pi_{M_1}.M_2 =_{\lambda_{\mathcal{L}\Pi}} \Pi_{N_1}.N_2$ if and only if $M_1 =_{\lambda_{\mathcal{L}\Pi}} N_1$ and $M_2 =_{\lambda_{\mathcal{L}\Pi}} N_2$, and
- let $M_1, M_2 \in \mathcal{R}_i^n$ for some $i, n \geq 0$, $(M_1\ M_2) =_{\lambda_{\mathcal{L}\Pi}} (N_1\ N_2)$ if and only if $M_1 =_{\lambda_{\mathcal{L}\Pi}} N_1$ and $M_2 =_{\lambda_{\mathcal{L}\Pi}} N_2$.

Therefore, we add the following rules to calculate $\lambda_{\mathcal{L}\Pi}$-normal forms of a valid constrained signature.

$$\Sigma_c'.\ \lambda_{M_1}.M_2 =^?_\Gamma \lambda_{N_1}N_2.\ \Sigma_c'' \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma_c'.\ M_1 =^?_\Gamma N_1.\ M_2 =^?_{\Gamma.M_1} N_2.\ \Sigma_c''$$

$$\Sigma_c'.\ \Pi_{M_1}.M_2 =^?_\Gamma \Pi_{N_1}N_2.\ \Sigma_c'' \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma_c'.\ M_1 =^?_\Gamma N_1.\ M_2 =^?_{\Gamma.M_1} N_2.\ \Sigma_c''$$

$$\Sigma_c'.\ (M_1\ M_2) =^?_\Gamma (N_1\ N_2).\ \Sigma_c'' \xrightarrow{\lambda_{\mathcal{L}\Pi}} \Sigma_c'.\ M_1 =^?_\Gamma N_1.\ M_2 =^?_\Gamma N_2.\ \Sigma_c'' \quad \text{if } (M_1\ M_2) \approx (N_1\ N_2)$$

4.6.2 Using the Flexible-Rigid Constraints

When we have a flexible-rigid constraint as $[\uparrow^m \cdot M_{j:N_j} \cdot \ldots \cdot M_{1:N_1}]X =^?_\Gamma (\underline{m'}\ M_1\ \ldots\ M_j')$, the only variables that are candidates for the instantiation of X are $\underline{1}, \ldots \underline{j}$, and eventually, the index $\underline{m' + j - m}$. This fact is reflected in the next elementary graftings.

Let X be a meta-variable in Σ_c, such that

1. $\Sigma_c'.\ X{:}_\Gamma M \preceq \Sigma_c$,
2. $\vdash \Sigma_c'.\ X{:}_\Gamma M$,
3. $\Sigma_c' :\cdot \Gamma \vdash M : s$, and
4. M is a $\lambda_{\mathcal{L}\Pi}$-normal form.

If $N =^?_\Gamma N' \in \Sigma_c'$, where $N \in \mathcal{F}_X^{(j,m)}$ and $N' \in \mathcal{R}$, then we **only** consider the variables n such that

1. $n \in \{1, \ldots j\} \cup (\text{if } (N' \in \mathcal{R}_{j'}^{m'} \wedge m < m') \text{ then } \{m' + j - m\} \text{ else } \emptyset)$
2. $\Sigma_c' :\cdot \Gamma \vdash \underline{n} : M'$, and
3. $\Sigma_c' :\cdot \Gamma \vdash M' : s$.

For these variables, we consider all the graftings

$$\{X/_{\Sigma_c''.\ M =^?_\Gamma M''} N''\}$$

where $(\Sigma_c'', N'', M'') \in \chi(\Gamma, \underline{n}, (M')\downarrow_{\lambda_{\mathcal{L}\Pi}})^i$ for some list χ of infinite distinct names of meta-variables not in Σ_c, and $i \geq 0$.

4.6.3 Solving Signatures in $\lambda\Pi_{\mathcal{L}}$

In a dependent-typed λ-calculus without polymorphism, the arity of a variable is fixed by its type. This property is no more true when we allow meta-variables of types and kinds. For example consider a variable y such that $(\Sigma_c)^{-c} :\cdot \Gamma \vdash y : \Pi x_1{:}M_1 \ldots \Pi x_k{:}M_k.M$. If $M \in \mathcal{F}_X^{(i,n)}$, then the arity of y depends on the instantiations of X. However, if $M \in \mathcal{R}_i^n$, then the arity of y is fixed to k. We can use this fact to avoid the exploration of certain nodes when we are solving signatures in $\lambda\Pi_{\mathcal{L}}$. Moreover, if we restraint to signatures without meta-variables of types and kinds, then we can consider finitely branching search trees **only** by re-defining elementary graftings as follows:

Definition 4.45 (Elementary Graftings in $\lambda\Pi_{\mathcal{L}}$) *Let Σ_c be the normal form of a valid constrained signature such that $\Sigma_c \neq nil$ and Σ_c is not a failure signature. We choose a meta-variable X in Σ_c, such that*

1. $\Sigma_c'.\ X{:}_\Gamma M \preceq \Sigma_c$,
2. $\Sigma_c' :\cdot \Gamma \vdash M : Type$, *and*
3. M *is a* $\lambda_{\mathcal{L}\Pi}$*-normal form.*

We define the following graftings by case analysis on M (the cases are not disjoint):

- $M = \Pi_N.N'$. *We consider the grafting* $\{X/_{\Sigma_c''}\lambda_N.Y\}$, *where* $Y \notin \Sigma_c$, *and* $\Sigma_c'' = nil.\ Y{:}_{\Gamma.N}N'$.
- $\Sigma_c' :\cdot \Gamma \vdash M : s$. *We re-define,*

  Definition 4.46 *Let $\chi = X_1, X_2, \ldots$ be an infinite list of distinct names of meta-variables. We define the set $\chi(\Gamma, M, N)^i$, $i \geq 0$, as formed by the triples (Σ_c', M', N') satisfying the following conditions:*

  - *If $i = 0$, then $\Sigma_c' = nil$, $M' = M$, and $N' = N$.*
  - *If $i > 0$ and $(\Sigma_c'', M'', N'') \in \chi(\Gamma, M, N)^{i-1}$, then*
    - *If N'' has the form $\Pi_{N_1}.N_2$, then*
      - $\Sigma_c' = \Sigma_c''.\ X_i{:}_\Gamma N_1$,
      - $M' = (M''\ X_i)$, *and*
      - $N' = \pi([\uparrow^0 \cdot X_{i:N_1}]N_2)$.
    - *Otherwise, $\Sigma_c' = \Sigma_c''$, $M' = M''$, and $N' = N''$.*

  For $1 \leq n \leq \mathrm{Length}(\Gamma)$ such that

  1. $\Sigma_c' :\cdot \Gamma \vdash \underline{n} : N$,
  2. $\Sigma_c' :\cdot \Gamma \vdash N : s$, *and*
  3. $N = \Pi_{N_1}. \ldots \Pi_{N_k}.N'$ *where* $N' \in \mathcal{R}_{i'}^{n'}$,

  we consider all the graftings

  $$\{X/_{\Sigma_c''.\ M =^?_\Gamma N'} M'\}$$

  where $(\Sigma_c'', M', N') \in \chi(\Gamma, \underline{n}, N)^i$ for some list χ of infinite distinct names of meta-variables not in Σ_c, and $0 \leq i \leq k$.

4.7 Related Works and Summary

A complete method for search of proof-trees based on resolution and unification was formulated by Robinson [84] for the first-order logic, and by Huet [45] for the higher-order logic. In type systems, higher-order unification (HOU) algorithms are known for the simply-typed λ-calculus [46], and for the $\lambda\Pi$-calculus of dependent types [28, 80].

For the cube type systems, Dowek [23, 24] re-formulates the unification procedure and generalizes it, as a method of term enumeration. Recently, Cornes [16] proposes an extension of Dowek's method to the Calculus of Constructions with Inductive Types.

Dowek, Hardin and Kirchner [25] propose a first-order presentation of the Huet's HOU-algorithm based on explicit substitutions and typed meta-variables. This algorithm is generalized to solve higher-order equational unification by Kirchner and Ringeissen [55], and restricted to the case of higher-order patterns in [26].

On other hand, Briaud [11] shows how HOU can be considered as a typed narrowing in the λv-calculus of explicit substitutions. Magnusson [62] presents a unification algorithm in the Martin-Löf's type theory with explicit substitutions. This algorithm solves first-order unification problems, but leaves unsolved the flexible-flexible constraints.

The main contribution of this chapter is the presentation of the Dowek's method of proof synthesis, via explicit substitutions and typed meta-variables. In this way, elementary substitutions become instantiations of meta-variables, and the functional coding of scope of unification variables may be avoided. Soundness of the instantiation is guaranteed by the typing system.

Just as in [23, 24], the method presented here is sound and we conjecture its completeness. Thus, it can be seen as a semi-algorithm for ground unification in the calculus of constructions.

Part II

A Calculus of Substitutions and Its Properties

In order to start the study of the meta-theoretical properties of the dependent-typed versions of $\lambda_{\mathcal{L}\Pi}$, as $\mathrm{CC}_{\mathcal{L}}$ and $\lambda\Pi_{\mathcal{L}}$, we have a technical drawback: the $\lambda_{\mathcal{L}\Pi}$-calculus is not confluent on semi-open expressions. In fact, $\lambda_{\mathcal{L}\Pi}$ is not even confluent on ground expressions, as shown by the following example:

$$\begin{array}{ccc} & (S \cdot M_{:N'}) \circ (\uparrow^{Suc(0)} \cdot \mathbf{1}_{:N}) & \\ \text{(Shift0;IdS)} \swarrow & & \searrow \text{(Map;VarCons;ShiftCons;IdS)} \\ S \cdot M_{:N'} & & S \cdot M_{:N} \end{array}$$

where N and N' are assumed to be different ground $\lambda_{\mathcal{L}\Pi}$-normal forms.

This problem is similar to that pointed out by Nederpelt in the λ-calculus extended with the η-rule. In that case, the confluence property holds on terms without type annotations (λ-calculus in Curry style), but does not on pre-terms (λ-calculus in Church style).

Geuvers proposes in [32] a method to prove confluence for $\beta\eta$-reduction on typed λ-terms written in Church style. In fact, Geuvers remarks that to prove the elementary typing properties of the system, it suffices a weak-form of the Church-Rosser property, namely Geuvers' lemma.

In our case, Geuvers' lemma says that if $\Pi_M.N =_{\lambda_{\mathcal{L}\Pi}} \Pi'_M.N'$, then $M =_{\lambda_{\mathcal{L}\Pi}} M'$ and $N =_{\lambda_{\mathcal{L}\Pi}} N'$. In Chapter 5 we present the proof of this lemma, which is based on the confluence of the underlying calculus without type annotations in substitutions, and in the following positive formulation of the above counter-example:

Positive Counter-Example *For any terms M, N, N' and any substitution S,*

$$S \cdot M_{:N'} =_{\lambda_{\mathcal{L}\Pi}} S \cdot M_{:N}$$

In Chapter 6 we show the elementary typing properties of $\mathrm{CC}_{\mathcal{L}}$, i.e. Sort Soundness, Type Uniqueness, Subject Reduction and Typing Soundness. With respect to the usual proofs in type theories without explicit substitutions, additional difficulties arise from occurrences of meta-variables and substitutions in expressions.

Typed versions of $\lambda_{\mathcal{L}\Pi}$ are not strongly normalizing. However, we show in Chapter 7 that they enjoy the weak normalization property. In fact, we show that the reduction of (Beta) followed by a normal reduction of substitutions, is strongly normalizing. The proofs we provide follow straightforwardly that developed by Geuvers in [33] for the calculus of constructions.

We remark that at this stage of the development, we have not the confluence property. In fact, confluence on typed terms is based on the Church-Rosser property which uses weak normalization. We detail the proofs of these two properties: Church-Rosser and Confluence on typed expressions, in the last chapter (Chapter 8) of Part 2.

Chapter 5

Geuvers' Lemma

The Geuvers' lemma is a weak form of the Church-Rosser property which suffices to prove the main typing properties in systems where confluence on terms with typing annotations —i.e in Church style— is not available. Geuvers' technique uses a positive reformulation of the counter-example of non-confluence, and the fact that the underlying calculus of terms without typing annotations —i.e. in Curry style— is confluent.

5.1 The $\lambda_{\mathcal{L}\Pi}$-Calculus in Curry Style

The underlying calculus without type annotation in substitutions of $\lambda_{\mathcal{L}\Pi}$ is called $|\lambda_{\mathcal{L}\Pi}|$. However, in this calculus, abstractions keep their type annotations. Expressions of $|\lambda_{\mathcal{L}\Pi}|$ are defined just as the expressions of $\lambda_{\mathcal{L}\Pi}$, but where substitutions have the following grammar:

$$\textbf{Substitutions} \quad S, T \quad ::= \quad \uparrow^n \mid S \cdot M \mid T \circ S$$

The rewrite system $|\lambda_{\mathcal{L}\Pi}|$ is obtained by dropping type annotations from $\lambda_{\mathcal{L}\Pi}$ as shown in Fig. 5.1. We denote by $|\mathcal{L}_\Pi|$ the rewrite system $|\lambda_{\mathcal{L}\Pi}|$ without (Beta).

The $|\lambda_{\mathcal{L}\Pi}|$-calculus, just as $\lambda_{\mathcal{L}}$, is not confluent on open expressions. But, as we will show, it is confluent on semi-open expressions. The proof of confluence of the $|\lambda_{\mathcal{L}\Pi}|$-calculus uses the Yokouchi-Hikita's lemma (Lemma 2.6).

Lemma 5.1 (Normalization of $|\mathcal{L}_\Pi|$) *The $|\mathcal{L}_\Pi|$-calculus is terminating.*

Proof. We use the semantic labeling technique [92] as in the proof of Proposition 3.11 (Chapter 3). □

Lemma 5.2 (Confluence of $|\mathcal{L}_\Pi|$) *The $|\mathcal{L}_\Pi|$-calculus is confluent on semi-open expressions.*

Proof. We check mechanically, for example using the RRL system [52], that $|\mathcal{L}_\Pi|$ has the following critical pairs:

- **(Id-Clos)**. $[S]M \xleftarrow{|\mathcal{L}_\Pi|^+} [\uparrow^0][S]M \xrightarrow{|\mathcal{L}_\Pi|^+} [\uparrow^0 \circ S]M$.
- **(Clos-Clos)**. $[T' \circ (T \circ S)]M \xleftarrow{|\mathcal{L}_\Pi|^+} [T'][T][S]M \xrightarrow{|\mathcal{L}_\Pi|^+} [(T' \circ T) \circ S]M$.

$(\lambda_{M'}.M\ N)$	$\longrightarrow$	$[\uparrow^0 \cdot N]M$	(Beta)
$[S](\lambda_M.N)$	$\longrightarrow$	$\lambda_{[S]M}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}]N$	(Lambda)
$[S](\Pi_M.N)$	$\longrightarrow$	$\Pi_{[S]M}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}]N$	(Pi)
$[S](M\ N)$	$\longrightarrow$	$([S]M\ [S]N)$	(Application)
$[T][S]M$	$\longrightarrow$	$[T \circ S]M$	(Clos)
$[S \cdot M]\mathbf{1}$	$\longrightarrow$	M	(VarCons)
$[\uparrow^0]M$	$\longrightarrow$	M	(Id)
$T \circ (S \cdot M)$	$\longrightarrow$	$(T \circ S) \cdot [T]M$	(Map)
$S \circ \uparrow^0$	$\longrightarrow$	S	(IdS)
$(S \cdot M) \circ \uparrow^{Suc(n)}$	$\longrightarrow$	$S \circ \uparrow^n$	(ShiftCons)
$\uparrow^m \circ \uparrow^{Suc(n)}$	$\longrightarrow$	$\uparrow^{Suc(m)} \circ \uparrow^n$	(ShiftShift)
$\uparrow^{Suc(0)} \cdot \mathbf{1}$	$\longrightarrow$	$\uparrow^0$	(Shift0)
$\uparrow^{Suc(n)} \cdot [\uparrow^n]\mathbf{1}$	$\longrightarrow$	$\uparrow^n$	(ShiftS)
$[S]\,Type$	$\longrightarrow$	$Type$	(Type)

Figure 5.1: The $|\lambda_{\mathcal{L}\Pi}|$-rewrite system

- **(Shift0-Map)**. $S \xleftarrow{|\mathcal{L}_\Pi|^+} S \circ (\uparrow^{Suc(0)} \cdot \mathbf{1}) \xrightarrow{|\mathcal{L}_\Pi|} (S \circ \uparrow^{Suc(0)}) \cdot [S]\mathbf{1}$.

- **(ShiftS-Map)**. $S \circ \uparrow^n \xleftarrow{|\mathcal{L}_\Pi|} S \circ (\uparrow^{Suc(n)} \cdot [\uparrow^n]\mathbf{1}) \xrightarrow{|\mathcal{L}_\Pi|^+} (S \circ \uparrow^{Suc(n)}) \cdot [S \circ \uparrow^n]\mathbf{1}$.

- **(Lambda-Clos)**. $\lambda_{[T\circ S]M'}.[((((\uparrow^{Suc(0)} \circ T) \cdot \mathbf{1}) \circ (\uparrow^{Suc(0)} \circ S)) \cdot \mathbf{1}]M$
  $\xleftarrow{|\mathcal{L}_\Pi|^+} [T][S](\lambda_{M'}.M) \xrightarrow{|\mathcal{L}_\Pi|^+} \lambda_{[T\circ S]M'}.[(\uparrow^{Suc(0)} \circ (T \circ S)) \cdot \mathbf{1}]M$.

- **(Pi-Clos)**. $\Pi_{[T\circ S]M'}.[((((\uparrow^{Suc(0)} \circ T) \cdot \mathbf{1}) \circ (\uparrow^{Suc(0)} \circ S)) \cdot \mathbf{1}]N'$
  $\xleftarrow{|\mathcal{L}_\Pi|^+} [T][S](\Pi_{M'}.N') \xrightarrow{|\mathcal{L}_\Pi|^+} \Pi_{[T\circ S]M'}.[(\uparrow^{Suc(0)} \circ (T \circ S)) \cdot \mathbf{1}]N'$.

These critical pairs are $|\mathcal{L}_\Pi|$-joinable on semi-open expressions. We use the General Critical Pair lemma (Lemma 2.14) to conclude that $|\mathcal{L}_\Pi|$ is locally confluent on semi-open expressions. Therefore, by Newman's lemma and Lemma 5.1, $|\mathcal{L}_\Pi|$ is confluent on semi-open expressions. □

Now, we define the parallelization of (Beta), namely $B_\parallel$, as

$$\frac{}{x \longrightarrow x}\,(\text{Refl}_\parallel) \qquad \frac{M \longrightarrow M' \qquad N \longrightarrow N'}{\lambda_M.N \longrightarrow \lambda_{M'}.N'}\,(\text{Lambda}_\parallel)$$

$$\frac{M \longrightarrow M' \qquad N \longrightarrow N'}{\Pi_M.N \longrightarrow \Pi_{M'}.N'}\,(\text{Pi}_\parallel) \qquad \frac{M \longrightarrow N \qquad S \longrightarrow T}{[S]M \longrightarrow [T]N}\,(\text{Clos}_\parallel)$$

$$\frac{M \longrightarrow M' \qquad N \longrightarrow N'}{(M\ N) \longrightarrow (M'\ N')}\,(\text{Application}_{\|}) \qquad \frac{M \longrightarrow N \qquad S \longrightarrow T}{S \cdot M \longrightarrow T \cdot N}\,(\text{Cons}_{\|})$$

$$\frac{S \longrightarrow S' \qquad T \longrightarrow T'}{T \circ S \longrightarrow T' \circ S'}\,(\text{Comp}_{\|}) \qquad \frac{M \longrightarrow M' \qquad N \longrightarrow N'}{(\lambda_{M''}.M\ N) \longrightarrow [\uparrow^0 \cdot N']M'}\,(\text{Beta}_{\|})$$

Lemma 5.3 *On semi-open expressions, $|\mathcal{L}_\Pi|$ YH-commutes over $B_{\|}$, i.e. if x reduces in one $|\mathcal{L}_\Pi|$-step to y, and in one $B_{\|}$-step to z, then there exists w such that $y \xrightarrow{|\mathcal{L}_\Pi|^* B_{\|} |\mathcal{L}_\Pi|^*} w$ and $z \xrightarrow{|\mathcal{L}_\Pi|^*} w$.*

Proof. The proof follows straightforwardly the proof of Lemma 2.40. □

Theorem 5.4 (Confluence) *The $|\lambda_{\mathcal{L}\Pi}|$-calculus is confluent on semi-open expressions.*

Proof. We verify that $|\mathcal{L}_\Pi|$ and $B_{\|}$ satisfy the conditions of the Yokouchi-Hikita's lemma (Def. 2.5 and Lemma 2.6) on semi-open expressions, e.g.

1. $|\mathcal{L}_\Pi|$ is terminating and confluent (Lemma 5.1 and Lemma 5.2).

2. $B_{\|}$ is strongly confluent, since (Beta) by itself is a left linear system with no critical pairs (c.f. [48]).

3. $|\mathcal{L}_\Pi|$ YH-commutes over $B_{\|}$ (Lemma 5.3).

Therefore, $|\mathcal{L}_\Pi|^* B_{\|} |\mathcal{L}_\Pi|^*$ is confluent on open expressions.

Note that $|\lambda_{\mathcal{L}\Pi}| \subseteq |\mathcal{L}_\Pi|^* B_{\|} |\mathcal{L}_\Pi|^* \subseteq {\lambda_{\mathcal{L}\Pi}}^*$. Let x be a semi-open expressions, if $x \xrightarrow{{\lambda_{\mathcal{L}\Pi}}^*} y$ and $x \xrightarrow{{\lambda_{\mathcal{L}\Pi}}^*} z$, then there exists w such that $y \xrightarrow{(|\mathcal{L}_\Pi|^* B_{\|} |\mathcal{L}_\Pi|^*)^*} w$ and $z \xrightarrow{(|\mathcal{L}_\Pi|^* B_{\|} |\mathcal{L}_\Pi|^*)^*} w$. So, $y \xrightarrow{{\lambda_{\mathcal{L}\Pi}}^*} w$ and $z \xrightarrow{{\lambda_{\mathcal{L}\Pi}}^*} w$. □

5.2 Erasing Typing Annotations and Decorating Expressions

Definition 5.5 *The erasing map $|.| : \lambda_{\mathcal{L}\Pi} \to |\lambda_{\mathcal{L}\Pi}|$ is defined as follows:*

$$\begin{array}{lcl}
|x| & = & x \quad \text{if } x \in \{\mathbf{1}, \mathit{Type}, \mathit{Kind}\} \text{ or } x \text{ is a meta-variable} \\
|\Pi_M.N| & = & \Pi_{|M|}.|N| \\
|\lambda_M.N| & = & \lambda_{|M|}.|N| \\
|(M\ N)| & = & (|M|\ |N|) \\
|[S]M| & = & [|S|]|M| \\
|\uparrow^n| & = & \uparrow^n \\
|T \circ S| & = & |T| \circ |S| \\
|S \cdot M_{:N}| & = & |S| \cdot |M|
\end{array}$$

Definition 5.6 *Let M be a $\lambda_{\mathcal{L}\Pi}$-term, the annotation map $(.)^{\underline{M}} : |\lambda_{\mathcal{L}\Pi}| \to \lambda_{\mathcal{L}\Pi}$ is defined as follows:*

$$\begin{array}{lcl}
x^{\underline{M}} & = & x \quad \text{if } x \in \{\mathbf{1}, \mathit{Type}, \mathit{Kind}\} \text{ or } x \text{ is a meta-variable} \\
(\Pi_{M'}.N')^{\underline{M}} & = & \Pi_{M'^{\underline{M}}}.N'^{\underline{M}} \\
(\lambda_{M'}.N')^{\underline{M}} & = & \lambda_{M'^{\underline{M}}}.N'^{\underline{M}} \\
(M'\ N')^{\underline{M}} & = & (M'^{\underline{M}}\ N'^{\underline{M}}) \\
([S]N)^{\underline{M}} & = & [S^{\underline{M}}]N^{\underline{M}} \\
(\uparrow^n)^{\underline{M}} & = & \uparrow^n \\
(T \circ S)^{\underline{M}} & = & T^{\underline{M}} \circ S^{\underline{M}} \\
(S \cdot M')^{\underline{M}} & = & S^{\underline{M}} \cdot M'^{\underline{M}}_{:M}
\end{array}$$

Following [32], we use a positive formulation of the counter-example of confluence on pre-expressions.

Lemma 5.7 (Positive Counter-example) *Let x and y be arbitrary $\lambda_{\mathcal{L}\Pi}$-expressions, if $|x| = |y|$, then $x =_{\mathcal{L}\Pi} y$, and therefore, $x =_{\lambda_{\mathcal{L}\Pi}} y$.*

Proof. By definition of $|.|$, we have that if $|x| = |y|$, then x and y have the same principal constructor. Now, we proceed by structural induction on x. If $x = \lambda_{M'}.M$, $y = \lambda_{N'}.N$ and $|x| = |y|$, then by definition, $\lambda_{|M'|}.|M| = \lambda_{|N'|}.|N|$ and so, $|M'| = |N'|$ and $|M| = |N|$. By induction hypothesis, $M' =_{\mathcal{L}\Pi} N'$ and $M =_{\mathcal{L}\Pi} N$. The only interesting case is when $x = S \cdot M_{:M'}$ and $y = T \cdot N_{:N'}$. We get by induction hypothesis:

$$M =_{\mathcal{L}\Pi} N \tag{5.1}$$

$$S =_{\mathcal{L}\Pi} T \tag{5.2}$$

Note that the function $|.|$ erases typing annotations from substitutions, thus we do not have $M' =_{\mathcal{L}\Pi} N'$. However, using the counter-example, we have

$$S \cdot M_{:N'} \xleftarrow{\mathcal{L}\Pi^*} (S \cdot M_{:N'}) \circ (\uparrow^{Suc(0)} \cdot \mathbf{1}_{:M'}) \xrightarrow{\mathcal{L}\Pi^*} S \cdot M_{:M'}$$

We conclude with equations 5.1 and 5.2, that $x = S \cdot M_{:M'} =_{\mathcal{L}\Pi} S \cdot M_{:N'} =_{\mathcal{L}\Pi} T \cdot N_{:N'} = y$. □

Lemma 5.8 *Let x and y be $\lambda_{\mathcal{L}\Pi}$-expressions and w be a $|\lambda_{\mathcal{L}\Pi}|$-expression, then*

1. *(a) If $x \xrightarrow{\lambda_{\mathcal{L}\Pi}} y$, then ($|x| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} |y|$ or $|x| = |y|$), and (b) if $x \xrightarrow{\mathcal{L}\Pi} y$, then ($|x| \xrightarrow{|\mathcal{L}\Pi|} |y|$ or $|x| = |y|$).*
2. *(a) If $|x| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} w$, then there exists w' in $\lambda_{\mathcal{L}\Pi}$ such that $x \xrightarrow{\lambda_{\mathcal{L}\Pi}} w'$ and $|w'| = w$, and (b) if $|x| \xrightarrow{|\mathcal{L}\Pi|} w$, then there exists w' in $\lambda_{\mathcal{L}\Pi}$ such that $x \xrightarrow{\mathcal{L}\Pi} w'$ and $|w'| = w$.*

Proof. We only detail the first cases on both properties since the second ones are similar.

1. By structural induction on x. If the redex on x is erased by the transformation, then $|x| = |y|$. Otherwise the same redex can still be done in $|\lambda_{\mathcal{L}\Pi}|$. We show the case when x is an application, the rest of cases are solved in the same way.
   - $x = (\lambda_{M'}.M\ N)$ and $y = [\uparrow^0 \cdot N_{:M'}]M$. By definition, $|x| = (\lambda_{|M'|}.|M|\ |N|)$ and $|y| = [\uparrow^0 \cdot |N|]|M|$. But also we have $|x| \xrightarrow{\text{(Beta)}} |y|$ in $|\lambda_{\mathcal{L}\Pi}|$.

- $x = (M\ N)$, $y = (M'\ N)$ and $M \xrightarrow{\lambda_{\mathcal{L}\Pi}} M'$. By definition, $|x| = (|M|\ |N|)$ and $|y| = (|M'|\ |N|)$. By induction hypothesis, $|M| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} |M'|$ or $|M| = |M'|$. If $|M| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} |M'|$ then $(|M|\ |N|) \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} (|M'|\ |N|)$, i.e. $|x| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} |y|$. Otherwise, if $|M| = |M'|$, then $(|M|\ |N|) = (|M'|\ |N|)$, i.e. $|x| = |y|$. The case $y = (M\ N')$ and $N \xrightarrow{\lambda_{\mathcal{L}\Pi}} N'$ is similar.

2. By induction on the depth of the $|\lambda_{\mathcal{L}\Pi}|$-redex reduced in x. Since $\lambda_{\mathcal{L}\Pi}$ is a left linear system and the transformation removes only typing annotations from substitutions, we can verify that a $|\lambda_{\mathcal{L}\Pi}|$-redex in $|x|$ also is a $\lambda_{\mathcal{L}\Pi}$-redex in x. Thus, we can find w' in $\lambda_{\mathcal{L}\Pi}$, such that $|w'| = w$. For example, we show when x is an application.

   - $x = (\lambda_{M'}.M\ N)$ and $|x| \xrightarrow{\text{(Beta)}} [\uparrow^0 \cdot |N|]|M| = w$. But also, $x \xrightarrow{\text{(Beta)}} [\uparrow^0 \cdot N_{:M'}]M = w'$. By definition, $|w'| = [\uparrow^0 \cdot |N|]|M| = w$.
   - $x = (M\ N)$ and $|x| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} M'\ |N| = w$, where $|M| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} M'$. By induction hypothesis, there exists M'' in $\lambda_{\mathcal{L}\Pi}$ such that $M \xrightarrow{\lambda_{\mathcal{L}\Pi}} M''$ and $|M''| = M'$. Therefore, $(M\ N) \xrightarrow{\lambda_{\mathcal{L}\Pi}} (M''\ N) = w'$. By definition, $|w'| = (|M''|\ |N|) = (M'\ |N|) = w$. The case $x = (M\ N)$ and $|x| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} (|M|\ N') = w$ is similar.

□

Corollary 5.9 *(a) If x is a $\lambda_{\mathcal{L}\Pi}$-normal form then $|x|$ is a $|\lambda_{\mathcal{L}\Pi}|$-normal form, and (b) if x is a $\mathcal{L}_\Pi$-normal form then $|x|$ is a $|\mathcal{L}_\Pi|$-normal form.*

Lemma 5.10 *Let x and y be $\lambda_{\mathcal{L}\Pi}$-expressions and w be a $|\lambda_{\mathcal{L}\Pi}|$-expression, then*

1. *(a) If $x \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} y$ then $|x| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} |y|$, and (b) if $x \xrightarrow{\mathcal{L}_\Pi^*} y$ then $|x| \xrightarrow{|\mathcal{L}_\Pi|^*} |y|$.*
2. *(a) If $|x| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} w$, then there exists w' in $\lambda_{\mathcal{L}\Pi}$ such that $x \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} w'$ and $|w'| = w$, and (b) if $|x| \xrightarrow{|\mathcal{L}_\Pi|^*} w$, then there exists w' in $\lambda_{\mathcal{L}\Pi}$ such that $x \xrightarrow{\mathcal{L}_\Pi^*} w'$ and $|w'| = w$.*

Proof. By induction on the length of derivations and Lemma 5.8. □

Lemma 5.11 *Let x be an arbitrary $|\lambda_{\mathcal{L}\Pi}|$-expression and M be a $\lambda_{\mathcal{L}\Pi}$-term, $x = |x\underline{M}|$.*

Proof. By structural induction on x. □

Lemma 5.12 *Let x be an arbitrary $\lambda_{\mathcal{L}\Pi}$-expression and M be a $\lambda_{\mathcal{L}\Pi}$-term, $x =_{\lambda_{\mathcal{L}\Pi}} |x|\underline{M}$.*

Proof. By Lemma 5.11, $|x| = ||x|\underline{M}|$, then by Lemma 5.7, $x =_{\lambda_{\mathcal{L}\Pi}} |x|\underline{M}$. □

Lemma 5.13 *Let x and y be arbitrary $|\lambda_{\mathcal{L}\Pi}|$-expressions and M be a $\lambda_{\mathcal{L}\Pi}$-term, if $x \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} y$, then $x\underline{M} =_{\lambda_{\mathcal{L}\Pi}} y\underline{M}$. Therefore, if $x \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} y$, then $x\underline{M} =_{\lambda_{\mathcal{L}\Pi}} y\underline{M}$.*

Proof. By induction on the depth of the $|\lambda_{\mathcal{L}\Pi}|$-redex reduced in x. If x is a $|\lambda_{\mathcal{L}\Pi}|$-normal form then the conclusion is trivial. Otherwise,

- $x = (\lambda_{M_1}.M_2\ N)$ and $y = [\uparrow^0 \cdot N]M_2$. By definition,

$$x^{\underline{M}} = (\lambda_{M_1^{\underline{M}}}.M_2^{\underline{M}}\ N^{\underline{M}}) \tag{5.3}$$

and

$$y^{\underline{M}} = [\uparrow^0 \cdot N^{\underline{M}}_{:M}]M_2^{\underline{M}} \tag{5.4}$$

We have $(\lambda_{M_1^{\underline{M}}}.M_2^{\underline{M}}\ N^{\underline{M}}) \xrightarrow{\lambda_{\mathcal{L}\Pi}} [\uparrow^0 \cdot N^{\underline{M}}_{:M_1^{\underline{M}}}]M_2^{\underline{M}}$. By Lemma 5.7, $\uparrow^0 \cdot N^{\underline{M}}_{:M_1^{\underline{M}}} =_{\lambda_{\mathcal{L}\Pi}} \uparrow^0 \cdot N^{\underline{M}}_{:M}$. Therefore, $x^{\underline{M}} =_{\lambda_{\mathcal{L}\Pi}} [\uparrow^0 \cdot N^{\underline{M}}_{:M_1^{\underline{M}}}]M_2^{\underline{M}} =_{\lambda_{\mathcal{L}\Pi}} [\uparrow^0 \cdot N^{\underline{M}}_{:M}]M_2^{\underline{M}} = y^{\underline{M}}$.

- $x = (M_1\ N)$, $y = (M_2\ N)$ and $M_1 \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} M_2$. By definition,

$$x^{\underline{M}} = (M_1^{\underline{M}}\ N^{\underline{M}}) \tag{5.5}$$

and

$$y^{\underline{M}} = (M_2^{\underline{M}}\ N^{\underline{M}}) \tag{5.6}$$

The induction hypothesis says that $M_1^{\underline{M}} =_{\lambda_{\mathcal{L}\Pi}} M_2^{\underline{M}}$. Therefore, $x^{\underline{M}} =_{\lambda_{\mathcal{L}\Pi}} y^{\underline{M}}$. The case $y = (M_1\ N')$ and $N \xrightarrow{|\lambda_{\mathcal{L}\Pi}|} N'$ is similar.

- The rest of cases are solved in the same way.

□

5.3 Geuvers' Lemma

Theorem 5.14 *Let M_1, N_1, M_2 and N_2 be semi-open $\lambda_{\mathcal{L}\Pi}$-expressions,*

1. *if $\Pi_{M_1}.N_1 =_{\lambda_{\mathcal{L}\Pi}} \Pi_{M_2}.N_2$, then $M_1 =_{\lambda_{\mathcal{L}\Pi}} M_2$ and $N_1 =_{\lambda_{\mathcal{L}\Pi}} N_2$, and*
2. *if $M_1 =_{\lambda_{\mathcal{L}\Pi}} M_2$, where M_2 is a $\lambda_{\mathcal{L}\Pi}$-normal form, then there exists M' in $\lambda_{\mathcal{L}\Pi}$ such that $M_1 \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} M'$ and $|M'| = |M_2|$.*

Proof.

1. By Lemma 5.10(1) and definition of $|.|$, $\Pi_{|M_1|}.|N_1| =_{|\lambda_{\mathcal{L}\Pi}|} \Pi_{|M_2|}.|N_2|$. Since $|\lambda_{\mathcal{L}\Pi}|$ is confluent on semi-open expressions (Theorem 5.4), there exists M' in $|\lambda_{\mathcal{L}\Pi}|$ such that $\Pi_{|M_1|}.|N_1| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} M'$ and $\Pi_{|M_2|}.|N_2| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} M'$. But there exists no $|\lambda_{\mathcal{L}\Pi}|$-redex with the form $\Pi_x.y$, so M' has the form $\Pi_M.N$ where $|M_1| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} M$, $|N_1| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} N$, $|M_2| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} M$ and $|N_2| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} N$. By Lemma 5.12 and Lemma 5.13, for any $\lambda_{\mathcal{L}\Pi}$-term N', $M_1 =_{\lambda_{\mathcal{L}\Pi}} |M_1|^{\underline{N'}} =_{\lambda_{\mathcal{L}\Pi}} M^{\underline{N'}}$, $N_1 =_{\lambda_{\mathcal{L}\Pi}} |N_1|^{\underline{N'}} =_{\lambda_{\mathcal{L}\Pi}} N^{\underline{N'}}$, $M_2 =_{\lambda_{\mathcal{L}\Pi}} |M_2|^{\underline{N'}} =_{\lambda_{\mathcal{L}\Pi}} M^{\underline{N'}}$ and $N_2 =_{\lambda_{\mathcal{L}\Pi}} |N_2|^{\underline{N'}} =_{\lambda_{\mathcal{L}\Pi}} N^{\underline{N'}}$. Therefore, $M_1 =_{\lambda_{\mathcal{L}\Pi}} M_2$ and $N_1 =_{\lambda_{\mathcal{L}\Pi}} N_2$.
2. By Lemma 5.10(1) and definition of $|.|$, $|M_1| =_{|\lambda_{\mathcal{L}\Pi}|} |M_2|$. Since $|\lambda_{\mathcal{L}\Pi}|$ is confluent on semi-open expressions (Theorem 5.4), there exists M in $|\lambda_{\mathcal{L}\Pi}|$ such that $|M_1| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} M$ and $|M_2| \xrightarrow{|\lambda_{\mathcal{L}\Pi}|^*} M$. But M_2 is a $\lambda_{\mathcal{L}\Pi}$-normal form, then by Corollary 5.9, $|M_2|$ is a $|\lambda_{\mathcal{L}\Pi}|$-normal form. Therefore, $|M_2| = M$. By Lemma 5.10(2), there exists M' such that $M_1 \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} M'$ and $|M'| = M$. Therefore, $|M'| = |M_2|$.

□

Chapter 6

Elementary Properties

In this chapter, we study the elementary typing properties of $\mathrm{CC}_{\mathcal{L}}$. The case of $\lambda\Pi_{\mathcal{L}}$ is either similar or simpler [69].

We recall the typing rules of the system $\mathrm{CC}_{\mathcal{L}}$.

Notation: We use the lowercase letter s to range over the set of sorts $\{\mathit{Kind}, \mathit{Type}\}$.

Valid Signatures and Contexts

$$\frac{}{\vdash \mathit{nil} :\cdot \mathit{nil}}\ (\text{Empty}) \qquad \frac{\Sigma :\cdot \Gamma \vdash M : s}{\vdash \Sigma :\cdot \Gamma.M}\ (\text{Var-Decl})$$

$$\frac{\vdash \Sigma :\cdot \Gamma \quad X \notin \Sigma}{\vdash \Sigma.\ X :_{\Gamma} \mathit{Kind}}\ (\text{Metavar-Decl}_1) \qquad \frac{\Sigma :\cdot \Gamma \vdash M : s \quad X \notin \Sigma}{\vdash \Sigma.\ X :_{\Gamma} M}\ (\text{Metavar-Decl}_2)$$

Valid Terms

$$\frac{\vdash \Sigma :\cdot \Gamma}{\Sigma :\cdot \Gamma \vdash \mathit{Type} : \mathit{Kind}}\ (\text{Type}) \qquad \frac{\Sigma :\cdot \Gamma.M \vdash N : s}{\Sigma :\cdot \Gamma \vdash \Pi_M.N : s}\ (\text{Prod})$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Delta \vdash M : \mathit{Kind}}{\Sigma :\cdot \Gamma \vdash [S]M : \mathit{Kind}}\ (\text{Clos-Kind}) \qquad \frac{\vdash \Sigma :\cdot \Gamma.M}{\Sigma :\cdot \Gamma.M \vdash \mathbf{1} : [\uparrow^{Suc(0)}]M}\ (\text{Var})$$

$$\frac{\Sigma :\cdot \Gamma \vdash M : \Pi_{M_1}.M_2 \quad \Sigma :\cdot \Gamma \vdash N : M_1}{\Sigma :\cdot \Gamma \vdash (M\ N) : [\uparrow^0 \cdot N_{:M_1}]M_2}\ (\text{Appl})$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Delta \vdash M : N \quad \Sigma :\cdot \Delta \vdash N : s}{\Sigma :\cdot \Gamma \vdash [S]M : [S]N}\ (\text{Clos})$$

$$\frac{\vdash \Sigma :\cdot \Gamma \quad X :_{\Delta} M \in \Sigma \quad \Delta =_{\lambda_{\mathcal{L}\Pi}} \Gamma}{\Sigma :\cdot \Gamma \vdash X : M}\ (\text{Metavar})$$

$$\frac{\Sigma :\cdot \Gamma.M_1 \vdash M_2 : N \quad \Sigma :\cdot \Gamma \vdash \Pi_{M_1}.N : s}{\Sigma :\cdot \Gamma \vdash \lambda_{M_1}.M_2 : \Pi_{M_1}.N} \text{(Abs)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash M : M_1 \quad \Sigma :\cdot \Gamma \vdash M_2 : s \quad M_1 =_{\lambda_{\mathcal{L}\Pi}} M_2}{\Sigma :\cdot \Gamma \vdash M : M_2} \text{(Conv)}$$

Valid Substitutions

$$\frac{\vdash \Sigma :\cdot \Gamma}{\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma} \text{(Id)} \qquad \frac{\vdash \Sigma :\cdot \Gamma.M \quad \Sigma :\cdot \Gamma \vdash \uparrow^n \triangleright \Delta}{\Sigma :\cdot \Gamma.M \vdash \uparrow^{Suc(n)} \triangleright \Delta} \text{(Shift)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta' \quad \Sigma :\cdot \Delta' \vdash T \triangleright \Delta}{\Sigma :\cdot \Gamma \vdash S \circ T \triangleright \Delta} \text{(Comp)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Gamma \vdash M : [S]N \quad \Sigma :\cdot \Delta \vdash N : s}{\Sigma :\cdot \Gamma \vdash S \cdot M_{:N} \triangleright \Delta.N} \text{(Cons)}$$

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \vdash \Sigma :\cdot \Delta' \quad \Delta =_{\lambda_{\mathcal{L}\Pi}} \Delta'}{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta'} \text{(Conv-Subs)}$$

The first section presents some basic lemmas. The remaining sections present the elementary properties of $\mathrm{CC}_{\mathcal{L}}$: Sort Soundness, Type Uniqueness, Subject Reduction and Typing Soundness.

6.1 Basic Lemmas

The following lemmas simplify the proofs to come. Almost all of them are proved by induction on typing derivations.

Lemma 6.1 *If we denote the prefix relation of lists by $\preceq$, then the following typing rules are admissible.*

$$\frac{\vdash \Sigma :\cdot \Delta \quad \Gamma \preceq \Delta}{\vdash \Sigma :\cdot \Gamma} \text{(Undeclare-Var)} \qquad \frac{\vdash \Sigma' \quad \Sigma \preceq \Sigma'}{\vdash \Sigma} \text{(Undeclare-Metavar)}$$

Proof. By structural induction on Δ and Σ'. □

Lemma 6.2 (Sub-Derivation Property)

1. *Every typing derivation of $\Sigma :\cdot \Gamma \vdash M : N$ and $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$ has a proof of $\vdash \Sigma :\cdot \Gamma$ as sub-derivation.*

2. *Every typing derivation of $\vdash \Sigma :\cdot \Gamma.M$ has a proof of $\Sigma :\cdot \Gamma \vdash M : s$ as sub-derivation.*

Proof. By simultaneous induction on typing derivations. □

Lemma 6.3 (Clos-Sorts) *The following typing rule is admissible.*

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Delta \vdash M : s}{\Sigma :\cdot \Gamma \vdash [S]M : s}\,(\text{Clos-Sorts})$$

Proof. By case analysis on s. If $s = \mathit{Kind}$ then this rule is (Clos-Kind). Otherwise, $s = \mathit{Type}$, and we have $\Sigma :\cdot \Gamma \vdash [S]M : [S]\mathit{Type}$. Therefore we have the typing derivation

$$\frac{\Sigma :\cdot \Gamma \vdash [S]M : [S]\mathit{Type} \quad \Sigma :\cdot \Gamma \vdash \mathit{Type} : \mathit{Kind} \quad [S]\mathit{Type} =_{\lambda\mathcal{L}\Pi} \mathit{Type}}{\Sigma :\cdot \Gamma \vdash [S]M : \mathit{Type}}\,(\text{Conv})$$

□

Lemma 6.4 (Inversion)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, *then*
   - *if* $M = \mathbf{1}$, *then* $\Gamma = \Gamma'.N'$ *and* $N =_{\lambda\mathcal{L}\Pi} [\uparrow^{Suc(0)}]N'$,
   - *if* $M = \mathit{Type}$, *then* $N =_{\lambda\mathcal{L}\Pi} \mathit{Kind}$,
   - *if* $M = X$ *(a meta-variable), then* $X :_{\Delta} N' \in \Sigma$, $\Gamma =_{\lambda\mathcal{L}\Pi} \Delta$ *and* $N =_{\lambda\mathcal{L}\Pi} N'$,
   - *if* $M = \Pi_{M_1}.M_2$, *then* $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : s$ *and* $s =_{\lambda\mathcal{L}\Pi} N$,
   - *if* $M = \lambda_{M_1}.M_2$, *then* $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : N'$, $\Sigma :\cdot \Gamma \vdash \Pi_{M_1}.N' : s$ *and* $N =_{\lambda\mathcal{L}\Pi} \Pi_{M_1}.N'$,
   - *if* $M = (M_1\ M_2)$, *then* $\Sigma :\cdot \Gamma \vdash M_1 : \Pi_{N_1}.N_2$, $\Sigma :\cdot \Gamma \vdash M_2 : N_1$ *and* $N =_{\lambda\mathcal{L}\Pi} [\uparrow^0 \cdot M_{2:N_1}]N_2$, *and*
   - *if* $M = [T]M'$, *then* $\Sigma :\cdot \Gamma \vdash T \triangleright \Delta$, $\Sigma :\cdot \Delta \vdash M' : N'$ *and* ($N' = \mathit{Kind}$, $N =_{\lambda\mathcal{L}\Pi} \mathit{Kind}$ *or* $\Sigma :\cdot \Delta \vdash N' : s$, $N =_{\lambda\mathcal{L}\Pi} [T]N'$).
2. *If* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then*
   - *if* $S = \uparrow^0$, *then* $\Gamma =_{\lambda\mathcal{L}\Pi} \Delta$,
   - *if* $S = \uparrow^{Suc(n)}$, *then* $\Gamma = \Gamma'.M'$, $\Sigma :\cdot \Gamma' \vdash \uparrow^n \triangleright \Delta'$ *and* $\Delta =_{\lambda\mathcal{L}\Pi} \Delta'$,
   - *if* $S = T \cdot M'_{:N'}$, *then* $\Sigma :\cdot \Gamma \vdash T \triangleright \Delta'$, $\Sigma :\cdot \Gamma \vdash M' : [T]N'$, $\Sigma :\cdot \Delta' \vdash N' : s$ *and* $\Delta =_{\lambda\mathcal{L}\Pi} \Delta'.N'$, *and*
   - *if* $S = T \circ S'$, *then* $\Sigma :\cdot \Gamma \vdash T \triangleright \Delta'$, $\Sigma :\cdot \Delta' \vdash S' \triangleright \Delta''$ *and* $\Delta'' =_{\lambda\mathcal{L}\Pi} \Delta$.

Proof. By simultaneous induction on typing derivations. If the last applied rule is (Conv) or (Conv-Subs), then we conclude by using the induction hypothesis. The other cases are solved by using the premises of the rules. □

Lemma 6.5 *If* Σ *is a valid signature, i.e.* $\vdash \Sigma$, *then all the meta-variables declared in* Σ *are different.*

Proof. By structural induction on Σ. □

Lemma 6.6 *Let* X *be a meta-variable, if* $\Sigma :\cdot \Gamma \vdash X : M$ *and* $\Sigma :\cdot \Gamma' \vdash X : M'$, *then* $\Gamma =_{\lambda\mathcal{L}\Pi} \Gamma'$ *and* $M =_{\lambda\mathcal{L}\Pi} M'$.

Proof. By Lemma 6.4, we have $X{:}_{\Delta_1} M_1 \in \Sigma$, $\Gamma =_{\lambda_{\mathcal{LP}}} \Delta_1$, $M =_{\lambda_{\mathcal{LP}}} M_1$, $X{:}_{\Delta_2} M_2 \in \Sigma$, $\Gamma' =_{\lambda_{\mathcal{LP}}} \Delta_2$ and $M' =_{\lambda_{\mathcal{LP}}} M_2$. By Lemma 6.5, we have $\Delta_1 = \Delta_2$ and $M_1 = M_2$. Therefore, $\Gamma =_{\lambda_{\mathcal{LP}}} \Gamma'$ and $M =_{\lambda_{\mathcal{LP}}} M'$. □

Lemma 6.7 *Let Σ and Σ' be two signatures such that $\Sigma \preceq \Sigma'$ and $\vdash \Sigma'$, then*

1. *if $\vdash \Sigma :\cdot \Gamma$, then $\vdash \Sigma' :\cdot \Gamma$,*
2. *if $\Sigma :\cdot \Gamma \vdash M : N$, then $\Sigma' :\cdot \Gamma \vdash M : N$, and*
3. *if $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, then $\Sigma' :\cdot \Gamma \vdash S \triangleright \Delta$.*

Proof. By simultaneous induction on typing derivations. □

Lemma 6.8 *If $\vdash \Sigma :\cdot \Gamma$ and $X{:}_{\Delta} M \in \Sigma$, then $M = Kind$ or $\Sigma :\cdot \Delta \vdash M : s$.*

Proof. We have the hypotheses:

1. $\vdash \Sigma$, by (Undeclare-Var) applied to hypothesis $\vdash \Sigma :\cdot \Gamma$.
2. (a) $\vdash \Sigma'$. $X{:}_{\Delta} M$ and (b) $\Sigma' \preceq \Sigma$, by (Undeclare-Metavar) applied to (1) and $X{:}_{\Delta} M \in \Sigma$.

Now, we analyze the last derivation of (2-a). If it is (Metavar-Decl$_1$), then $M = Kind$. Otherwise, it is (Metavar-Decl$_2$). In this case, we have $\vdash \Sigma'$. $X{:}_{\Delta} M$, and we conclude with Lemma 6.7, (1) and (2-b), that $\Sigma :\cdot \Delta \vdash M : s$. □

Lemma 6.9 (Kind Inversion) *If $\Sigma :\cdot \Gamma \vdash M : Kind$, then $M = Type$ or $M = X$ (where X is a meta-variable) or ($M = \Pi_{M_1}.M_2$, $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : Kind$) or ($M = [S]M'$, $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$ and $\Sigma :\cdot \Delta \vdash M' : Kind$).*

Proof. We check the last applied rule in the typing derivation. By Remark 3.15, the rule (Conv) cannot be the last applied rule. Hence, the only possible rules are (Type), (Prod), (Metavar) and (Clos-Kind). The result is obtained from premises of these rules. □

6.2 Sort Soundness

Sort Soundness property says that if $\Sigma :\cdot \Gamma \vdash M : N$ is a valid judgment, then $N = Kind$, or well the type of N is a sort, i.e. $\Sigma :\cdot \Gamma \vdash N : s$.

We need the following lemmas.

Lemma 6.10 (Equivalent Contexts) *If $\vdash \Sigma :\cdot \Gamma'$ and $\Gamma =_{\lambda_{\mathcal{LP}}} \Gamma'$, then*

1. *if $\Sigma :\cdot \Gamma \vdash M : N$, then $\Sigma :\cdot \Gamma' \vdash M : N$, and*
2. *if $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, then $\Sigma :\cdot \Gamma' \vdash S \triangleright \Delta$.*

Proof. By simultaneous induction on typing derivations. If the last applied rule is (Type), then $M = Type$ and $N = Kind$. In this case, the result is obtained directly by applying (Type) to hypothesis $\vdash \Sigma :\cdot \Gamma'$. If the last applied rule is one of (Clos-Kind), (Appl), (Clos), (Conv), (Cons), (Comp) or (Conv-Subs), then we apply the induction hypothesis to the premises. The only interesting cases are:

- (Prod). The last derivation has the form:

$$\frac{\Sigma :\cdot \Gamma.M_1 \vdash M_2 : s}{\Sigma :\cdot \Gamma \vdash \Pi_{M_1}.M_2 : s}\,(\text{Prod})$$

where $M = \Pi_{M_1}.M_2$ and $N = s$. We have the hypotheses:

1. $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : s$, premise of (Prod).
2. $\Sigma :\cdot \Gamma \vdash M_1 : s$ is a sub-derivation, by Lemma 6.2(2) applied to (1).
3. $\Sigma :\cdot \Gamma' \vdash M_1 : s$, by induction hypothesis applied to (2).
4. $\vdash \Sigma :\cdot \Gamma'.M_1$, by (Var-Decl) applied to (3).
5. $\Sigma :\cdot \Gamma'.M_1 \vdash M_2 : s$, by the induction hypothesis applied to (4) and the hypothesis $\Gamma.M_1 =_{\lambda_{\mathcal{L}\Pi}} \Gamma'.M_1$.

We conclude with (Prod) applied to (5) that $\Sigma :\cdot \Gamma' \vdash \Pi_{M_1}.M_2 : s$.

- (Var). The last derivation has the form:

$$\frac{\vdash \Sigma :\cdot \Gamma_1.M_1}{\Sigma :\cdot \Gamma_1.M_1 \vdash \mathbf{1} : [\uparrow^{Suc(0)}]M_1}\,(\text{Var})$$

where $\Gamma = \Gamma_1.M_1$, $M = \mathbf{1}$ and $N = [\uparrow^{Suc(0)}]M_1$. We have the hypotheses:

1. $\vdash \Sigma :\cdot \Gamma_1.M_1$, premise of (Var).
2. (a) $\Gamma' = \Gamma_2.M_2$, where $\Gamma_1 =_{\lambda_{\mathcal{L}\Pi}} \Gamma_2$ and $M_1 =_{\lambda_{\mathcal{L}\Pi}} M_2$, and (b) $\vdash \Sigma :\cdot \Gamma_2.M_2$, hypothesis of lemma.
3. $[\uparrow^{Suc(0)}]M_1 =_{\lambda_{\mathcal{L}\Pi}} [\uparrow^{Suc(0)}]M_2$, by (2-a).
4. $\vdash \Sigma :\cdot \Gamma_2$, by (Undeclare-Var) applied to (2-b).
5. $\Sigma :\cdot \Gamma_2 \vdash \uparrow^0 \triangleright \Gamma_2$, by (Id) applied to (4).
6. $\Sigma :\cdot \Gamma_2.M_2 \vdash \uparrow^{Suc(0)} \triangleright \Gamma_2$, by (Shift) applied to (2-b) and (5).
7. $\Sigma :\cdot \Gamma_1 \vdash M_1 : s$ is a sub-derivation of $\vdash \Sigma :\cdot \Gamma_1.M_1$, by Lemma 6.2(2) applied to (1).
8. $\Sigma :\cdot \Gamma_2 \vdash M_1 : s$, by induction hypothesis applied to (2-a), (4) and (7).
9. $\Sigma :\cdot \Gamma_2.M_2 \vdash [\uparrow^{Suc(0)}]M_1 : s$, by Lemma 6.3 applied to (6) and (8).
10. $\Sigma :\cdot \Gamma_2.M_2 \vdash \mathbf{1} : [\uparrow^{Suc(0)}]M_2$, by (Var) applied to (2-b).

We conclude with (Conv) applied to (10), (9) and (3) that $\Sigma :\cdot \Gamma_2.M_2 \vdash \mathbf{1} : [\uparrow^{Suc(0)}]M_1$.

- (Metavar). The last derivation has the form:

$$\frac{\vdash \Sigma :\cdot \Gamma \quad X :_{\Delta} N \in \Sigma \quad \Delta =_{\lambda_{\mathcal{L}\Pi}} \Gamma}{\Sigma :\cdot \Gamma \vdash X : N}\,(\text{Metavar})$$

where $M = X$ (X is a meta-variable). We have the hypotheses:

1. (a) $\vdash \Sigma :\cdot \Gamma$, (b) $X :_{\Delta} N \in \Sigma$ and (c) $\Delta =_{\lambda_{\mathcal{L}\Pi}} \Gamma$, premises of (Metavar).
2. $\Delta =_{\lambda_{\mathcal{L}\Pi}} \Gamma'$, by (1-c) and hypothesis $\Gamma =_{\lambda_{\mathcal{L}\Pi}} \Gamma'$.

3. $\vdash \Sigma :\cdot \Gamma'$, hypothesis of lemma.

We conclude with (Metavar) applied to (1-b), (2) and (3) that $\Sigma :\cdot \Gamma' \vdash X : N$.

- (Abs). The last derivation has the form:

$$\frac{\Sigma :\cdot \Gamma.M_1 \vdash M_2 : N' \quad \Sigma :\cdot \Gamma \vdash \Pi_{M_1}.N' : s}{\Sigma :\cdot \Gamma \vdash \lambda_{M_1}.M_2 : \Pi_{M_1}.N'}\,(\text{Abs})$$

where $M = \lambda_{M_1}.M_2$ and $N = \Pi_{M_1}.N'$. We have the hypotheses:

1. (a) $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : N'$ and (b) $\Sigma :\cdot \Gamma \vdash \Pi_{M_1}.N' : s$, premises of (Abs).
2. $\Sigma :\cdot \Gamma \vdash M_1 : s'$ is a sub-derivation by Lemma 6.2(2) applied to (1-a).
3. $\Sigma :\cdot \Gamma' \vdash M_1 : s'$, by induction hypothesis applied to (2).
4. $\vdash \Sigma :\cdot \Gamma'.M_1$, by (Var-Decl) applied to (3).
5. $\Sigma :\cdot \Gamma' \vdash \Pi_{M_1}.N' : s$, by induction hypothesis applied to (1-b).
6. $\Sigma :\cdot \Gamma'.M_1 \vdash M_2 : N'$, by the induction hypothesis applied to (1-a), (4) and the hypothesis $\Gamma.M_1 =_{\lambda_{\mathcal{LP}}} \Gamma'.M_1$.

We conclude with (Abs) applied to (2) and (4) that $\Sigma :\cdot \Gamma' \vdash \lambda_{M_1}.M_2 : \Pi_{M_1}.N'$.

- (Id). The last derivation has the form:

$$\frac{\vdash \Sigma :\cdot \Gamma}{\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma}\,(\text{Id})$$

where $S = \uparrow^0$ and $\Delta = \Gamma$. We have the hypotheses:

1. $\vdash \Sigma :\cdot \Gamma$, premise of (Id).
2. $\Sigma :\cdot \Gamma' \vdash \uparrow^0 \triangleright \Gamma'$, by (Id) applied to hypothesis $\vdash \Sigma :\cdot \Gamma'$.
3. $\Gamma =_{\lambda_{\mathcal{LP}}} \Gamma'$, hypothesis of lemma.

We conclude with (Conv-Subs) applied to (1), (2) and (3) that $\Sigma :\cdot \Gamma' \vdash \uparrow^0 \triangleright \Gamma$.

- (Shift). The last derivation has the form:

$$\frac{\vdash \Sigma :\cdot \Gamma_1.M_1 \quad \Sigma :\cdot \Gamma_1 \vdash \uparrow^n \triangleright \Delta}{\Sigma :\cdot \Gamma_1.M_1 \vdash \uparrow^{Suc(n)} \triangleright \Delta}\,(\text{Shift})$$

where $\Gamma = \Gamma_1.M_1$ and $S = \uparrow^{Suc(n)}$. We have the hypotheses:

1. $\Sigma :\cdot \Gamma_1 \vdash \uparrow^n \triangleright \Delta$, premise of (Shift).
2. (a) $\Gamma' = \Gamma_2.M_2$, where $\Gamma_1 =_{\lambda_{\mathcal{LP}}} \Gamma_2$ and $M_1 =_{\lambda_{\mathcal{LP}}} M_2$, and (b) $\vdash \Sigma :\cdot \Gamma_2.M_2$, hypothesis of lemma.
3. $\vdash \Sigma :\cdot \Gamma_2$, by (Undeclare-Var) applied to (2-b).
4. $\Sigma :\cdot \Gamma_2 \vdash \uparrow^n \triangleright \Delta$, by induction hypothesis applied to (1), (2-a) and (3).

We conclude with (Shift) applied to (2-b) and (4) that $\Sigma :\cdot \Gamma_2.A_2 \vdash \uparrow^{Suc(n)} \triangleright \Delta$.

□

Lemma 6.11 *If* $\Sigma :\cdot \Gamma \vdash M : N$ *and* $\Sigma :\cdot \Gamma \vdash N : s$, *then* $\Sigma :\cdot \Gamma \vdash \uparrow^0 \cdot M_{:N} \triangleright \Gamma.N$.

Proof. We have the hypotheses:

1. $\vdash \Sigma :\cdot \Gamma$, by Lemma 6.2(1) applied to hypothesis.
2. $\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma$, by (Id) applied to (1).
3. $\Sigma :\cdot \Gamma \vdash [\uparrow^0]N : s$, by Lemma 6.3 applied to (2) and hypothesis $\Sigma :\cdot \Gamma \vdash N : s$.
4. $\Sigma :\cdot \Gamma \vdash M : [\uparrow^0]N$, by (Conv) applied to (3), $\Sigma :\cdot \Gamma \vdash M : N$ and $[\uparrow^0]N =_{\lambda_{\mathcal{LP}}} N$.

Therefore, we have the derivation

$$\frac{\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma \quad \Sigma :\cdot \Gamma \vdash N : s \quad \Sigma :\cdot \Gamma \vdash M : [\uparrow^0]N}{\Sigma :\cdot \Gamma \vdash \uparrow^0 \cdot M_{:N} \triangleright \Gamma.N}\ (\text{Cons})$$

□

Theorem 6.12 (Sort Soundness)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, *then either* $N = \mathit{Kind}$, *or* $\Sigma :\cdot \Gamma \vdash N : s$, *and*
2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* $\vdash \Sigma :\cdot \Delta$.

Proof. By induction on typing derivations. If the last rule is one of (Conv), (Abs), (Id) or (Conv-Subs), then the result is obtained from premises. The other cases are:

- (Type) and (Clos-Kind). In these cases $N = \mathit{Kind}$.
- (Prod). In this case the conclusion has the form $\Sigma :\cdot \Gamma \vdash \Pi_{M_1}.M_2 : s$. If $s = \mathit{Kind}$ then the conclusion is trivial. Otherwise $s = \mathit{Type}$, and by Lemma 6.2(1), we have $\vdash \Sigma :\cdot \Gamma$. Finally, we apply (Type) to conclude $\Sigma :\cdot \Gamma \vdash \mathit{Type} : \mathit{Kind}$.
- (Var). In this case the conclusion has the form $\Sigma :\cdot \Gamma.N' \vdash \mathbf{1} : [\uparrow^{Suc(0)}]N'$. We have the hypotheses:

  1. $\vdash \Sigma :\cdot \Gamma.N'$, premise of (Var).
  2. (a) $\vdash \Sigma :\cdot \Gamma$ and (b) $\Sigma :\cdot \Gamma \vdash N' : s'$, by Lemma 6.2 applied to (1).
  3. $\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma$, by (Id) applied to (2-a).
  4. $\Sigma :\cdot \Gamma.N' \vdash \uparrow^{Suc(0)} \triangleright \Gamma$, by (Shift) applied to (1) and (3).

  We conclude with Lemma 6.3 applied to (2-b) and (4) that $\Sigma :\cdot \Gamma.N' \vdash [\uparrow^{Suc(0)}]N' : s'$.
- (Metavar). The last derivation has the form:

$$\frac{\vdash \Sigma :\cdot \Gamma \quad X:_{\Delta} N \in \Sigma \quad \Delta =_{\lambda_{\mathcal{LP}}} \Gamma}{\Sigma :\cdot \Gamma \vdash X : N}\ (\text{Metavar})$$

  where $M = X$ (X is a meta-variable). We have the hypotheses:

1. (a) $\vdash \Sigma :\cdot \Gamma$, (b) $X:_\Delta N \in \Sigma$ and (c) $\Delta =_{\lambda_{\mathcal{L}\Pi}} \Gamma$, premises of (Metavar).
2. $\Sigma :\cdot \Delta \vdash N : s$, by Lemma 6.8 applied to (1-a) and (1-b).

By Lemma 6.8 applied to (1-a) and (1-b), we have $N = \mathit{Kind}$ or $\Sigma :\cdot \Delta \vdash N : s$. In the latter case we conclude with Lemma 6.10 applied to (1-a), (1-c) and (2) that $\Sigma :\cdot \Gamma \vdash N : s$.

- (Appl). The last derivation has the form:

$$\frac{\Sigma :\cdot \Gamma \vdash M_1 : \Pi_{N_1}.N_2 \quad \Sigma :\cdot \Gamma \vdash M_2 : N_1}{\Sigma :\cdot \Gamma \vdash (M_1\ M_2) : [\uparrow^0 \cdot M_{2:N_1}]N_2}\ (\text{Appl})$$

We have the hypotheses:

1. (a) $\Sigma :\cdot \Gamma \vdash M_1 : \Pi_{N_1}.N_2$ and (b) $\Sigma :\cdot \Gamma \vdash M_2 : N_1$, premises of (Appl).
2. $\Sigma :\cdot \Gamma \vdash \Pi_{N_1}.N_2 : s_1$, by induction hypothesis applied to (1-a) (remark that $\Pi_{N_1}.N_2 \neq \mathit{Kind}$).
3. (a) $\Sigma :\cdot \Gamma.N_1 \vdash N_2 : s_2$ and (b) $s_2 =_{\lambda_{\mathcal{L}\Pi}} s_1$, by Lemma 6.4 applied to (2).
4. $\Sigma :\cdot \Gamma \vdash N_1 : s'$, by Lemma 6.2(2) applied to (3-a).
5. $\Sigma :\cdot \Gamma \vdash \uparrow^0 \cdot M_{2:N_1} \triangleright \Gamma.N_1$, by Lemma 6.11 applied to (1-b) and (4).

We conclude with Lemma 6.3 applied to (3-a) and (5) that $\Sigma :\cdot \Gamma \vdash [\uparrow^0 \cdot M_{2:N_1}]N_2 : s_2$.

- (Clos). The last derivation has the form:

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Delta \vdash M' : N' \quad \Sigma :\cdot \Delta \vdash N' : s'}{\Sigma :\cdot \Gamma \vdash [S]M' : [S]N'}\ (\text{Clos})$$

We conclude with Lemma 6.3 that $\Sigma :\cdot \Gamma \vdash [S]N' : s'$.

- (Shift) or (Comp). We conclude with the induction hypothesis.

- (Cons). The last derivation has the form

$$\frac{\Sigma :\cdot \Gamma \vdash S \triangleright \Delta \quad \Sigma :\cdot \Gamma \vdash M : [S]N \quad \Sigma :\cdot \Delta \vdash N : s}{\Sigma :\cdot \Gamma \vdash S \cdot M_{:N} \triangleright \Delta.N}\ (\text{Cons})$$

we use (Var-Decl) on hypothesis $\Sigma :\cdot \Delta \vdash N : s$, to obtain $\vdash \Sigma :\cdot \Delta.N$.

□

6.3 Type Uniqueness

Type Uniqueness property says that types are unique modulo $\lambda_{\mathcal{L}\Pi}$-conversions.

We need the following lemmas.

Lemma 6.13 *If $M \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} s$, where s in $\{\mathit{Kind}, \mathit{Type}\}$, then s is a sub-term of M.*

Proof. By induction on the length of the reduction. The base case is trivial. At the induction step we use the induction hypothesis and the fact that for any rule $l \longrightarrow r$ in $\lambda_{\mathcal{L}\Pi}$, if s is a sub-term of r then s is a sub-term of l. □

Lemma 6.14 (Kind Soundness) *If* $\Sigma :\cdot \Gamma \vdash M : N$ *and* $N =_{\lambda_{\mathcal{LP}}} \mathit{Kind}$, *then* $N = \mathit{Kind}$.

Proof. By sort soundness theorem (Theorem 6.12), $N = \mathit{Kind}$ or $\Sigma :\cdot \Gamma \vdash N : s$. We show that the second case is not possible. *Kind* is a $\lambda_{\mathcal{LP}}$-normal form, then by Geuvers' lemma (Theorem 5.14), there exists s' such that $N \xrightarrow{\lambda_{\mathcal{LP}}^*} s'$ and $|s'| = |\mathit{Kind}|$. By definition of $|.|$, $s' = \mathit{Kind}$. By Lemma 6.13, *Kind* is a sub-term of N. By Remark 3.15, the judgment $\Sigma :\cdot \Gamma \vdash N : s$ is not valid. Therefore, the only possible case is $N = \mathit{Kind}$. □

The proof of the type uniqueness property proceeds by structural induction on expressions. In typed λ-calculus (without the η-rule) the confluence property (on pre-expressions) is used in the case of applications. In systems where the confluence property is not available a this stage of the development, the Geuvers's lemma provides a decomposing property of constructors —for instance, if $\Pi_{M_1}.N_1 =_{\lambda_{\mathcal{LP}}} \Pi_{M_2}.N_2$ then $M_1 =_{\lambda_{\mathcal{LP}}} M_2$ and $N_1 =_{\lambda_{\mathcal{LP}}} N_2$— which suffices to continue the proof.

Theorem 6.15 (Type Uniqueness) *Let* Γ_1 *and* Γ_2 *be such that* $\Gamma_1 =_{\lambda_{\mathcal{LP}}} \Gamma_2$,

1. *if* $\Sigma :\cdot \Gamma_1 \vdash M : N_1$ *and* $\Sigma :\cdot \Gamma_2 \vdash M : N_2$, *then* $N_1 =_{\lambda_{\mathcal{LP}}} N_2$, *and*
2. *if* $\Sigma :\cdot \Gamma_1 \vdash S \triangleright \Delta_1$ *and* $\Sigma :\cdot \Gamma_2 \vdash S \triangleright \Delta_2$, *then* $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_2$.

Proof. By simultaneous structural induction on M and S.

- $M = \mathit{Kind}$. By Remark 3.15 this case does not apply.
- $M = \mathbf{1}$. By Lemma 6.4, we have $\Gamma_1 = \Gamma_1'.N_1'$, $N_1 =_{\lambda_{\mathcal{LP}}} [\uparrow^{Suc(0)}]N_1'$, $\Gamma_2 = \Gamma_2'.N_2'$ and $N_2 =_{\lambda_{\mathcal{LP}}} [\uparrow^{Suc(0)}]N_2'$. Since $\Gamma_1 =_{\lambda_{\mathcal{LP}}} \Gamma_2$, we have $N_1' =_{\lambda_{\mathcal{LP}}} N_2'$. Therefore, $[\uparrow^{Suc(0)}]N_1' =_{\lambda_{\mathcal{LP}}} [\uparrow^{Suc(0)}]N_2'$ and so, $N_1 =_{\lambda_{\mathcal{LP}}} N_2$.
- $M = X$ (X is a meta-variable). In this case we use Lemma 6.6.
- $M = \mathit{Type}$. By Lemma 6.4, we have $N_1 =_{\lambda_{\mathcal{LP}}} \mathit{Kind}$ and $N_2 =_{\lambda_{\mathcal{LP}}} \mathit{Kind}$. Therefore, $N_1 =_{\lambda_{\mathcal{LP}}} N_2$.
- $M = \Pi_{M_1}.M_2$. By Lemma 6.4, we have $\Sigma :\cdot \Gamma_1.M_1 \vdash M_2 : s_1$, $N_1 =_{\lambda_{\mathcal{LP}}} s_1$, $\Sigma :\cdot \Gamma_2.M_1 \vdash M_2 : s_2$ and $N_2 =_{\lambda_{\mathcal{LP}}} s_2$. Since $\Gamma_1 =_{\lambda_{\mathcal{LP}}} \Gamma_2$, we have $\Gamma_1.M_1 =_{\lambda_{\mathcal{LP}}} \Gamma_2.M_1$. By induction hypothesis, we have $s_1 =_{\lambda_{\mathcal{LP}}} s_2$. Therefore, $N_1 =_{\lambda_{\mathcal{LP}}} N_2$.
- $M = \lambda_{M_1}.M_2$. By Lemma 6.4, we have $\Sigma :\cdot \Gamma_1.M_1 \vdash M_2 : N_1'$, $N_1 =_{\lambda_{\mathcal{LP}}} \Pi_{M_1}.N_1'$, $\Sigma :\cdot \Gamma_2.M_1 \vdash M_2 : N_2'$ and $N_2 =_{\lambda_{\mathcal{LP}}} \Pi_{M_1}.N_2'$. Since $\Gamma_1 =_{\lambda_{\mathcal{LP}}} \Gamma_2$, we have $\Gamma_1.M_1 =_{\lambda_{\mathcal{LP}}} \Gamma_2.M_1$. By induction hypothesis, we have $N_1' =_{\lambda_{\mathcal{LP}}} N_2'$. Therefore, $\Pi_{M_1}.N_1' =_{\lambda_{\mathcal{LP}}} \Pi_{M_1}.N_2'$ and so, $N_1 =_{\lambda_{\mathcal{LP}}} N_2$.
- $M = (M_1\ M_2)$. By Lemma 6.4, we have $\Sigma :\cdot \Gamma_1 \vdash M_1 : \Pi_{N_1'}.N_2'$, $N_1 =_{\lambda_{\mathcal{LP}}} [\uparrow^0 \cdot M_{2:N_1'}]N_2'$, $\Sigma :\cdot \Gamma_2 \vdash M_1 : \Pi_{N_1''}.N_2''$ and $N_2 =_{\lambda_{\mathcal{LP}}} [\uparrow^0 \cdot M_{2:N_1''}]N_2''$. By induction hypothesis, $\Pi_{N_1'}.N_2' =_{\lambda_{\mathcal{LP}}} \Pi_{N_1''}.N_2''$. By Geuvers' lemma (Theorem 5.14), $N_1' =_{\lambda_{\mathcal{LP}}} N_1''$ and $N_2' =_{\lambda_{\mathcal{LP}}} N_2''$. Therefore, $[\uparrow^0 \cdot M_{2:N_1'}]N_2' =_{\lambda_{\mathcal{LP}}} [\uparrow^0 \cdot M_{2:N_1''}]N_2''$ and so, $N_1 =_{\lambda_{\mathcal{LP}}} N_2$.
- $M = [S']M'$. By Lemma 6.4, we have $\Sigma :\cdot \Gamma_1 \vdash S' \triangleright \Delta_1$, $\Sigma :\cdot \Delta_1 \vdash M' : N_1'$, ($N_1' = \mathit{Kind}$, $N_1 =_{\lambda_{\mathcal{LP}}} \mathit{Kind}$ or $\Sigma :\cdot \Delta_1 \vdash N_1' : s_1$, $N_1 =_{\lambda_{\mathcal{LP}}} [S']N_1'$), $\Sigma :\cdot \Gamma_2 \vdash S' \triangleright \Delta_2$, $\Sigma :\cdot \Delta_2 \vdash M' : N_2'$ and ($N_2' = \mathit{Kind}$, $N_2 =_{\lambda_{\mathcal{LP}}} \mathit{Kind}$ or $\Sigma :\cdot \Delta_2 \vdash N_2' : s_2$, $N_2 =_{\lambda_{\mathcal{LP}}} [S']N_2'$). By induction hypothesis on S', $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_2$. By induction hypothesis on M', $N_1' =_{\lambda_{\mathcal{LP}}} N_2'$. Now we proceed by case analysis on N_1':

  - $N_1' = \mathit{Kind}$. By Lemma 6.14, $N_2' = \mathit{Kind}$. Therefore, $N_1 =_{\lambda_{\mathcal{LP}}} \mathit{Kind} =_{\lambda_{\mathcal{LP}}} N_2$.
  - $N_1' \neq \mathit{Kind}$. By Lemma 6.14, $N_2' \neq \mathit{Kind}$. Therefore, $N_1 =_{\lambda_{\mathcal{LP}}} [S']N_1' =_{\lambda_{\mathcal{LP}}} [S']N_2' =_{\lambda_{\mathcal{LP}}} N_2$.

- $S = \uparrow^n$. In this case we prove by structural induction on n that if $\Sigma :\cdot \Gamma_1 \vdash \uparrow^n \triangleright \Delta_1$, $\Sigma :\cdot \Gamma_2 \vdash \uparrow^n \triangleright \Delta_2$ and $\Gamma_1 =_{\lambda_{\mathcal{LP}}} \Gamma_2$, then $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_2$.

- $S = S' \cdot M'_{:N}$. By Lemma 6.4, we have $\Sigma :\cdot \Gamma_1 \vdash S' \triangleright \Delta_1'$, $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_1'.N$, $\Sigma :\cdot \Gamma_2 \vdash S' \triangleright \Delta_2'$ and $\Delta_2 =_{\lambda_{\mathcal{LP}}} \Delta_2'.N$. By induction hypothesis, $\Delta_1' =_{\lambda_{\mathcal{LP}}} \Delta_2'$. Therefore, $\Delta_1'.N =_{\lambda_{\mathcal{LP}}} \Delta_2'.N$ and so, $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_2$.

- $S = S' \circ S''$. By Lemma 6.4, we have $\Sigma :\cdot \Gamma_1 \vdash S' \triangleright \Delta_1'$, $\Sigma :\cdot \Delta_1' \vdash S'' \triangleright \Delta_1''$, $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_1''$, $\Sigma :\cdot \Gamma_2 \vdash S' \triangleright \Delta_2'$, $\Sigma :\cdot \Delta_2' \vdash S'' \triangleright \Delta_2''$ and $\Delta_2 =_{\lambda_{\mathcal{LP}}} \Delta_2''$. By induction hypothesis on S' and S'', $\Delta_1' =_{\lambda_{\mathcal{LP}}} \Delta_2'$ and $\Delta_1'' =_{\lambda_{\mathcal{LP}}} \Delta_2''$. Therefore, $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_1'' =_{\lambda_{\mathcal{LP}}} \Delta_2'' =_{\lambda_{\mathcal{LP}}} \Delta_2$.

□

6.4 Subject Reduction

Subject Reduction property says that typing is invariant under $\lambda_{\mathcal{LP}}$-reductions.

We need the following lemmas.

Lemma 6.16 *If* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$ *and* $\Sigma :\cdot \Delta \vdash N : s$, *then* $\Sigma :\cdot \Gamma.[S]N \vdash (\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:N} \triangleright \Delta.N$.

Proof. We have the hypotheses:

1. $\Sigma :\cdot \Gamma \vdash [S]N : s$, by Lemma 6.3 applied to hypotheses.
2. $\vdash \Sigma :\cdot \Gamma.[S]N$, by (Var-Decl) applied to (1).
3. $\vdash \Sigma :\cdot \Gamma$, by Lemma 6.2(1) applied to hypothesis.
4. $\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma$, by (Id) applied to (3).
5. $\Sigma :\cdot \Gamma.[S]N \vdash \uparrow^{Suc(0)} \triangleright \Gamma$, by (Shift) applied to (2) and (4).
6. $\Sigma :\cdot \Gamma.[S]N \vdash \mathbf{1} : [\uparrow^{Suc(0)}][S]N$, by (Var) applied to (2).
7. $\Sigma :\cdot \Gamma.[S]N \vdash \uparrow^{Suc(0)} \circ S \triangleright \Delta$, by (Comp) applied to (5) and hypothesis $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$.
8. $\Sigma :\cdot \Gamma.[S]N \vdash [\uparrow^{Suc(0)} \circ S]N : s$, by Lemma 6.3 applied to (7) and hypothesis $\Sigma :\cdot \Delta \vdash N : s$.
9. $\Sigma :\cdot \Gamma.[S]N \vdash \mathbf{1} : [\uparrow^{Suc(0)} \circ S]N$, by (Conv) applied to (6), (8) and $[\uparrow^{Suc(0)} \circ S]N =_{\lambda_{\mathcal{LP}}} [\uparrow^{Suc(0)}][S]N$.

Therefore, we have the derivation

$$\frac{\Sigma :\cdot \Gamma.[S]N \vdash \uparrow^{Suc(0)} \circ S \triangleright \Delta \quad \Sigma :\cdot \Gamma.[S]N \vdash \mathbf{1} : [\uparrow^{Suc(0)} \circ S]N \quad \Sigma :\cdot \Delta \vdash N : s}{\Sigma :\cdot \Gamma.[S]N \vdash (\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:N} \triangleright \Delta.N}\,(\text{Cons})$$

□

Lemma 6.17 *If* $\Sigma :\cdot \Gamma \vdash \uparrow^m \triangleright \Delta.M$, *then* $\Sigma :\cdot \Gamma \vdash \uparrow^{Suc(m)} \triangleright \Delta$.

Proof. We have the hypotheses:

1. $\vdash \Sigma :\cdot \Gamma$, by Lemma 6.2(1) applied to hypothesis.
2. $\vdash \Sigma :\cdot \Delta.M$, by sort soundness theorem (Theorem 6.12) applied to hypothesis.
3. $\vdash \Sigma :\cdot \Delta$, by (Undeclare-Var) applied to (2).
4. $\Sigma :\cdot \Delta \vdash \uparrow^0 \triangleright \Delta$, by (Id) applied to (3).
5. $\Sigma :\cdot \Delta.M \vdash \uparrow^{Suc(0)} \triangleright \Delta$, by (Shift) applied to (2) and (4).

Now, we prove by structural induction on m that $\Sigma :\cdot \Gamma \vdash \uparrow^{Suc(m)} \triangleright \Delta$. □

Lemma 6.18 *If* $[\uparrow^n]M =_{\lambda_{\mathcal{L}\Pi}} [\uparrow^n]M'$, *then* $M =_{\lambda_{\mathcal{L}\Pi}} M'$.

Proof. By structural induction on n. Note that if $[\uparrow^{Suc(0)}]M =_{\lambda_{\mathcal{L}\Pi}} [\uparrow^{Suc(0)}]M'$, then

$$[\uparrow^0 \cdot \mathbf{1}_{:M}][\uparrow^{Suc(0)}]M =_{\lambda_{\mathcal{L}\Pi}} [\uparrow^0 \cdot \mathbf{1}_{:M}][\uparrow^{Suc(0)}]M'$$

But also,

$$M \xleftarrow{\lambda_{\mathcal{L}\Pi}^*} [\uparrow^0 \cdot \mathbf{1}_{:M}][\uparrow^{Suc(0)}]M =_{\lambda_{\mathcal{L}\Pi}} [\uparrow^0 \cdot \mathbf{1}_{:M}][\uparrow^{Suc(0)}]M' \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} M'$$

□

Lemma 6.19 *The* $\lambda_{\mathcal{L}\Pi}$*-rewrite system preserves typing.*

Proof. We verify the property for each rewrite rule. We detail some cases.

- (Beta) Let $\Sigma :\cdot \Gamma \vdash (\lambda_{M_1}.M_2\ N_1) : N_2$, we show $\Sigma :\cdot \Gamma \vdash [\uparrow^0 \cdot N_{1:M_1}]M_2 : N_2$. We have the hypotheses:

  1. (a) $\Sigma :\cdot \Gamma \vdash \lambda_{M_1}.M_2 : \Pi_{M_1'}.M_2'$, (b) $\Sigma :\cdot \Gamma \vdash N_1 : M_1'$ and (c) $N_2 =_{\lambda_{\mathcal{L}\Pi}} [\uparrow^0 \cdot N_{1:M_1'}]M_2'$, by Lemma 6.4 applied to hypothesis $\Sigma :\cdot \Gamma \vdash (\lambda_{M_1}.M_2\ N_1) : N_2$.
  2. (a) $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : M_2''$, (b) $\Sigma :\cdot \Gamma \vdash \Pi_{M_1}.M_2'' : s$ and (c) $\Pi_{M_1'}.M_2' =_{\lambda_{\mathcal{L}\Pi}} \Pi_{M_1}.M_2''$, by Lemma 6.4 applied to (1-a).
  3. (a) $\Sigma :\cdot \Gamma \vdash M_1 : s_1$, by the Lemma 6.2(2) applied to (2-a), and (b) $\Sigma :\cdot \Gamma.M_1 \vdash M_2'' : s_2$, by Lemma 6.4 applied to (2-b).
  4. (a) $M_1' =_{\lambda_{\mathcal{L}\Pi}} M_1$ and (b) $M_2' =_{\lambda_{\mathcal{L}\Pi}} M_2''$, by Geuvers' lemma (Theorem 5.14) applied to (2-c).
  5. $\Sigma :\cdot \Gamma \vdash N_1 : M_1$, by (Conv) applied to (1-b), (3-a) and (4-a).
  6. $\Sigma :\cdot \Gamma \vdash \uparrow^0 \cdot N_{1:M_1} \triangleright \Gamma.M_1$, by Lemma 6.11 applied to (3-a) and (5).
  7. $\Sigma :\cdot \Gamma \vdash N_2 : s'$, by sort soundness theorem (Theorem 6.12) applied to hypothesis of lemma. Remark that the case $N_2 = \mathit{Kind}$ is not possible by Lemma 6.9.

Therefore, we have the derivation

$$\frac{\Sigma :\cdot \Gamma \vdash \uparrow^0 \cdot N_{1:M_1} \triangleright \Gamma.M_1 \quad \Sigma :\cdot \Gamma.M_1 \vdash M_2 : M_2'' \quad \Sigma :\cdot \Gamma.M_1 \vdash M_2'' : s_2}{\Sigma :\cdot \Gamma \vdash [\uparrow^0 \cdot N_{1:M_1}]M_2 : [\uparrow^0 \cdot N_{1:M_1}]M_2''} \text{(Clos)}$$

Finally, $[\uparrow^0 \cdot N_{1:M_1}]M_2'' =_{\lambda_{\mathcal{LP}}} [\uparrow^0 \cdot N_{1:M_1'}]M_2' =_{\lambda_{\mathcal{LP}}} N_2$, so we conclude with (Conv) and (7) that $\Sigma :\cdot \Gamma \vdash [\uparrow^0 \cdot N_{1:M_1}]M_2 : N_2$.

- (Lambda). Let $\Sigma :\cdot \Gamma \vdash [S](\lambda_{M_1}.M_2) : N$, we show $\Sigma :\cdot \Gamma \vdash \lambda_{[S]M_1}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]M_2 : N$. We have the hypotheses:

  1. (a) $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, (b) $\Sigma :\cdot \Delta \vdash \lambda_{M_1}.M_2 : N_1$, (c) $\Sigma :\cdot \Delta \vdash N_1 : s$ and (d) $N =_{\lambda_{\mathcal{LP}}} [S]N_1$, by Lemma 6.4 applied to hypothesis of lemma. Remark that the case $N_1 = \mathit{Kind}$ is not possible by Lemma 6.9.
  2. (a) $\Sigma :\cdot \Delta.M_1 \vdash M_2 : N_2$, (b) $\Sigma :\cdot \Delta \vdash \Pi_{M_1}.N_2 : s'$ and (c) $N_1 =_{\lambda_{\mathcal{LP}}} \Pi_{M_1}.N_2$, by Lemma 6.4 applied to (1-b).
  3. (a) $\Sigma :\cdot \Delta \vdash M_1 : s_1$, by Lemma 6.2(2) applied to (2-a), and (b) $\Sigma :\cdot \Delta.M_1 \vdash N_2 : s_2$, by Lemma 6.4 applied to (2-b).
  4. $\Sigma :\cdot \Gamma \vdash [S]M_1 : s_1$, by Lemma 6.3 applied to (1-a) and (3-a).
  5. $\Sigma :\cdot \Gamma.[S]M_1 \vdash (\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1} \triangleright \Delta.M_1$, by Lemma 6.16 applied to (1-a) and (3-a).
  6. (a) $\Sigma :\cdot \Gamma.[S]M_1 \vdash [(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]M_2 : [(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]N_2$, by (Clos) applied to (2-a), (3-b) and (5); and (b) $\Sigma :\cdot \Gamma.[S]M_1 \vdash [(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]N_2 : s_2$ by Lemma 6.3 applied to (3-b) and (5).
  7. $\Sigma :\cdot \Gamma \vdash N : s''$, by sort soundness theorem (Theorem 6.12) applied to hypothesis of lemma. Remark that the case $N = \mathit{Kind}$ is not possible by Lemma 6.9.
  8. $\Sigma :\cdot \Gamma \vdash \Pi_{[S]M_1}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]N_2 : s_2$.

  Therefore, we have the derivation

$$\frac{\Sigma :\cdot \Gamma.[S]M_1 \vdash [(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]M_2 : [(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]N_2 \quad (8)}{\Sigma :\cdot \Gamma \vdash \lambda_{[S]M_1}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]M_2 : \Pi_{[S]M_1}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]N_2} \text{(Abs)}$$

  Finally, $\Pi_{[S]M_1}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]N_2 =_{\lambda_{\mathcal{LP}}} [S](\Pi_{M_1}.N_2) =_{\lambda_{\mathcal{LP}}} [S]N_1 =_{\lambda_{\mathcal{LP}}} N$, so we conclude with (Conv) and (7) that $\Sigma :\cdot \Gamma \vdash \lambda_{[S]M_1}.[(\uparrow^{Suc(0)} \circ S) \cdot \mathbf{1}_{:M_1}]M_2 : N$.

- (Clos). Let $\Sigma :\cdot \Gamma \vdash [T][S]M : N$, we show $\Sigma :\cdot \Gamma \vdash [T \circ S]M : N$. We have the hypotheses:

  1. (a) $\Sigma :\cdot \Gamma \vdash T \triangleright \Delta_1$, (b) $\Sigma :\cdot \Delta_1 \vdash [S]M : N_1$, (c) ($N_1 = \mathit{Kind}$, $N =_{\lambda_{\mathcal{LP}}} \mathit{Kind}$ or $\Sigma :\cdot \Delta_1 \vdash N_1 : s_1$, $N =_{\lambda_{\mathcal{LP}}} [T]N_1$), by Lemma 6.4 applied to hypothesis of lemma.
  2. (a) $\Sigma :\cdot \Delta_1 \vdash S \triangleright \Delta_2$, (b) $\Sigma :\cdot \Delta_2 \vdash M : N_2$, (c) ($N_2 = \mathit{Kind}$, $N_1 =_{\lambda_{\mathcal{LP}}} \mathit{Kind}$ or $\Sigma :\cdot \Delta_2 \vdash N_2 : s_2$, $N_1 =_{\lambda_{\mathcal{LP}}} [S]N_2$), by Lemma 6.4 applied to (1-b).
  3. $\Sigma :\cdot \Gamma \vdash T \circ S \triangleright \Delta_2$, by (Comp) applied to (1-a) and (2-a).
  4. $N = \mathit{Kind}$ or $\Sigma :\cdot \Gamma \vdash N : s$, by sort soundness theorem (Theorem 6.12) applied to hypothesis.

By case analysis on N_2.

- $N_2 = \mathit{Kind}$. By Lemma 6.14 applied to (1-c) and to (2-c), $N_2 = N_1 = N = \mathit{Kind}$. Therefore, we have the derivation

$$\frac{\Sigma :\cdot \Gamma \vdash T \circ S \triangleright \Delta_2 \quad \Sigma :\cdot \Delta_2 \vdash M : \mathit{Kind}}{\Sigma :\cdot \Gamma \vdash [T \circ S]M : \mathit{Kind}}\,(\text{Clos-Kind})$$

- $N_2 \neq \mathit{Kind}$. By Lemma 6.14, we have $N_1 \neq \mathit{Kind}$ and $N \neq \mathit{Kind}$. Therefore, we have the derivation

$$\frac{\Sigma :\cdot \Gamma \vdash T \circ S \triangleright \Delta_2 \quad \Sigma :\cdot \Delta_2 \vdash M : N_2 \quad \Sigma :\cdot \Delta_2 \vdash N_2 : s_2}{\Sigma :\cdot \Gamma \vdash [T \circ S]M : [T \circ S]N_2}\,(\text{Clos})$$

  Finally, $[T \circ S]N_2 =_{\lambda_{\mathcal{LP}}} [T][S]N_2 =_{\lambda_{\mathcal{LP}}} [T]N_1 =_{\lambda_{\mathcal{LP}}} N$. We conclude with (Conv) and (4) that $\Sigma :\cdot \Gamma \vdash [T \circ S]M : N$.

- (VarCons). Let $\Sigma :\cdot \Gamma \vdash [S \cdot M_{1:M_2}]\mathbf{1} : N$, we show $\Sigma :\cdot \Gamma \vdash M_1 : N$. We have the hypotheses:

  1. (a) $\Sigma :\cdot \Gamma \vdash S \cdot M_{1:M_2} \triangleright \Delta_1$, (b) $\Sigma :\cdot \Delta_1 \vdash \mathbf{1} : N_1$, (c) $\Sigma :\cdot \Delta_1 \vdash N_1 : a$ and (d) $N =_{\lambda_{\mathcal{LP}}} [S \cdot M_{1:M_2}]N_1$, by Lemma 6.4 applied to hypothesis of lemma. Remark that the case $N_1 = \mathit{Kind}$ is not possible by Lemma 6.9.
  2. (a) $\Delta_1 = \Delta_2.N_2$ and (b) $N_1 =_{\lambda_{\mathcal{LP}}} [\uparrow^{Suc(0)}]N_2$, by Lemma 6.4 applied to (1-b).
  3. (a) $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta_2'$, (b) $\Sigma :\cdot \Gamma \vdash M_1 : [S]M_2$, (c) $\Sigma :\cdot \Delta_2' \vdash M_2 : s'$ and (d) $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_2'.M_2$, by Lemma 6.4 applied to (1-a).
  4. $M_2 =_{\lambda_{\mathcal{LP}}} N_2$, by (2-a) and (3-d).
  5. $\Sigma :\cdot \Gamma \vdash N : s''$, by sort soundness theorem (Theorem 6.12) applied to hypothesis of lemma. Remark that the case $N = \mathit{Kind}$ is not possible by Lemma 6.9.

  Finally, $[S]M_2 =_{\lambda_{\mathcal{LP}}} [S \cdot M_{1:M_2}][\uparrow^{Suc(0)}]M_2 =_{\lambda_{\mathcal{LP}}} [S \cdot M_{1:M_2}][\uparrow^{Suc(0)}]N_2 =_{\lambda_{\mathcal{LP}}} [S \cdot M_{1:M_2}]N_1 =_{\lambda_{\mathcal{LP}}} N$, so we conclude with (Conv) applied to (3-b) and (5) that $\Sigma :\cdot \Gamma \vdash M_1 : N$.

- (Map). Let $\Sigma :\cdot \Gamma \vdash T \circ (S \cdot M_{:N}) \triangleright \Delta$, we show $\Sigma :\cdot \Gamma \vdash (T \circ S) \cdot [T]M_{:N} \triangleright \Delta$. We have the hypotheses:

  1. (a) $\Sigma :\cdot \Gamma \vdash T \triangleright \Delta_1$, (b) $\Sigma :\cdot \Delta_1 \vdash S \cdot M_{:N} \triangleright \Delta'$ and (c) $\Delta =_{\lambda_{\mathcal{LP}}} \Delta'$, by Lemma 6.4 applied to hypothesis of lemma.
  2. (a) $\Sigma :\cdot \Delta_1 \vdash S \triangleright \Delta_2$, (b) $\Sigma :\cdot \Delta_1 \vdash M : [S]N$, (c) $\Sigma :\cdot \Delta_2 \vdash N : s$ and (d) $\Delta' =_{\lambda_{\mathcal{LP}}} \Delta_2.N$, by Lemma 6.4 applied to (1-b).
  3. $\Sigma :\cdot \Delta_1 \vdash [S]N : s$, by Lemma 6.3 applied to (2-a) and (2-c).
  4. $\Sigma :\cdot \Gamma \vdash T \circ S \triangleright \Delta_2$, by (Comp) applied to (1-a) and (2-a).
  5. $\Sigma :\cdot \Gamma \vdash [T]M : [T][S]N$, by (Clos) applied to (1-a), (2-b) and (3).
  6. $\Sigma :\cdot \Gamma \vdash [T][S]N : s$, by Lemma 6.3 applied to (1-a) and (3).
  7. $\Sigma :\cdot \Gamma \vdash [T \circ S]N : s$, using the above (Clos)-case with (5).
  8. $\Sigma :\cdot \Gamma \vdash [T]M : [T \circ S]N$, by (Conv) applied to (4), (6) and $[T][S]N =_{\lambda_{\mathcal{LP}}} [T \circ S]N$.
  9. $\vdash \Sigma :\cdot \Delta$, by sort soundness theorem (Theorem 6.12) applied to hypothesis of lemma.

Therefore, we have the derivation

$$\frac{\Sigma :\cdot \Gamma \vdash T \circ S \triangleright \Delta_2 \quad \Sigma :\cdot \Gamma \vdash [T]M : [T \circ S]N \quad \Sigma :\cdot \Delta_2 \vdash N : s}{\Sigma :\cdot \Gamma \vdash (T \circ S) \cdot [T]M_{:N} \triangleright \Delta_2.N} \text{(Cons)}$$

Finally, $\Delta_2.N =_{\lambda_{\mathcal{LP}}} \Delta' =_{\lambda_{\mathcal{LP}}} \Delta$, so we conclude with (Conv-Subs) and (8) that $\Sigma :\cdot \Gamma \vdash (T \circ S) \cdot [T]M_{:N} \triangleright \Delta$.

- (Shift0). Let $\Sigma :\cdot \Gamma \vdash \uparrow^{Suc(0)} \cdot \mathbf{1}_{:M} \triangleright \Delta$, we show $\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Delta$. We have the hypotheses:

  1. (a) $\Sigma :\cdot \Gamma \vdash \uparrow^{Suc(0)} \triangleright \Delta'$, (b) $\Sigma :\cdot \Gamma \vdash \mathbf{1} : [\uparrow^{Suc(0)}]M$, (c) $\Sigma :\cdot \Delta' \vdash M : s$ and (d) $\Delta =_{\lambda_{\mathcal{LP}}} \Delta'.M$, by Lemma 6.4 applied to hypothesis of lemma.
  2. (a) $\Gamma = \Gamma'.M'$, (b) $\Sigma :\cdot \Gamma' \vdash \uparrow^0 \triangleright \Delta''$ and (c) $\Delta'' =_{\lambda_{\mathcal{LP}}} \Delta'$, by Lemma 6.4 applied to (1-a).
  3. $\Gamma' =_{\lambda_{\mathcal{LP}}} \Delta''$, by Lemma 6.4 applied to (2-b).
  4. $\vdash \Sigma :\cdot \Gamma$, by Lemma 6.2(1) applied (1-a).
  5. $\Sigma :\cdot \Gamma'.M' \vdash \mathbf{1} : [\uparrow^{Suc(0)}]M'$, by (Var) applied to (4) and (2-a).
  6. $[\uparrow^{Suc(0)}]M' =_{\lambda_{\mathcal{LP}}} [\uparrow^{Suc(0)}]M$, by type uniqueness theorem (Theorem 6.15) applied to (1-b), (5) and (2-a).
  7. (a) $M =_{\lambda_{\mathcal{LP}}} M'$, by Lemma 6.18 applied to (6), and (b) $\Gamma' =_{\lambda_{\mathcal{LP}}} \Delta'$, by (3) and (2-c).
  8. $\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma$, by (Id) applied to (4).
  9. $\vdash \Sigma :\cdot \Delta$, by sort soundness (Theorem 6.12) applied to hypothesis of lemma.

  Finally, $\Gamma = \Gamma'.M' =_{\lambda_{\mathcal{LP}}} \Delta'.M =_{\lambda_{\mathcal{LP}}} \Delta$, so we conclude with (Conv-Subs) applied to (8) and (9) that $\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Delta$.

- (ShiftS). Let $\Sigma :\cdot \Gamma \vdash \uparrow^{Suc(n)} \cdot [\uparrow^n]\mathbf{1}_{:M} \triangleright \Delta$, we show $\Sigma :\cdot \Gamma \vdash \uparrow^n \triangleright \Delta$. We have the hypotheses:

  1. (a) $\Sigma :\cdot \Gamma \vdash \uparrow^{Suc(n)} \triangleright \Delta'$, (b) $\Sigma :\cdot \Gamma \vdash [\uparrow^n]\mathbf{1} : [\uparrow^{Suc(n)}]M$, (c) $\Sigma :\cdot \Delta' \vdash M : s$ and (d) $\Delta =_{\lambda_{\mathcal{LP}}} \Delta'.M$, by Lemma 6.4 applied to hypothesis of lemma.
  2. (a) $\Gamma = \Gamma'.M'$, (b) $\Sigma :\cdot \Gamma' \vdash \uparrow^n \triangleright \Delta''$ and (c) $\Delta'' =_{\lambda_{\mathcal{LP}}} \Delta'$, by Lemma 6.4 applied to (1-a).
  3. (a) $\Sigma :\cdot \Gamma \vdash \uparrow^n \triangleright \Gamma''$, (b) $\Sigma :\cdot \Gamma'' \vdash \mathbf{1} : N$ (c) $\Sigma :\cdot \Gamma'' \vdash N : s'$ and (d) $[\uparrow^{Suc(n)}]M =_{\lambda_{\mathcal{LP}}} [\uparrow^n]N$, by Lemma 6.4 applied to (1-b). Remark that the case $N = \mathit{Kind}$ is not possible by Lemma 6.9.
  4. (a) $\Gamma'' = \Omega.N'$ and (b) $N =_{\lambda_{\mathcal{LP}}} [\uparrow^{Suc(0)}]N'$, by Lemma 6.4 applied to (3-b).
  5. $[\uparrow^{Suc(n)}]M =_{\lambda_{\mathcal{LP}}} [\uparrow^n][\uparrow^{Suc(0)}]N' =_{\lambda_{\mathcal{LP}}} [\uparrow^{Suc(n)}]N'$, by (3-d) and (4-b).
  6. $M =_{\lambda_{\mathcal{LP}}} N'$, by Lemma 6.18 applied to (5).
  7. $\Sigma :\cdot \Gamma \vdash \uparrow^{Suc(n)} \triangleright \Omega$, by Lemma 6.17 applied to (3-a) and (4-a).
  8. $\Omega =_{\lambda_{\mathcal{LP}}} \Delta'$, by type uniqueness theorem (Theorem 6.15) applied to (1-a) and (7).
  9. $\vdash \Sigma :\cdot \Delta$, by sort soundness theorem (Theorem 6.12) applied to hypothesis of lemma.

  Finally, $\Gamma'' = \Omega.N' =_{\lambda_{\mathcal{LP}}} \Delta'.M =_{\lambda_{\mathcal{LP}}} \Delta$, so we conclude with (Conv-Subs) applied to (3-a) and (9) that $\Sigma :\cdot \Gamma \vdash \uparrow^n \triangleright \Delta$.

- The other cases are similar.

□

Theorem 6.20 (Subject Reduction) *If* $x \xrightarrow{\lambda_{\mathcal{L}\Pi}{}^*} y$, *then*

1. *if* x *is a term and* $\Sigma :\cdot \Gamma \vdash x : N$, *then* $\Sigma :\cdot \Gamma \vdash y : N$, *and*
2. *if* x *is a substitution and* $\Sigma :\cdot \Gamma \vdash x \triangleright \Delta$, *then* $\Sigma :\cdot \Gamma \vdash y \triangleright \Delta$.

Proof. We show that typing is preserved for one-step reductions (i.e. $\xrightarrow{\lambda_{\mathcal{L}\Pi}}$), and then it is also for the reflexive and transitive closure (i.e. $\xrightarrow{\lambda_{\mathcal{L}\Pi}{}^*}$). Let $x \xrightarrow{\lambda_{\mathcal{L}\Pi}} y$ be a one-step reduction, we proceed by induction on the depth of the redex reduced in x. At the initial case x is reduced at the top level. In this case we conclude with Lemma 6.19. At the induction step we resolve with Lemma 6.4 and induction hypothesis. □

6.5 Typing Soundness

Sometimes the conversion rule (Conv) is expressed as [35]

$$\frac{\Gamma \vdash M : N_1 \quad \Gamma \vdash N_2 : s \quad (N_1 \longrightarrow N_2 \text{ or } N_2 \longrightarrow N_1)}{\Gamma \vdash M : N_2} \text{(Conv')}$$

The rule (Conv) seems to be more general that the rule (Conv'). In fact, the latter one allows conversions of types only via a path through well-typed terms. Geuvers and Werner [35] define *soundness of a type system* when it holds that convertibility of terms remains in the set of well-typed terms. In sound systems the rules (Conv) and (Conv') are equivalent.

Typing Soundness property says that if two typed expressions are $\lambda_{\mathcal{L}\Pi}$-convertible, then they are convertible via a path of well-typed expressions.

We need the following lemma.

Lemma 6.21 *Let* M, M', S, S' *be* $\mathcal{L}_\Pi$*-normal forms such that* $|M| = |M'|$ *and* $|S| = |S'|$, *then*

1. *if* $\Sigma :\cdot \Gamma \vdash M : N$ *and* $\Sigma :\cdot \Gamma \vdash M' : N'$, *then there exists a path of well-typed terms to convert* M *and* M',
2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta'$, $\Sigma :\cdot \Gamma \vdash S' \triangleright \Delta''$ *and* $\Delta' =_{\lambda_{\mathcal{L}\Pi}} \Delta''$, *then there exists a path of well-typed substitutions to convert* S *and* S'.

Proof. By structural induction on M and S. We only detail two cases, the others are similar.

- $M = [T]X$ (X is a meta-variable). By definition, $M' = [T']X$ with $|T| = |T'|$. From Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash T \triangleright \Delta_1$, $\Sigma :\cdot \Delta_1 \vdash X : A_1$, $\Sigma :\cdot \Gamma \vdash T' \triangleright \Delta_2$, $\Sigma :\cdot \Delta_2 \vdash X : A_2$. By Lemma 6.6, $\Delta_1 =_{\lambda_{\mathcal{L}\Pi}} \Delta_2$. By induction hypothesis there exists a path of well-typed substitutions to convert T and T'.
- $S = T \cdot N_{1:N_2}$. By definition, $S' = T' \cdot N'_{1:N'_2}$ with $|N_1| = |N'_1|$ and $|T| = |T'|$. From Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash N_1 : [T]N_2$, $\Sigma :\cdot \Gamma \vdash T \triangleright \Delta_1$, $\Sigma :\cdot \Gamma \vdash N'_1 : [T']N'_2$, $\Sigma :\cdot \Gamma \vdash T' \triangleright \Delta_2$ and $\Delta_1 =_{\lambda_{\mathcal{L}\Pi}} \Delta_2$. By induction hypothesis there exists a path of well-typed terms trough $N_1 =_{\lambda_{\mathcal{L}\Pi}} N'_1$ and $T =_{\lambda_{\mathcal{L}\Pi}} T'$. It is not difficult to see that there exists a path of well-typed terms trough:
  $T \cdot N_{1:N_2} \xleftarrow{\mathcal{L}_\Pi{}^+} (T \cdot N_{1:N_2}) \circ (\uparrow^{Suc(0)} \cdot \mathbf{1}_{:N'_2}) \xrightarrow{\mathcal{L}_\Pi{}^+} T \cdot N_{1:N'_2} =_{\lambda_{\mathcal{L}\Pi}} T' \cdot N'_{1:N'_2}$.

□

Theorem 6.22 (Typing Soundness)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, $\Sigma :\cdot \Gamma \vdash M' : N'$ *and* $M =_{\lambda_{\mathcal{LP}}} M'$, *then there exists a path of well-typed terms to convert* M *and* M', *and*

2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, $\Sigma :\cdot \Gamma \vdash S' \triangleright \Delta$ *and* $S =_{\lambda_{\mathcal{LP}}} S'$, *then there exists a path of well-typed substitutions to convert* S *and* S'.

Proof. We only detail the first case, the second one is similar. From Lemma 5.10(1) we have $|M| = |M'|$. The confluence property of $|\lambda_{\mathcal{LP}}|$ says that there exists x in $|\lambda_{\mathcal{LP}}|$ such that $|M| \xrightarrow{|\lambda_{\mathcal{LP}}|^*} x$ and $|M'| \xrightarrow{|\lambda_{\mathcal{LP}}|^*} x$. From Lemma 5.10(2) we have that there exists M_1 and M_2 such that $M \xrightarrow{\lambda_{\mathcal{LP}}^*} M_1$, $M' \xrightarrow{\lambda_{\mathcal{LP}}^*} M_2$ and $|M_1| = |M_2| = x$. Since $\mathcal{L}_\Pi$ is terminating (Proposition 3.11), there exists M_1' and M_2' such that $M_1 \xrightarrow{\mathcal{L}_\Pi^*} M_1'$, $M_2 \xrightarrow{\mathcal{L}_\Pi^*} M_2'$ and M_1', M_2' are $\mathcal{L}_\Pi$-normal forms. By subject reduction theorem (Theorem 6.20), we have $\Sigma :\cdot \Gamma \vdash M_1' : N$ and $\Sigma :\cdot \Gamma \vdash M_2' : N'$.

Now, from Lemma 5.10(1), we have $x \xrightarrow{|\mathcal{L}_\Pi|^*} |M_1'|$ and $x \xrightarrow{|\mathcal{L}_\Pi|^*} |M_2'|$. But M_1' and M_2' are $\mathcal{L}_\Pi$-normal forms, hence, by Corollary 5.9, $|M_1'|$ and $|M_2'|$ are $|\mathcal{L}_\Pi|$-normal forms. Since $|\mathcal{L}_\Pi|$ is confluent, $|M_1'| = |M_2'|$. By Lemma 6.21 there exists a path of well-typed terms trough $M_1' =_{\lambda_{\mathcal{LP}}} M_2'$. □

A direct consequence of typing soundness and subject reduction is the following property.

Corollary 6.23 *If* $\Sigma :\cdot \Gamma \vdash M_1 : N_1$, $\Sigma :\cdot \Gamma \vdash M_2 : N_2$ *and* $M_1 =_{\lambda_{\mathcal{LP}}} M_2$, *then* $N_1 =_{\lambda_{\mathcal{LP}}} N_2$.

Chapter 7

Weak Normalization

Strong normalization does not hold for typed versions of $\lambda_{\mathcal{L}}$. In fact, Melliès shows in [63] that his counter-example for preservation of strong normalization in the $\lambda\sigma$-calculus [64], can be adapted either to systems without associativity of composition (as $\lambda_{\mathcal{L}}$), and even if we give priority to the rules (ShiftCons) and (VarCons).

In λ-calculi with explicit substitutions that implement one-step semantics of β-reduction —i.e. if M, N are ground terms and $M \xrightarrow{\beta} N$, then $M \xrightarrow{\text{(Beta)}} M'$ where N is the substitution-normal form of M'— as $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$ and $\lambda_{\mathcal{L}}$, weak normalization on typed pure terms follows directly from strong normalization of typed λ-calculus. When we consider semi-open expressions, an additional difficulty arises: the occurrence of meta-variables and substitutions in normal forms. Note that the set of normal forms of semi-open terms is not included in the set of pure terms, e.g. the term $[\uparrow^{Suc(0)}]X$ is a $\lambda_{\mathcal{L}}$-normal form, but it is not pure.

For the simply-typed version of $\lambda\sigma$ (with meta-variables), Goubault-Larreq [38] proposes a clever translation from $\lambda\sigma$-terms into a family of λ-terms. In this approach, weak normalization is deduced from strong normalization of the simply-typed λ-calculus. That proof is adapted to a second-order type system without dependent types in [37].

In this chapter we prove weak normalization for three typed versions of $\lambda_{\mathcal{L}}$: simple types, dependent types ($\lambda\Pi_{\mathcal{L}}$) and the calculus of constructions ($\mathrm{CC}_{\mathcal{L}}$). The proof we provide for the simply-typed $\lambda_{\mathcal{L}}$-calculus, can be adapted to $\lambda\sigma$ in a straightforward way. This gives an alternative proof to that developed independently by Goubault-Larreq.

The weak normalization proofs presented in this chapter are based on strong normalization of the reduction of (Beta) followed by a normal reduction of substitutions. We use the standard technique of reducibility, originally due to Tait for the simply-typed λ-calculus ans system T, and posteriorly extended by Girard to the system F (the λ-calculus of second-order), by Coquand for the Calculus of Constructions, and by Werner for the Calculus of Inductive Constructions. The general idea of this technique is to define an interpretation for each type into a set of terms satisfying certain closure properties. In particular, terms in the interpretations are defined to be strongly normalizing. The final step of the proof is to verify that if a term M is of type A, then M is in the interpretation of A, and so M is strongly normalizing.

Among the diverse proofs of termination using a reducibility notion, we follow the presentation given in [33] for the calculus of constructions, which is based on saturated sets. We adapt this proof for the typed versions of $\lambda_{\mathcal{L}}$. In order to solve technical problems due to the non-confluence of the

calculus of pre-expressions, we define saturated sets in a different way. However, the structure of the proofs is the same.

7.1 Simple Types

In Geuvers' proof, terms are interpreted by functions called *valuations*. In our proof, valuations are just particular explicit substitutions. We prove that if M is a $\mathcal{L}$-normal form and $\Gamma \vdash M : A$, then for any valuation S of M, the $\mathcal{L}$-normal form of $[S]M$, i.e. $([S]M)\downarrow_{\mathcal{L}}$, is included in the interpretation of A, denoted $[\![A]\!]$. The identity substitution is a valuation of any term, thus $([\uparrow^0]M)\downarrow_{\mathcal{L}} = M \in [\![A]\!]$.

We define $\mathcal{NF}_{\mathcal{L}}$ as the set of all the $\mathcal{L}$-normal forms of semi-open expressions.

Definition 7.1 *Let x, y be in $\mathcal{NF}_{\mathcal{L}}$, we say that x $\beta_{\mathcal{L}}$-converts to y, denoted by $x \xrightarrow{\beta_{\mathcal{L}}} y$, if $x \xrightarrow{\text{(Beta)}} w$ and $y = (w)\downarrow_{\mathcal{L}}$.*

We denote by $\mathcal{SN}_{\mathcal{L}}$ the set of $\beta_{\mathcal{L}}$-strongly normalizing expressions of $\mathcal{NF}_{\mathcal{L}}$.

Definition 7.2 *Let M be in $\mathcal{NF}_{\mathcal{L}}$, M is* neutral *if it does not have the form $\lambda A.N$. The set of neutral terms is denoted by $\mathcal{NT}$.*

7.1.1 Saturated Sets and Interpretation of Types

Definition 7.3 *A set of terms $\Lambda \subseteq \mathcal{NF}_{\mathcal{L}}$ is* saturated *(or a reducibility candidate) if*

1. $\Lambda \subseteq \mathcal{SN}_{\mathcal{L}}$.
2. *If $M \in \Lambda$ and $M \xrightarrow{\beta_{\mathcal{L}}} M'$, then $M' \in \Lambda$.*
3. *If $M \in \mathcal{NT}$, and whenever we reduce a $\beta_{\mathcal{L}}$-redex of M we obtain a term $M' \in \Lambda$, then $M \in \Lambda$.*

The set of saturated sets is denoted by **SAT**.

From Def. 7.3(3), we have

Remark 7.4 *Let M be in $\mathcal{NT}$ such that M is a $\beta_{\mathcal{L}}$-normal form. For any $\Lambda \in$ **SAT**, $M \in \Lambda$.*

Lemma 7.5 *For any $\Lambda \in$ **SAT**, substitution $S \in \mathcal{SN}_{\mathcal{L}}$, and meta-variable X, $([S]X)\downarrow_{\mathcal{L}} \in \Lambda$.*

Proof. Let $M = ([S]X)\downarrow_{\mathcal{L}}$, we reason by induction on $\nu(S)$[1]. M is neutral, then by Def. 7.3(3), it suffices to consider the reductions of M.

- If M is a $\beta_{\mathcal{L}}$-normal form, then M is neutral and so, by Remark 7.4, $M \in \Lambda$.
- $M \xrightarrow{\beta_{\mathcal{L}}} X$. By Remark 7.4, we have $X \in \Lambda$.
- $M \xrightarrow{\beta_{\mathcal{L}}} [S']X$, with $S \xrightarrow{\beta_{\mathcal{L}}} S'$. By hypothesis, we have $S' \in \mathcal{SN}_{\mathcal{L}}$ and $\nu(S') < \nu(S)$. So, by induction hypothesis, $([S']X)\downarrow_{\mathcal{L}} = [S']X \in \Lambda$.

[1] "If x is strongly normalizing, $\nu(x)$ is a number which bounds the length of every normalization sequence beginning with x" [36].

In every case, M $\beta_{\mathcal{L}}$-reduces into terms in Λ, thus by Def. 7.3(3), $([S]X)\downarrow_{\mathcal{L}} \in \Lambda$. □

Lemma 7.6 $\mathcal{SN}_{\mathcal{L}} \in \mathbf{SAT}$.

Proof. We verify easily the following conditions.

1. $\mathcal{SN}_{\mathcal{L}} \subseteq \mathcal{SN}_{\mathcal{L}}$.
2. If $M \in \mathcal{SN}_{\mathcal{L}}$ and $M \xrightarrow{\beta_{\mathcal{L}}} M'$, then $M' \in \mathcal{SN}_{\mathcal{L}}$.
3. If $M \in \mathcal{NT}$, and whenever we reduce a $\beta_{\mathcal{L}}$-redex of M we obtain a term $M' \in \mathcal{SN}_{\mathcal{L}}$, then $M \in \mathcal{SN}_{\mathcal{L}}$.

□

Definition 7.7 *Let Λ, Λ' be in* **SAT**, *we define the set*

$$\Lambda \to \Lambda' = \{M \in \mathcal{NF}_{\mathcal{L}} \mid \forall N \in \Lambda \ : \ (M\ N) \in \Lambda'\}$$

Lemma 7.8 **SAT** *is closed under function spaces, i.e. if* $\Lambda, \Lambda' \in \mathbf{SAT}$, *then* $\Lambda{\to}\Lambda' \in \mathbf{SAT}$.

Proof. We show

1. $\Lambda{\to}\Lambda' \subseteq \mathcal{SN}_{\mathcal{L}}$.

   Let M be in $\Lambda{\to}\Lambda'$, by Def. 7.7 and Def. 7.3(1), $(M\ N) \in \Lambda' \subseteq \mathcal{SN}_{\mathcal{L}}$ for all $N \in \Lambda$. Thus, $M \in \mathcal{SN}_{\mathcal{L}}$.

2. If $M \in \Lambda{\to}\Lambda'$ and $M \xrightarrow{\beta_{\mathcal{L}}} M'$, then $M' \in \Lambda{\to}\Lambda'$.

   Let N be in Λ, we show that $(M'\ N) \in \Lambda'$. By hypothesis, we have $(M\ N) \xrightarrow{\beta_{\mathcal{L}}} (M'\ N)$, and $(M\ N) \in \Lambda'$. Thus, by Def. 7.3(2), $(M'\ N) \in \Lambda'$.

3. If $M \in \mathcal{NT}$, and whenever we reduce a $\beta_{\mathcal{L}}$-redex of M we obtain a term $M' \in \Lambda{\to}\Lambda'$, then $M \in \Lambda{\to}\Lambda'$.

   Let N be in Λ, we show that $(M\ N) \in \Lambda'$. Since $(M\ N) \in \mathcal{NT}$, then by Def. 7.3(3), it suffices to prove that if $(M\ N) \xrightarrow{\beta_{\mathcal{L}}} M''$, then $M'' \in \Lambda'$. We have $N \in \Lambda \subseteq \mathcal{SN}_{\mathcal{L}}$, so we can reason by induction on $\nu(N)$. In one step, $(M\ N)$ $\beta_{\mathcal{L}}$-reduces to

   - $(M'\ N)$, with $M \xrightarrow{\beta_{\mathcal{L}}} M'$. By hypothesis, $M' \in \Lambda{\to}\Lambda'$ and $N \in \Lambda$, thus $(M'\ N) \in \Lambda'$.
   - $(M\ N')$, with $N \xrightarrow{\beta_{\mathcal{L}}} N'$. By Def. 7.3(2), $N' \in \Lambda$, and $\nu(N') < \nu(N)$, so by induction hypothesis, $(M\ N') \in \Lambda'$.
   - There is no other possibility since $M \in \mathcal{NT}$.

□

Definition 7.9 *The* type interpretation function *is defined inductively on types as follows:*

$$\begin{array}{lcll} [\![\iota]\!] & = & \mathcal{SN}_{\mathcal{L}} & \text{if } \iota \text{ is a basic type} \\ [\![A{\to}B]\!] & = & [\![A]\!]{\to}[\![B]\!] & \end{array}$$

Corollary 7.10 *By Lemma 7.8, for any type A, $[\![A]\!] \in \mathbf{SAT}$.*

Definition 7.11 *The* set of valuations *of Γ, denoted by $[\![\Gamma]\!]$, is a set of substitutions in $\mathcal{NF}_\mathcal{L}$ defined inductively on Γ as follows:*

$$\begin{array}{lcl} [\![nil]\!] & = & \{\uparrow^n \mid \textit{for any natural } n\} \\ [\![\Gamma'.A]\!] & = & [\![nil]\!] \cup \{S \cdot M \in \mathcal{NF}_\mathcal{L} \mid M \in [\![A]\!], S \in [\![\Gamma']\!]\} \end{array}$$

Note that if $M \in [\![A]\!]$ and $S \in [\![\Gamma]\!]$, then $S \cdot M$ is not necessarily in $[\![\Gamma.A]\!]$ (since $S \cdot M$ may not be in $\mathcal{NF}_\mathcal{L}$, e.g. $\uparrow^{Suc(0)} \cdot \mathbf{1} \xrightarrow{\text{(Shift0)}} \uparrow^0$). However, we verify easily the following property.

Lemma 7.12 *If $M \in [\![A]\!]$ and $S \in [\![\Gamma]\!]$, then $(S \cdot M)\downarrow_\mathcal{L} \in [\![\Gamma.A]\!]$.*

Proof. There are two cases: $(S \cdot M)\downarrow_\mathcal{L} = S \cdot M$ or $(S \cdot M)\downarrow_\mathcal{L} = \uparrow^n$. In both cases we verify that $(S \cdot M)\downarrow_\mathcal{L} \in [\![\Gamma.A]\!]$. □

We verify that valuations are $\beta_\mathcal{L}$-strongly normalizing.

Lemma 7.13 *For any Γ, $[\![\Gamma]\!] \subseteq \mathcal{SN}_\mathcal{L}$.*

Proof. We prove by structural induction on S that if $S \in [\![\Gamma]\!]$, then $S \in \mathcal{SN}_\mathcal{L}$.

- $S = \uparrow^n$. In this case S is a $\beta_\mathcal{L}$-normal form, then the conclusion is trivial.
- $S = T \cdot M$. By Def. 7.11, we have $\Gamma = \Gamma'.A$, $T \in [\![\Gamma']\!]$ and $M \in [\![A]\!] \subseteq \mathcal{SN}_\mathcal{L}$. By induction hypothesis, $T \in \mathcal{SN}_\mathcal{L}$. We prove by induction on $\nu(M) + \nu(T)$ that $T \cdot M \in \mathcal{SN}_\mathcal{L}$ (note that $T \cdot M \in \mathcal{NF}_\mathcal{L}$).

Definition 7.14 *Let M, S be in $\mathcal{NF}_\mathcal{L}$, we define*

1. *Γ satisfies that M is of type A, denoted by $\Gamma \models M : A$, if and only if $([T]M)\downarrow_\mathcal{L} \in [\![A]\!]$ for any $T \in [\![\Gamma]\!]$.*
2. *Γ satisfies that S is of type Δ, denoted by $\Gamma \models S \triangleright \Delta$, if and only if $(T \circ S)\downarrow_\mathcal{L} \in [\![\Delta]\!]$ for any $T \in [\![\Gamma]\!]$.*

7.1.2 Weak Normalization Proof for $\lambda_\mathcal{L}$

We need the following technical lemmas about $\beta_\mathcal{L}$-reductions.

Lemma 7.15 *Let M, S be in $\mathcal{NF}_\mathcal{L}$, for any substitution T*

1. *if $M \xrightarrow{\beta_\mathcal{L}} M'$, then $([T]M)\downarrow_\mathcal{L} \xrightarrow{\beta_\mathcal{L}} ([T]M')\downarrow_\mathcal{L}$, and*
2. *if $S \xrightarrow{\beta_\mathcal{L}} S'$, then $(T \circ S)\downarrow_\mathcal{L} \xrightarrow{\beta_\mathcal{L}} (T \circ S')\downarrow_\mathcal{L}$.*

Proof. By simultaneous structural induction on M and S. □

Corollary 7.16 *Let M, S be in $\mathcal{NF}_\mathcal{L}$, for any substitution T*

1. *if* $([T]M)\downarrow_{\mathcal{L}} \in \mathcal{SN}_{\mathcal{L}}$, *then* $M \in \mathcal{SN}_{\mathcal{L}}$, *and*
2. *if* $(T \circ S)\downarrow_{\mathcal{L}} \in \mathcal{SN}_{\mathcal{L}}$, *then* $S \in \mathcal{SN}_{\mathcal{L}}$.

Lemma 7.17 *Let* M *be in* $\mathcal{NF}_{\mathcal{L}}$, *if for all* $N \in [\![A]\!]$, $([\uparrow^0 \cdot N]M)\downarrow_{\mathcal{L}} \in [\![B]\!]$, *then* $\lambda_A.M \in [\![A]\!] \rightarrow [\![B]\!]$.

Proof. Let N be in $[\![A]\!]$, we show that $(\lambda_A.M\ N) \in [\![B]\!]$. Since $(\lambda_A.M\ N) \in \mathcal{NT}$ and $[\![B]\!] \in \mathbf{SAT}$, it suffices to prove that if $(\lambda_A.M\ N) \xrightarrow{\beta_{\mathcal{L}}} M''$, then $M'' \in [\![B]\!]$. We have $([\uparrow^0 \cdot N]M)\downarrow_{\mathcal{L}} \in [\![B]\!] \subseteq \mathcal{SN}_{\mathcal{L}}$, so by Corollary 7.16, $M \in \mathcal{SN}_{\mathcal{L}}$; and by hypothesis, $N \in [\![A]\!] \subseteq \mathcal{SN}_{\mathcal{L}}$. Thus, we can reason by induction on $\nu(M) + \nu(N)$. In one step, $(\lambda_A.M\ N)$ $\beta_{\mathcal{L}}$-reduces to

- $([\uparrow^0 \cdot N]M)\downarrow_{\mathcal{L}}$. By hypothesis, $([\uparrow^0 \cdot N]M)\downarrow_{\mathcal{L}} \in [\![B]\!]$.
- $(\lambda_A.M'\ N)$, with $M \xrightarrow{\beta_{\mathcal{L}}} M'$. By Lemma 7.15, $([\uparrow^0 \cdot N]M)\downarrow_{\mathcal{L}} \xrightarrow{\beta_{\mathcal{L}}} ([\uparrow^0 \cdot N]M')\downarrow_{\mathcal{L}}$. Since $([\uparrow^0 \cdot N]M)\downarrow_{\mathcal{L}} \in [\![B]\!]$, and by Def. 7.3(2), we have $([\uparrow^0 \cdot N]M')\downarrow_{\mathcal{L}} \in [\![B]\!]$. But also, $\nu(M') < \nu(M)$, so by induction hypothesis, $(\lambda_A.M'\ N) \in [\![B]\!]$.
- $(\lambda_A.M\ N')$, with $N \xrightarrow{\beta_{\mathcal{L}}} N'$. By Def. 7.3(2), $N' \in [\![A]\!]$, so by hypothesis, $([\uparrow^0 \cdot N']M)\downarrow_{\mathcal{L}} \in [\![B]\!]$. But also, $\nu(N') < \nu(N)$, so by induction hypothesis, $(\lambda_A.M\ N') \in [\![B]\!]$.

□

Weak normalization proof is based on the following proposition.

Proposition 7.18 (Soundness of $\models$)

1. *If* $\Gamma \vdash M : A$, *then* $\Gamma \models M : A$, *and*
2. *if* $\Gamma \vdash S \triangleright \Delta$, *then* $\Gamma \models S \triangleright \Delta$.

Proof. By simultaneous induction on derivations $\Gamma \vdash M : A$ and $\Gamma \vdash S \triangleright \Delta$. We reason according to the last applied rule.

- (Var). In this case, $M = \mathbf{1}$ and $\Gamma = \Gamma'.A$. Let T be in $[\![\Gamma]\!]$, there are three cases:
  - $T = \uparrow^0$. Therefore, $([T]\mathbf{1})\downarrow_{\mathcal{L}} = \mathbf{1}$. But $\mathbf{1}$ is a neutral $\beta_{\mathcal{L}}$-normal form, then by Remark 7.4, $\mathbf{1} \in [\![A]\!]$.
  - $T = \uparrow^{Suc(n)}$. Therefore, $([T]\mathbf{1})\downarrow_{\mathcal{L}} = [\uparrow^{Suc(n)}]\mathbf{1}$. But $[\uparrow^{Suc(n)}]\mathbf{1}$ is a neutral $\beta_{\mathcal{L}}$-normal form, then by Remark 7.4, $[\uparrow^{Suc(n)}]\mathbf{1} \in [\![A]\!]$.
  - $T = S' \cdot M'$. Therefore, $([T]\mathbf{1})\downarrow_{\mathcal{L}} = M'$. By Def. 7.11 and hypothesis $\Gamma = \Gamma'.A$, we have $M \in [\![A]\!]$.
- (Clos). In this case, $M = [S']M'$, $\Gamma \vdash S \triangleright \Delta$, and $\Delta \vdash M' : A$. We reason by cases analysis on M' and S'.
  - $M' = \mathbf{1}$ and $S' = \uparrow^{Suc(n)}$. Let T be in $[\![\Gamma]\!]$, by induction hypothesis, $(T \circ \uparrow^{Suc(n)})\downarrow_{\mathcal{L}} \in [\![\Delta]\!]$. Note that $([T][\uparrow^{Suc(n)}]\mathbf{1})\downarrow_{\mathcal{L}} = ([T \circ \uparrow^{Suc(n)}]\mathbf{1})\downarrow_{\mathcal{L}} = ([(T \circ \uparrow^{Suc(n)})\downarrow_{\mathcal{L}}]\mathbf{1})\downarrow_{\mathcal{L}}$. By induction hypothesis, $([(T \circ \uparrow^{Suc(n)})\downarrow_{\mathcal{L}}]\mathbf{1})\downarrow_{\mathcal{L}} \in [\![A]\!]$, and thus, $([T][\uparrow^{Suc(n)}]\mathbf{1})\downarrow_{\mathcal{L}} \in [\![A]\!]$.

  - $M = X$ (X is a meta-variable). Let T be in $[\![\Gamma]\!]$, by induction hypothesis, $(T \circ S')\downarrow_{\mathcal{L}} \in [\![\Delta]\!]$, and by Lemma 7.13, $(T \circ S')\downarrow_{\mathcal{L}} \in \mathcal{SN}_{\mathcal{L}}$. Note that $([T][S']X)\downarrow_{\mathcal{L}} = ([T \circ S']X)\downarrow_{\mathcal{L}} = ([(T \circ S')\downarrow_{\mathcal{L}}]X)\downarrow_{\mathcal{L}}$. By Lemma 7.5, $([(T \circ S')\downarrow_{\mathcal{L}}]X)\downarrow_{\mathcal{L}} \in [\![A]\!]$, and thus, $([T][S']X)\downarrow_{\mathcal{L}} \in [\![A]\!]$.

- (Meta$_X$). In this case, $M = X$ (X is a meta-variable). Let T be in $[\![\Gamma]\!]$, there are two cases:
  - $T = \uparrow^0$. Therefore, $([T]X)\downarrow_{\mathcal{L}} = X$. But X is a neutral $\beta_{\mathcal{L}}$-normal form, then by Remark 7.4, $X \in [\![A]\!]$.
  - $T \neq \uparrow^0$. Therefore, $([T]X)\downarrow_{\mathcal{L}} = [T]X$. By Lemma 7.13, $T \in \mathcal{SN}_{\mathcal{L}}$, then by Lemma 7.5, $[T]X \in [\![A]\!]$.

- (Abs). In this case, $M = \lambda_{A_1}.M_1$, $\Gamma.A_1 \vdash M_1 : B_1$, and $A = A_1 \to B_1$. By Def. 7.9, $[\![A]\!] = [\![A_1 \to B_1]\!] = [\![A_1]\!] \to [\![B_1]\!]$. Let T be in $[\![\Gamma]\!]$ and $\Uparrow(T)$ be a notation for $(\uparrow^{Suc(0)} \circ T) \cdot \mathbf{1}$. We have $([T](\lambda_{A_1}.M_1))\downarrow_{\mathcal{L}} = \lambda_{A_1}.([\Uparrow(T)]M_1)\downarrow_{\mathcal{L}}$. By Lemma 7.17, it suffices to prove that for any $N \in [\![A_1]\!]$, $([\uparrow^0 \cdot N]([\Uparrow(T)]M_1)\downarrow_{\mathcal{L}})\downarrow_{\mathcal{L}} \in [\![B_1]\!]$. By hypothesis and Lemma 7.12, $(T \cdot N)\downarrow_{\mathcal{L}} \in [\![\Gamma.A_1]\!]$, then by induction hypothesis, $([(T \cdot N)\downarrow_{\mathcal{L}}]M)\downarrow_{\mathcal{L}} = ([\uparrow^0 \cdot N]([\Uparrow(T)]M_1)\downarrow_{\mathcal{L}})\downarrow_{\mathcal{L}} \in [\![B_1]\!]$.

- (Appl). In this case, $M = (M_1\ N_1)$, $\Gamma \vdash M_1 : B \to A$ and $\Gamma \vdash N_1 : B$. Let T be in $[\![\Gamma]\!]$, so we have $([T](M_1\ N_1))\downarrow_{\mathcal{L}} = (([T]M_1)\downarrow_{\mathcal{L}}\ ([T]N_1)\downarrow_{\mathcal{L}})$. By induction hypothesis, $([T]M_1)\downarrow_{\mathcal{L}} \in [\![B \to A]\!] = [\![B]\!] \to [\![A]\!]$ and $([T]N_1)\downarrow_{\mathcal{L}} \in [\![B]\!]$. Hence, $([T](M_1\ N_1))\downarrow_{\mathcal{L}} \in [\![A]\!]$.

- (Id), (Shift). In this case, $S = \uparrow^n$. We prove by structural induction on n and T that if $T \in [\![\Gamma]\!]$ and $\Gamma \vdash \uparrow^n \triangleright \Delta$, then $(T \circ \uparrow^n)\downarrow_{\mathcal{L}} \in [\![\Delta]\!]$.

- (Cons). In this case, $S = S' \cdot M'$, $\Gamma \vdash M' : A'$, $\Gamma \vdash S' \triangleright \Delta'$ and $\Delta'.A' = \Delta$. Let T be in $[\![\Gamma]\!]$, so we have $(T \circ S)\downarrow_{\mathcal{L}} = ((T \circ S')\downarrow_{\mathcal{L}} \cdot ([T]M')\downarrow_{\mathcal{L}})\downarrow_{\mathcal{L}}$. By induction hypothesis, $([T]M')\downarrow_{\mathcal{L}} \in [\![A']\!]$ and $(T \circ S')\downarrow_{\mathcal{L}} \in [\![\Delta']\!]$. From Lemma 7.12, we conclude $((T \circ S')\downarrow_{\mathcal{L}} \cdot ([T]M')\downarrow_{\mathcal{L}})\downarrow_{\mathcal{L}} \in [\![\Delta]\!]$.

□

Now, we show that $\beta_{\mathcal{L}}$ is strongly normalizing.

Lemma 7.19 *Let M, S be expressions in $\mathcal{NF}_{\mathcal{L}}$,*

1. *if $\Gamma \vdash M : A$, then $M \in \mathcal{SN}_{\mathcal{L}}$, and*
2. *if $\Gamma \vdash S \triangleright \Delta$, then $S \in \mathcal{SN}_{\mathcal{L}}$.*

Proof. By Def. 7.11, $\uparrow^0 \in [\![\Gamma]\!]$. Hence,

1. By Proposition 7.18, $([\uparrow^0]M)\downarrow_{\mathcal{L}} = M \in [\![A]\!]$. By Corollary 7.10 and Def. 7.3(1), $[\![A]\!] \subseteq \mathcal{SN}_{\mathcal{L}}$.
2. By Proposition 7.18, $(\uparrow^0 \circ S)\downarrow_{\mathcal{L}} = S \in [\![\Delta]\!]$, and by Lemma 7.13, $[\![\Delta]\!] \subseteq \mathcal{SN}_{\mathcal{L}}$.

□

Finally, we prove weak normalization on simply-typed $\lambda_{\mathcal{L}}$-expressions.

Theorem 7.20 (Weak Normalization)

1. *If $\Gamma \vdash M : A$, then M is weakly normalizing, and*

2. *if* $\Gamma \vdash S \triangleright \Delta$, *then* S *is weakly normalizing.*

Therefore, M *and* S *have* $\lambda_{\mathcal{L}}$*-normal forms.*

Proof. Let $N = (M)\downarrow_{\mathcal{L}}$ and $T = (S)\downarrow_{\mathcal{L}}$, the subject reduction theorem (Theorem 3.9) says that typing is preserved under reductions, hence $\Gamma \vdash N : A$ and $\Gamma \vdash T \triangleright \Delta$. Therefore, by Lemma 7.19, N and T are both in $\mathcal{SN}_{\mathcal{L}}$. Finally, note that $\beta_{\mathcal{L}}$-normal forms in $\mathcal{NF}_{\mathcal{L}}$ are $\lambda_{\mathcal{L}}$-normal forms too. □

7.2 Systems $\mathrm{CC}_{\mathcal{L}}$ and $\lambda\Pi_{\mathcal{L}}$

Saturated sets for $\lambda\Pi_{\mathcal{L}}$ and $\mathrm{CC}_{\mathcal{L}}$ are defined on the set of $\lambda_{\mathcal{L}\Pi}$-expressions.

As we have said in Chapter 3, the $\mathcal{L}_\Pi$-calculus is terminating but not confluent on pre-expressions. For this reason, saturated sets in the $\lambda_{\mathcal{L}\Pi}$-calculus are redefined so as to deal with this problem.

We use $(x)\downarrow_{\Pi_{\mathcal{L}}}$ to denote the set of $\mathcal{L}_\Pi$-normal forms of x. Since the $\lambda_{\mathcal{L}\Pi}$-calculus without annotations of types in substitutions is confluent on semi-open pre-expressions (Lemma 5.2), we have the following property.

Remark 7.21 *For any* $M_1, M_2 \in (M)\downarrow_{\Pi_{\mathcal{L}}}$, $|M_1| = |M_2|$. *We write* $(M)\downarrow_{|\mathcal{L}_\Pi|}$ *to denote the* $|\mathcal{L}_\Pi|$*-normal form of* $|M|$.

We define $\mathcal{NF}_{\mathcal{L}}$ as the set of all the $\mathcal{L}_\Pi$-normal forms of semi-open expressions.

Definition 7.22 *Let* x, y *be in* $\mathcal{NF}_{\mathcal{L}}$, *we say that* x $\beta\Pi_{\mathcal{L}}$-converts *to* y, *denoted by* $x \xrightarrow{\beta\Pi_{\mathcal{L}}} y$, *if* $x \xrightarrow{(\mathrm{Beta})} w$ *and* $y \in (w)\downarrow_{\Pi_{\mathcal{L}}}$.

We denote by $\mathcal{SN}_{\mathcal{L}}$ the set of $\beta\Pi_{\mathcal{L}}$-strongly normalizing expressions of $\mathcal{NF}_{\mathcal{L}}$.

Definition 7.23 *Let* M *be in* $\mathcal{NF}_{\mathcal{L}}$, M *is* neutral *if it does not have the form* $\lambda_{M_1}.M_2$. *The set of neutral terms is denoted by* $\mathcal{NT}$.

Definition 7.24 *Let* x *be in* $\mathcal{NF}_{\mathcal{L}}$, *the set of* domains *of* x, *denoted by* $\mathrm{D}(x)$, *is defined inductively as follows:*

$$
\begin{array}{lcl}
\mathrm{D}(x) & = & \emptyset \quad \textit{if } x \in \{\mathit{Kind}, \mathit{Type}\} \textit{ or } x = \mathbf{1} \textit{ or } x = \uparrow^n \textit{ or } x \textit{ is a meta-variable} \\
\mathrm{D}(\Pi_{M_1}.M_2) & = & \mathrm{D}(M_1) \cup \mathrm{D}(M_2) \\
\mathrm{D}(\lambda_{M_1}.M_2) & = & \mathrm{D}(M_1) \cup \mathrm{D}(M_2) \\
\mathrm{D}(M\ N) & = & \mathrm{D}(M) \cup \mathrm{D}(N) \\
\mathrm{D}([S]M) & = & \mathrm{D}(S) \cup \mathrm{D}(M) \\
\mathrm{D}(T \circ S) & = & \mathrm{D}(T) \cup \mathrm{D}(S) \\
\mathrm{D}(S \cdot M_{:N}) & = & \{N\} \cup \mathrm{D}(S) \cup \mathrm{D}(M)
\end{array}
$$

Definition 7.25 *A set of terms* $\Lambda \subseteq \mathcal{NF}_{\mathcal{L}}$ *is* saturated *if*

1. $\Lambda \subseteq \mathcal{SN}_{\mathcal{L}}$.

2. *If* $M \in \Lambda$ *and* $M \xrightarrow{\beta\Pi_{\mathcal{L}}} M'$, *then* $M' \in \Lambda$.

3. *If* $M \in \mathcal{NT}$, *and whenever we reduce a* $\beta\Pi_{\mathcal{L}}$*-redex of* M *we obtain a term* $M' \in \Lambda$, *then* $M \in \Lambda$.

4. *If* $M \in \Lambda$, $|M| = |M'|$ *and* $\mathrm{D}(M') \subseteq \mathcal{SN}_{\mathcal{L}}$, *then* $M' \in \Lambda$.

The set of saturated sets is denoted by **SAT**.

The following corollary is a particular case of Def. 7.25(3).

Corollary 7.26 *Let* M *be in* $\mathcal{NT}$ *such that* M *is a* $\beta\Pi_{\mathcal{L}}$*-normal form. For any* $\Lambda \in \mathbf{SAT}$, $M \in \Lambda$.

Lemma 7.27 *For any* $\Lambda \in \mathbf{SAT}$, *substitution* $S \in \mathcal{SN}_{\mathcal{L}}$, *and meta-variable* X, $([S]X)\downarrow_{\Pi_{\mathcal{L}}} \subseteq \Lambda$.

Proof. Let M be in $([S]X)\downarrow_{\Pi_{\mathcal{L}}}$, we reason by induction on $\nu(S)$. Terms in $([S]X)\downarrow_{\Pi_{\mathcal{L}}}$ are neutral, then by Def. 7.25(3), it suffices to consider the reductions of M.

- $M \xrightarrow{\beta\Pi_{\mathcal{L}}} X$. By Corollary 7.26, $X \in \Lambda$.
- $M \xrightarrow{\beta\Pi_{\mathcal{L}}} [S']X$, with $S \xrightarrow{\beta\Pi_{\mathcal{L}}} S'$. By hypothesis, $S' \in \mathcal{SN}_{\mathcal{L}}$ and $\nu(S') < \nu(S)$, so by induction hypothesis, $([S']X)\downarrow_{\Pi_{\mathcal{L}}} = \{[S']X\} \subseteq \Lambda$.

In every case, M $\beta\Pi_{\mathcal{L}}$-reduces into terms in Λ, thus by Def. 7.25(3), $([S]X)\downarrow_{\Pi_{\mathcal{L}}} \subseteq \Lambda$. □

Lemma 7.28 *For any* $\Lambda \in \mathbf{SAT}$, *and terms* $M, N \in \mathcal{SN}_{\mathcal{L}}$, $\Pi_M.N \in \Lambda$.

Proof. The term $\Pi_M.N$ is neutral, then by Def. 7.25(3), it suffices to consider the reductions of $\Pi_M.N$. We reason by induction on $\nu(M) + \nu(N)$. □

Lemma 7.29 $\mathcal{SN}_{\mathcal{L}} \in \mathbf{SAT}$.

Proof. We verify easily the following conditions.

1. $\mathcal{SN}_{\mathcal{L}} \subseteq \mathcal{SN}_{\mathcal{L}}$.
2. If $M \in \mathcal{SN}_{\mathcal{L}}$ and $M \xrightarrow{\beta\Pi_{\mathcal{L}}} M'$, then $M' \in \mathcal{SN}_{\mathcal{L}}$.
3. If $M \in \mathcal{NT}$, and whenever we reduce a $\beta\Pi_{\mathcal{L}}$-redex of M we obtain a term $M' \in \mathcal{SN}_{\mathcal{L}}$, then $M \in \mathcal{SN}_{\mathcal{L}}$.
4. If $M \in \mathcal{SN}_{\mathcal{L}}$, $|M| = |M'|$ and $\mathrm{D}(M') \subseteq \mathcal{SN}_{\mathcal{L}}$, then $M' \in \mathcal{SN}_{\mathcal{L}}$.

□

Definition 7.30 *If* $\Lambda, \Lambda' \in \mathbf{SAT}$, *we define the set*

$$\Lambda{\rightarrow}\Lambda' = \{M \in \mathcal{NF}_{\mathcal{L}} \mid \forall N \in \Lambda, (M\ N) \in \Lambda'\}$$

Lemma 7.31 **SAT** *is closed under function spaces, i.e. if* $\Lambda, \Lambda' \in \mathbf{SAT}$, *then* $\Lambda{\rightarrow}\Lambda' \in \mathbf{SAT}$.

Proof. We verify:

1. $\Lambda{\to}\Lambda' \subseteq \mathcal{SN}_{\mathcal{L}}$.

   Let M be in $\Lambda{\to}\Lambda'$, by Def. 7.30 and Def. 7.25(1), $(M\ N) \in \Lambda' \subseteq \mathcal{SN}_{\mathcal{L}}$ for all $N \in \Lambda$. Thus, $M \in \mathcal{SN}_{\mathcal{L}}$.

2. If $M \in \Lambda{\to}\Lambda'$ and $M \xrightarrow{\beta\Pi_{\mathcal{L}}} M'$, then $M' \in \Lambda{\to}\Lambda'$.

   Let N be in Λ, we show that $(M'\ N) \in \Lambda'$. By hypothesis, $(M\ N) \in \Lambda'$ and $(M\ N) \xrightarrow{\beta\Pi_{\mathcal{L}}} (M'\ N)$. Thus, by Def. 7.25(2), $(M'\ N) \in \Lambda'$.

3. If $M \in \mathcal{NT}$, and whenever we reduce a $\beta\Pi_{\mathcal{L}}$-redex of M we obtain a term $M' \in \Lambda{\to}\Lambda'$, then $M \in \Lambda{\to}\Lambda'$.

   Let N be in Λ, we show that $(M\ N) \in \Lambda'$. Since $(M\ N) \in \mathcal{NT}$, then by Def. 7.25(3), it suffices to prove that if $(M\ N) \xrightarrow{\beta\Pi_{\mathcal{L}}} M''$, then $M'' \in \Lambda'$. We have $N \in \Lambda \subseteq \mathcal{SN}_{\mathcal{L}}$, so we can reason by induction on $\nu(N)$. In one step, $(M\ N)$ $\beta\Pi_{\mathcal{L}}$-reduces to

   - $(M'\ N)$, with $M \xrightarrow{\beta\Pi_{\mathcal{L}}} M'$. By hypothesis, $M' \in \Lambda{\to}\Lambda'$ and $N \in \Lambda$, thus $(M'\ N) \in \Lambda'$.
   - $(M\ N')$, with $N \xrightarrow{\beta\Pi_{\mathcal{L}}} N'$. By Def. 7.25(2), $N' \in \Lambda$, and $\nu(N') < \nu(N)$, so by induction hypothesis, $(M\ N') \in \Lambda'$.
   - There is no other possibility since $M \in \mathcal{NT}$.

4. If $M \in \Lambda{\to}\Lambda'$, $|M| = |M'|$ and $\mathrm{D}(M') \subseteq \mathcal{SN}_{\mathcal{L}}$, then $M' \in \Lambda{\to}\Lambda'$.

   Let N be in Λ, we show that $(M'\ N) \in \Lambda'$. By hypothesis, $(M\ N) \in \Lambda'$, but also, $|(M\ N)| = |(M'\ N)|$. By Def. 7.25(4), it suffices to show that $\mathrm{D}(M'\ N) \subseteq \mathcal{SN}_{\mathcal{L}}$. Since $N \in \Lambda \subseteq \mathcal{SN}_{\mathcal{L}}$, we have $\mathrm{D}(N) \subseteq \mathcal{SN}_{\mathcal{L}}$. Therefore, $\mathrm{D}(M'\ N) = \mathrm{D}(M') \cup \mathrm{D}(N) \subseteq \mathcal{SN}_{\mathcal{L}}$.

□

The following lemma is used in the proof of weak normalization for $\mathrm{CC}_{\mathcal{L}}$.

Lemma 7.32 *Saturated sets are closed under arbitrary intersections, i.e. for I a set and $\Lambda_i \in \mathbf{SAT}$ for all $i \in I$, we have $\cap_{i\in I}\Lambda_i \in \mathbf{SAT}$.*

Proof. We verify

1. $\cap_{i\in I}\Lambda_i \subseteq \mathcal{SN}_{\mathcal{L}}$.

2. If $M \in \cap_{i\in I}\Lambda_i$ and $M \xrightarrow{\beta\Pi_{\mathcal{L}}} M'$, then $M' \in \cap_{i\in I}\Lambda_i$.

3. If $M \in \mathcal{NT}$, and whenever we reduce a $\beta\Pi_{\mathcal{L}}$-redex of M we obtain a term $M' \in \cap_{i\in I}\Lambda_i$, then $M \in \cap_{i\in I}\Lambda_i$.

4. If $M \in \cap_{i\in I}\Lambda_i$, $|M| = |M'|$ and $\mathrm{D}(M') \subseteq \mathcal{SN}_{\mathcal{L}}$, then $M' \in \cap_{i\in I}\Lambda_i$.

□

7.2.1 Interpretation of Terms in $\lambda\Pi_{\mathcal{L}}$

Type interpretation in $\lambda\Pi_{\mathcal{L}}$ is just a generalization of type interpretation in simple types.

Definition 7.33 *The* type interpretation function *of terms in $\lambda\Pi_{\mathcal{L}}$ is defined inductively as follows:*

$$\begin{array}{lcll} [\![x]\!] & = & \mathcal{SN}_{\mathcal{L}} & \text{if } x \in \{\mathit{Kind}, \mathit{Type}\} \text{ or } x = \mathbf{1} \text{ or } x \text{ is a meta-variable} \\ [\![[S]M]\!] & = & [\![M]\!] & \\ [\![(M\ N)]\!] & = & [\![M]\!] & \\ [\![\lambda_M.N]\!] & = & [\![N]\!] & \\ [\![\Pi_M.N]\!] & = & [\![M]\!] \to [\![N]\!] & \end{array}$$

We have the following corollary of Lemma 7.31.

Corollary 7.34 *For any term M, $[\![M]\!] \in \mathbf{SAT}$.*

Definition 7.35 *The* set of $\lambda\Pi_{\mathcal{L}}$-valuations *of Γ, denoted by $[\![\Gamma]\!]$, is a set of substitutions in $\mathcal{NF}_{\mathcal{L}}$ defined inductively on Γ as follows:*

$$\begin{array}{lcl} [\![\mathit{nil}]\!] & = & \{\uparrow^n \mid \text{for any natural } n\} \\ [\![\Gamma'.N']\!] & = & [\![\mathit{nil}]\!] \cup \{S \cdot M_{:N} \in \mathcal{NF}_{\mathcal{L}} \mid S \in [\![\Gamma']\!], M \in [\![N]\!], N \in \mathcal{SN}_{\mathcal{L}}, [\![N]\!] = [\![N']\!]\} \end{array}$$

Lemma 7.36 *For any Γ, $[\![\Gamma]\!] \subseteq \mathcal{SN}_{\mathcal{L}}$.*

Proof. Just as in Lemma 7.13, we show by structural induction on S that if $S \in [\![\Gamma]\!]$, then $S \in \mathcal{SN}_{\mathcal{L}}$. □

Definition 7.37 *Let M, S be in $\mathcal{NF}_{\mathcal{L}}$, we define*

1. *Γ satisfies that M is of type N, denoted by $\Gamma \models M : N$, if and only if $([T]M)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N]\!]$ for any $T \in [\![\Gamma]\!]$.*
2. *Γ satisfies that S is of type Δ, denoted by $\Gamma \models S \triangleright \Delta$, if and only if $(T \circ S)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Delta]\!]$ for any $T \in [\![\Gamma]\!]$.*

7.2.2 Weak Normalization Proof for $\lambda\Pi_{\mathcal{L}}$

First, we show some technical lemmas.

Lemma 7.38 *If $S \in [\![\Gamma]\!]$, $M \in [\![N]\!]$ and $N \in \mathcal{SN}_{\mathcal{L}}$, then $(S \cdot M_{:N})\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Gamma.N]\!]$.*

Proof. Note that $S \cdot M_{:N}$ is not necessarily in $\mathcal{NF}_{\mathcal{L}}$. But there are two cases: $(S \cdot M_{:N})\downarrow_{\Pi_{\mathcal{L}}} = \{S \cdot M_{:N}\}$ or $(S \cdot M_{:N})\downarrow_{\Pi_{\mathcal{L}}} = \{\uparrow^n\}$. In both cases we verify that $(S \cdot M_{:N})\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Gamma.N]\!]$. □

Lemma 7.39 *If $\Sigma :\cdot \Gamma \vdash M : N$ and $\Sigma :\cdot \Gamma \vdash N : \mathit{Type}$, then $[\![M]\!] = \mathcal{SN}_{\mathcal{L}}$.*

Proof. By structural induction on M.

- $M \in \{\mathit{Kind}, \mathit{Type}\}$. This case is not possible by Remark 3.15.
- $M = \mathbf{1}$ or $M = X$ (X a meta-variable). By Def. 7.33, $[\![M]\!] = \mathcal{SN}_{\mathcal{L}}$.

- $M = \Pi_{M_1}.M_2$. By Lemma 6.4 we have $N =_{\lambda_{\mathcal{L}\Pi}} s$. By Geuvers' lemma (Theorem 5.14), $N \xrightarrow{\lambda_{\mathcal{L}\Pi}^*} s$. By subject reduction theorem (Theorem 6.20), $\Sigma :\cdot \Gamma \vdash s : \mathit{Type}$. But this is not possible by Remark 3.15, Lemma 6.4 and Lemma 6.14. So, this case does not apply.

- $M = \lambda_{M_1}.M_2$. By Lemma 6.4 we have $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : N_2$ and $N =_{\lambda_{\mathcal{L}\Pi}} \Pi_{M_1}.N_2$. From hypothesis and Corollary 6.23, $\Sigma :\cdot \Gamma \vdash \Pi_{M_1}.N_2 : \mathit{Type}$. By Lemma 6.4 we have $\Sigma :\cdot \Gamma.M_1 \vdash N_2 : \mathit{Type}$. So, by Def. 7.33 and induction hypothesis, $[\![\lambda_{M_1}.M_2]\!] = [\![M_2]\!] = \mathcal{SN}_{\mathcal{L}}$.

- $M = (M_1\ M_2)$. By Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash M_1 : \Pi_{N_1}.N_2$, $N =_{\lambda_{\mathcal{L}\Pi}} [\uparrow^0 \cdot M_{2:N_1}]N_2$ and $\Sigma :\cdot \Gamma.N_1 \vdash N_2 : s$. From hypothesis and Corollary 6.23, $\Sigma :\cdot \Gamma \vdash [\uparrow^0 \cdot M_{2:N_1}]N_2 : \mathit{Type}$, but also by Lemma 6.3, $\Sigma :\cdot \Gamma \vdash [\uparrow^0 \cdot M_{2:N_1}]N_2 : s$. Hence, by Geuvers' lemma (Theorem 5.14), $s = \mathit{Type}$, and so, $\Sigma :\cdot \Gamma \vdash \Pi_{N_1}.N_2 : \mathit{Type}$. By Def. 7.33 and induction hypothesis, $[\![(M_1\ M_2)]\!] = [\![M_1]\!] = \mathcal{SN}_{\mathcal{L}}$.

- $M = [S]M'$. By Remark 3.15, $N \neq \mathit{Kind}$, so by Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, $\Sigma :\cdot \Delta \vdash M' : N'$, $N' \neq \mathit{Kind}$ and $N =_{\lambda_{\mathcal{L}\Pi}} [S]N'$. By sort soundness theorem (Theorem 6.12), $\Sigma :\cdot \Delta \vdash N' : s$. From hypothesis and Corollary 6.23, $\Sigma :\cdot \Gamma \vdash [S]N' : \mathit{Type}$, but also by Lemma 6.3, $\Sigma :\cdot \Gamma \vdash [S]N' : s$. Hence, by Geuvers' lemma (Theorem 5.14), $s = \mathit{Type}$, and so, by Def. 7.33 and induction hypothesis, $[\![[S]M']\!] = [\![M']\!] = \mathcal{SN}_{\mathcal{L}}$.

□

Lemma 7.40

1. *If* $\Sigma :\cdot \Gamma \vdash M : N_1$ *and* $\Sigma :\cdot \Gamma \vdash M : N_2$, *then* $[\![N_1]\!] = [\![N_2]\!]$, *and*
2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta_1$ *and* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta_2$, *then* $[\![\Delta_1]\!] = [\![\Delta_2]\!]$.

Proof.

1. By type uniqueness theorem (Theorem 6.15), we have $N_1 =_{\lambda_{\mathcal{L}\Pi}} N_2$, and by sort soundness theorem (Theorem 6.12), ($N_1 = N_2 = \mathit{Kind}$) or ($\Sigma :\cdot \Gamma \vdash N_1 : s_1$ and $\Sigma :\cdot \Gamma \vdash N_2 : s_2$). The first case is trivial. For the second one, we use typing soundness theorem (Theorem 6.22) to conclude that N_1 and N_2 are convertible via a path of well typed terms. Hence, it suffices to prove that for any well typed term N if $N \xrightarrow{\beta\Pi_{\mathcal{L}}} N'$, then $[\![N]\!] = [\![N']\!]$. We prove that by induction on the depth of the $\beta\Pi_{\mathcal{L}}$-redex reduced in N. The only interesting case is (VarCons), i.e. $[S \cdot M_{1:M_2}]\mathbf{1} \xrightarrow{(\mathrm{VarCons})} M_1$. From Def. 7.33, $[\![[S \cdot M_{1:M_2}]\mathbf{1}]\!] = [\![\mathbf{1}]\!] = \mathcal{SN}_{\mathcal{L}}$. But if $[S \cdot M_{1:M_2}]\mathbf{1}$ is well-typed in $\Sigma :\cdot \Gamma$, then we conclude $\Sigma :\cdot \Gamma \vdash M_1 : [S]M_2$ and $\Sigma :\cdot \Gamma \vdash [S]M_2 : \mathit{Type}$. Hence, by Lemma 7.39, $[\![M_1]\!] = \mathcal{SN}_{\mathcal{L}}$ too.
2. We proceed by structural induction on Δ_1. We use the first part of the lemma.

□

Lemma 7.41 *Let* M *be in* $\mathcal{NF}_{\mathcal{L}}$ *and* $N_1' \in \mathcal{SN}_{\mathcal{L}}$, *if for all* $N \in [\![N_1]\!]$, $([\uparrow^0 \cdot N_{:N_1'}]M)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N_2]\!]$, *then* $\lambda_{N_1'}.M \in [\![N_1]\!] \to [\![N_2]\!]$.

Proof. Let N be in $[\![N_1]\!]$, we want to show $(\lambda_{N_1'}.M\ N) \in [\![N_2]\!]$. Since $(\lambda_{N_1'}.M\ N) \in \mathcal{NT}$ and $[\![N_2]\!] \subseteq \mathbf{SAT}$, it suffices to prove that if $(\lambda_{N_1'}.M\ N) \xrightarrow{\beta\Pi_{\mathcal{L}}} M''$, then $M'' \in [\![N_2]\!]$. Since for all

$N \in [\![N_1]\!]$, we have $([\uparrow^0 \cdot N_{:N_1'}]M)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![N_2]\!] \subseteq \mathcal{SN}_\mathcal{L}$. In particular, $([\uparrow^0 \cdot \mathbf{1}_{:N_1'}]M)\downarrow_{\Pi_\mathcal{L}} \subseteq \mathcal{SN}_\mathcal{L}$, and thus $M \in \mathcal{SN}_\mathcal{L}$. But also, $N \in [\![N_1]\!] \subseteq \mathcal{SN}_\mathcal{L}$, and $N_1' \in \mathcal{SN}_\mathcal{L}$. Thus, we can reason by induction on $\nu(M) + \nu(N) + \nu(N_1')$. In one step, $(\lambda_{N_1'}.M\ N)$ $\beta\Pi_\mathcal{L}$-reduces to

- $([\uparrow^0 \cdot N_{:N_1'}]M)\downarrow_{\Pi_\mathcal{L}}$. By hypothesis, $([\uparrow^0 \cdot N_{:N_1'}]M)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![N_2]\!]$.

- $(\lambda_{N_1'}.M\ N')$, with $N \xrightarrow{\beta\Pi_\mathcal{L}} N'$. By Def. 7.25(2), $N' \in [\![N_1]\!]$, so by hypothesis, $([\uparrow^0 \cdot N'_{:N_1'}]M)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![N_2]\!]$. But also, $\nu(N') < \nu(N)$, so by induction hypothesis, $(\lambda_{N_1'}.M\ N') \in [\![N_2]\!]$.

- $(\lambda_{N_1'}.M\ N)$, with $N_1' \xrightarrow{\beta\Pi_\mathcal{L}} N_1''$. Since $N_1' \in \mathcal{SN}_\mathcal{L}$, $N_1'' \in \mathcal{SN}_\mathcal{L}$ too. By hypothesis, $([\uparrow^0 \cdot N_{:N_1'}]M)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![N_2]\!]$, but also $([\uparrow^0 \cdot N_{:N_1'}]M)\downarrow_{|\mathcal{L}_\Pi|} = ([\uparrow^0 \cdot N_{:N_1''}]M)\downarrow_{|\mathcal{L}_\Pi|}$. Since $N_1' \in \mathcal{SN}_\mathcal{L}$, we have $N_1'' \in \mathcal{SN}_\mathcal{L}$, and so, for any $M' \in ([\uparrow^0 \cdot N_{:N_1''}]M)\downarrow_{\Pi_\mathcal{L}}$, $\mathrm{D}(M') \subseteq \mathcal{SN}_\mathcal{L}$. Therefore, by Def. 7.25(4), $([\uparrow^0 \cdot N_{:N_1''}]M)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![N_2]\!]$.

  Since $([\uparrow^0 \cdot N_{:N_1''}]M)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![N_2]\!]$ and $\nu(N_1'') < \nu(N_1')$, we conclude with the induction hypothesis that $(\lambda_{N_1''}.M\ N) \in [\![N_2]\!]$.

- $(\lambda_{N_1'}.M'\ N)$, with $M \xrightarrow{\beta\Pi_\mathcal{L}} M'$. Using the properties of $\lambda_{\mathcal{L}\Pi}$ and $|\lambda_{\mathcal{L}\Pi}|$, we have if $M_1 \in ([\uparrow^0 \cdot N_{:N_1'}]M)\downarrow_{\Pi_\mathcal{L}}$, then $M_1 \xrightarrow{\beta\Pi_\mathcal{L}} M_1'$, where $|M_1'| = ([\uparrow^0 \cdot N_{:N_1'}]M')\downarrow_{|\mathcal{L}_\Pi|}$. Since $M_1 \in [\![N_2]\!]$ and by Def. 7.25(2), we have $M_1 \in [\![N_2]\!]$. But also, for any $M_2 \in ([\uparrow^0 \cdot N_{:N_1'}]M')\downarrow_{\Pi_\mathcal{L}}$, $\mathrm{D}(M_2) \subseteq \mathcal{SN}_\mathcal{L}$, thus by Def. 7.25(4), $([\uparrow^0 \cdot N_{:N_1'}]M')\downarrow_{\Pi_\mathcal{L}} \subseteq [\![N_2]\!]$.

  Since $([\uparrow^0 \cdot N_{:N_1'}]M')\downarrow_{\Pi_\mathcal{L}} \subseteq [\![N_2]\!]$ and $\nu(M') < \nu(M)$, we conclude with the induction hypothesis that $(\lambda_{N_1'}.M'\ N) \in [\![N_2]\!]$.

□

Lemma 7.42 *If* $S \in [\![\Gamma]\!]$ *and* $\Sigma :\cdot \Gamma \vdash \uparrow^n \triangleright \Delta$, *then* $(S \circ \uparrow^n)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![\Delta]\!]$.

Proof. By structural induction on n and S.

- $n = 0$. From the typing rules we know that $\Sigma :\cdot \Gamma \vdash \uparrow^0 \triangleright \Gamma$, so by Lemma 7.40(2), $[\![\Gamma]\!] = [\![\Delta]\!]$. Hence, $(S \circ \uparrow^0)\downarrow_{\Pi_\mathcal{L}} = \{S\} \subseteq [\![\Gamma]\!] = [\![\Delta]\!]$.

- $n = Suc(n')$, $S = \uparrow^m$. We have $(\uparrow^m \circ \uparrow^n)\downarrow_{\Pi_\mathcal{L}} = \{\uparrow^{n+m}\} \subseteq [\![\Delta]\!]$.

- $n = Suc(n')$, $S = T \cdot M_{:N}$. We have $\Gamma = \Gamma'.N'$ and $T \in [\![\Gamma']\!]$. From the typing rules we know that $\Sigma :\cdot \Gamma' \vdash \uparrow^{n'} \triangleright \Delta$, so by induction hypothesis, $(T \circ \uparrow^{n'})\downarrow_{\Pi_\mathcal{L}} = ((T \cdot M_{:N}) \circ \uparrow^{Suc(n')})\downarrow_{\Pi_\mathcal{L}} \subseteq [\![\Delta]\!]$.

□

Notation: The expression $\Uparrow_{M_1}(T)$ denotes $(\uparrow^{Suc(0)} \circ T) \cdot \mathbf{1}_{:M_1}$.

Weak normalization proof of $\lambda\Pi_\mathcal{L}$ is based on the following proposition.

Proposition 7.43 (Soundness of $\models$)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, *then* $\Gamma \models M : N$, *and*
2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* $\Gamma \models S \triangleright \Delta$.

Proof. By simultaneous structural induction on M and S.

- $M = \mathit{Kind}$. This case is not possible by Remark 3.15.
- $M = \mathit{Type}$. Let T be in $[\![\Gamma]\!]$, we have $([T]\mathit{Type})\downarrow_{\Pi_{\mathcal{L}}} = \{\mathit{Type}\}$. But Type is a neutral $\beta\Pi_{\mathcal{L}}$-normal form, then by Corollary 7.26, $\mathit{Type} \in [\![N]\!]$.
- $M = \mathbf{1}$. Let T be in $[\![\Gamma]\!]$, there are three cases:
  - $T = \uparrow^0$. Therefore, $([T]\mathbf{1})\downarrow_{\Pi_{\mathcal{L}}} = \{\mathbf{1}\}$. But $\mathbf{1}$ is a neutral $\beta\Pi_{\mathcal{L}}$-normal form, then by Corollary 7.26, $\mathbf{1} \in [\![N]\!]$.
  - $T = \uparrow^{Suc(n)}$. Therefore, $([T]\mathbf{1})\downarrow_{\Pi_{\mathcal{L}}} = \{[\uparrow^{Suc(n)}]\mathbf{1}\}$. But $[\uparrow^{Suc(n)}]\mathbf{1}$ is a neutral $\beta\Pi_{\mathcal{L}}$-normal form, then by Corollary 7.26, $[\uparrow^{Suc(n)}]\mathbf{1} \in [\![N]\!]$.
  - $T = S' \cdot M'_{:N_1}$. Therefore, $([T]\mathbf{1})\downarrow_{\Pi_{\mathcal{L}}} = \{M'\}$. By Def. 7.35, $\Gamma = \Gamma'.N_2$ and $M' \in [\![N_2]\!]$. From the typing rules we know that $\Sigma :\cdot \Gamma \vdash \mathbf{1} : [\uparrow^{Suc(0)}]N_2$. By Lemma 7.40, $[\![N]\!] = [\![[\uparrow^{Suc(0)}]N_2]\!] = [\![N_2]\!]$. Hence, $M' \in [\![N]\!]$.
- $M = [\uparrow^{Suc(n)}]\mathbf{1}$. By Lemma 6.4 and Lemma 6.9 we have $\Sigma :\cdot \Gamma \vdash \uparrow^{Suc(n)} \triangleright \Delta$, $\Sigma :\cdot \Delta \vdash \mathbf{1} : N'$ and $\Sigma :\cdot \Gamma \vdash [\uparrow^{Suc(n)}]\mathbf{1} : [\uparrow^{Suc(n)}]N'$. By Lemma 7.40(1), $[\![N]\!] = [\![[\uparrow^{Suc(n)}]N']\!] = [\![N']\!]$. Let T be in $[\![\Gamma]\!]$, note that if $M' \in ([T][\uparrow^{Suc(n)}]\mathbf{1})\downarrow_{\Pi_{\mathcal{L}}}$, then $M' \in ([T']\mathbf{1})\downarrow_{\Pi_{\mathcal{L}}}$ for some $T' \in (T \circ \uparrow^{Suc(n)})\downarrow_{\Pi_{\mathcal{L}}}$. By induction hypothesis, $(T \circ \uparrow^{Suc(n)})\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Delta]\!]$. Using induction hypothesis again, we have $([T']\mathbf{1})\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N']\!]$. Hence, $([T][\uparrow^{Suc(n)}]\mathbf{1})\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N']\!] = [\![N]\!]$.
- $M = X$ (X is a meta-variable). Let T be in $[\![\Gamma]\!]$, there are two cases:
  - $T = \uparrow^0$. Therefore, $([T]X)\downarrow_{\Pi_{\mathcal{L}}} = \{X\}$. But X is a neutral $\beta\Pi_{\mathcal{L}}$-normal form, then by Corollary 7.26, $X \in [\![N]\!]$.
  - $T \neq \uparrow^0$. Therefore, $([T]X)\downarrow_{\Pi_{\mathcal{L}}} = \{[T]X\}$. By Lemma 7.13, $T \in \mathcal{SN}_{\mathcal{L}}$, then by Lemma 7.27, $[T]X \in [\![N]\!]$.
- $M = [S']X$ (X is a meta-variable). By Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash S' \triangleright \Delta$, $\Sigma :\cdot \Delta \vdash X : N'$ and $\Sigma :\cdot \Gamma \vdash [S']X : [S']N'$. By Lemma 7.40, $[\![N]\!] = [\![[S']N']\!] = [\![N']\!]$. Let T be in $[\![\Gamma]\!]$, note that if $M' \in ([T][S']X)\downarrow_{\Pi_{\mathcal{L}}}$, then $M' \in ([T']X)\downarrow_{\Pi_{\mathcal{L}}}$ for some $T' \in (T \circ S')\downarrow_{\Pi_{\mathcal{L}}}$. By induction hypothesis, $(T \circ S')\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Delta]\!]$, and by Lemma 7.36, $(T \circ S')\downarrow_{\Pi_{\mathcal{L}}} \subseteq \mathcal{SN}_{\mathcal{L}}$. By Lemma 7.27, $([T']X)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N]\!]$. Hence, $([T][S']X)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N]\!]$.
- $M = \Pi_{M_1}.M_2$. By Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash M_1 : \mathit{Type}$ and $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : s$. Let T be in $[\![\Gamma]\!]$. Note that if $M' \in ([T](\Pi_{M_1}.M_2))\downarrow_{\Pi_{\mathcal{L}}}$, then $M' = \Pi_{N_1}.N_2$ where $N_1 \in ([T]M_1)\downarrow_{\Pi_{\mathcal{L}}}$ and $N_2 \in ([\Uparrow_{M_1}(T)]M_2)\downarrow_{\Pi_{\mathcal{L}}}$. By induction hypothesis, $N_1 \in ([T]M_1)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\mathit{Type}]\!] = \mathcal{SN}_{\mathcal{L}}$. In particular, if $T = \uparrow^0$, then $N_1 \in \mathcal{SN}_{\mathcal{L}}$ and $([\uparrow^0 \cdot \mathbf{1}_{:M_1}]N_2)\downarrow_{\Pi_{\mathcal{L}}} \subseteq ([\uparrow^0 \cdot \mathbf{1}_{:M_1}]M_2)\downarrow_{\Pi_{\mathcal{L}}}$. But also by Def. 7.35, $\uparrow^0 \cdot \mathbf{1}_{:M_1} \in [\![\Gamma.M_1]\!]$. Thus, by induction hypothesis, $([\uparrow^0 \cdot \mathbf{1}_{:M_1}]N_2)\downarrow_{\Pi_{\mathcal{L}}} \subseteq ([\uparrow^0 \cdot \mathbf{1}_{:M_1}]M_2)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![s]\!] = \mathcal{SN}_{\mathcal{L}}$. Therefore, $N_2 \in \mathcal{SN}_{\mathcal{L}}$, and by Lemma 7.28, $\Pi_{N_1}.N_2 \in [\![N]\!]$.
- $M = \lambda_{M_1}.M_2$. By Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash M_1 : \mathit{Type}$, $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : N'$ and $\Sigma :\cdot \Gamma \vdash \lambda_{M_1}.M_2 : \Pi_{M_1}.N'$. By Lemma 7.40, $[\![N]\!] = [\![\Pi_{M_1}.N']\!] = [\![M_1]\!] \to [\![N']\!]$. Let T be in $[\![\Gamma]\!]$, note that if $M' \in ([T](\lambda_{M_1}.M_2))\downarrow_{\Pi_{\mathcal{L}}}$, then $M' = \lambda_{M'_1}.M'_2$ where $M'_1 \in ([T]M_1)\downarrow_{\Pi_{\mathcal{L}}}$ and $M'_2 \in ([\Uparrow_{M_1}(T)]M_2)\downarrow_{\Pi_{\mathcal{L}}}$. By induction hypothesis, $M'_1 \in ([T]M_1)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\mathit{Type}]\!] = \mathcal{SN}_{\mathcal{L}}$.

In particular, if $T = \uparrow^0$, $M_1' \in \mathcal{SN}_{\mathcal{L}}$. Now we prove that $\lambda_{M_1'}.M_2' \in [\![M_1]\!] \to [\![N']\!]$. From Lemma 7.41, it suffices to prove that for any $M'' \in [\![M_1]\!]$, $([\uparrow^0 \cdot M''_{:M_1'}]M_2')\downarrow_{\Pi_{\mathcal{L}}} \in [\![N']\!]$. There are two cases:

- $T = \uparrow^0$. By definition of $\lambda_{\mathcal{L}\Pi}$, $([\uparrow^0 \cdot M''_{:M_1'}]M_2')\downarrow_{\Pi_{\mathcal{L}}} \subseteq ([\uparrow^0 \cdot M''_{:M_1'}]M_2)\downarrow_{\Pi_{\mathcal{L}}}$. It is not difficult to see from the typing rules, subject reduction theorem (Theorem 6.20) and Lemma 7.40, that $[\![M_1]\!] = [\![M_1']\!]$. Hence, by Def. 7.35, $\uparrow^0 \cdot M''_{:M_1'} \in [\![\Gamma.M_1]\!]$, and by induction hypothesis, $([\uparrow^0 \cdot M''_{:M_1'}]M_2)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N']\!]$.
- $T \neq \uparrow^0$. By definition of $\lambda_{\mathcal{L}\Pi}$, $([\uparrow^0 \cdot M''_{:M_1'}]M_2')\downarrow_{\Pi_{\mathcal{L}}} \subseteq \bigcup_{S \in (T \cdot M''_{:M_1})\downarrow_{\Pi_{\mathcal{L}}}} ([S]M_2)\downarrow_{\Pi_{\mathcal{L}}}$. Remark that $M'' \in [\![M_1]\!]$, $M_1 \in \mathcal{SN}_{\mathcal{L}}$, and by Def. 7.25 and Def. 7.35, $(T)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Gamma]\!]$. By Lemma 7.38, $(T \cdot M''_{:M_1})\downarrow_{\Pi_{\mathcal{L}}} \subseteq \Gamma.M_1$. Thus, by induction hypothesis, $\bigcup_{S \in (T \cdot M''_{:M_1})\downarrow_{\Pi_{\mathcal{L}}}} ([S]M_2)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N']\!]$.

- $M = (M_1\ M_2)$. By Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash M_1 : \Pi_{N_1}.N_2$, $\Sigma :\cdot \Gamma \vdash M_2 : N_1$ and $\Sigma :\cdot \Gamma \vdash (M_1\ M_2) : [\uparrow^0 \cdot M_{2:N_1}]N_2$. By Lemma 7.40, $[\![N]\!] = [\![[\uparrow^0 \cdot M_{2:N_1}]N_2]\!] = [\![N_2]\!]$. Let T be in $[\![\Gamma]\!]$, note that if $M' \in ([T](M_1\ M_2))\downarrow_{\Pi_{\mathcal{L}}}$, then $M' = (M_1'\ M_2')$ where $M_1' \in ([T]M_1)\downarrow_{\Pi_{\mathcal{L}}}$ and $M_2' \in ([T]M_2)\downarrow_{\Pi_{\mathcal{L}}}$. By induction hypothesis, $([T]M_1)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Pi_{N_1}.N_2]\!] = [\![N_1]\!] \to [\![N_2]\!]$ and $([T]M_2)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N_1]\!]$. Hence, $(M_1'\ M_2') \in [\![N_2]\!] = [\![N]\!]$.

- $S = \uparrow^n$. In this case we apply Lemma 7.42.

- $S = S' \cdot M'_{:N'}$. By Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash S' \triangleright \Delta'$, $\Sigma :\cdot \Gamma \vdash M' : [T]N'$, $\Sigma :\cdot \Delta' \vdash N' : \mathit{Type}$ and $\Sigma :\cdot \Gamma \vdash S' \cdot M'_{:N'} \triangleright \Delta'.N'$. By Lemma 7.40(2), $[\![\Delta]\!] = [\![\Delta'.N']\!]$. Let T be in $[\![\Gamma]\!]$, note that if $S'' \in (T \circ (S' \cdot M'_{:N'}))\downarrow_{\Pi_{\mathcal{L}}}$, then $S'' \in (T' \cdot M''_{:N'})\downarrow_{\Pi_{\mathcal{L}}}$ for some $M'' \in ([T]M')\downarrow_{\Pi_{\mathcal{L}}}$ and $T' \in (T \circ S')\downarrow_{\Pi_{\mathcal{L}}}$. By induction hypothesis, $([T]M')\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![[T]N']\!] = [\![N']\!]$ and $(T \circ S')\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Delta']\!]$. Since $\uparrow^0 \in [\![\Delta']\!]$, we conclude by induction hypothesis that $N' \in ([\uparrow^0]N')\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\mathit{Type}]\!] = \mathcal{SN}_{\mathcal{L}}$. Therefore, by Lemma 7.38, $(T' \cdot M''_{:N'})\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Delta'.N']\!] = [\![\Delta]\!]$.

□

Now, we show that $\beta\Pi_{\mathcal{L}}$ is strongly normalizing.

Lemma 7.44 *Let M, S be expressions in $\mathcal{NF}_{\mathcal{L}}$,*

1. *if $\Sigma :\cdot \Gamma \vdash M : N$, then $M \in \mathcal{SN}_{\mathcal{L}}$, and*
2. *if $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, then $S \in \mathcal{SN}_{\mathcal{L}}$.*

Proof. By Def. 7.35, $\uparrow^0 \in [\![\Gamma]\!]$. Hence,

1. By Proposition 7.43, $M \in ([\uparrow^0]M)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N]\!]$, and by Corollary 7.34 and Def. 7.25(1), $[\![N]\!] \subseteq \mathcal{SN}_{\mathcal{L}}$.
2. By Proposition 7.43, $S \in (\uparrow^0 \circ S)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Delta]\!]$, and by Lemma 7.36, $[\![\Delta]\!] \subseteq \mathcal{SN}_{\mathcal{L}}$.

□

Finally, we prove weak normalization on typed $\lambda_{\mathcal{L}\Pi}$-expressions in $\lambda\Pi_{\mathcal{L}}$.

Theorem 7.45 (Weak Normalization)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, *then* M *is weakly normalizing, and*

2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* S *is weakly normalizing.*

Therefore, M *and* S *have* $\lambda_{\mathcal{L}\Pi}$*-normal forms.*

Proof. By Proposition 3.11, there exists $M', S' \in \mathcal{NF}_{\mathcal{L}}$ such that $M \xrightarrow{\mathcal{L}_{\Pi}^*} M'$ and $S \xrightarrow{\mathcal{L}_{\Pi}^*} S'$. The subject reduction theorem (Theorem 6.20) says that typing is preserved under reductions, hence $\Sigma :\cdot \Gamma \vdash M' : N$ and $\Sigma :\cdot \Gamma \vdash S' \triangleright \Delta$. Therefore, by Lemma 7.44, M' and S' are both in $\mathcal{SN}_{\mathcal{L}}$. Finally, note that $\beta\Pi_{\mathcal{L}}$-normal forms in $\mathcal{NF}_{\mathcal{L}}$ are $\lambda_{\mathcal{L}\Pi}$-normal forms too. □

7.2.3 Interpretation of Terms in $\mathrm{CC}_{\mathcal{L}}$

The interpretation of types in $\mathrm{CC}_{\mathcal{L}}$ is more complex than that of $\lambda\Pi_{\mathcal{L}}$ due to polymorphism and constructions of types.

Typed terms in $\mathrm{CC}_{\mathcal{L}}$ can be classified according to their types as follows:

1. $\Sigma :\cdot \Gamma \vdash M :$ *Kind*. In this case we say that M *is a kind.*

2. $\Sigma :\cdot \Gamma \vdash M :$ *Type*. In this case we say that M *is a type.*

3. $\Sigma :\cdot \Gamma \vdash M : N$ and $\Sigma :\cdot \Gamma \vdash N :$ *Kind*. In this case we say that M *is a constructor.*

4. $\Sigma :\cdot \Gamma \vdash M : N$, $\Sigma :\cdot \Gamma \vdash N :$ *Type*. In this case we say that M is *an object.*

Note that (2) is a particular of case (3), i.e. a type is also a constructor.

In contrast to the proof for $\lambda\Pi_{\mathcal{L}}$, in this proof we consider that all the types, contexts and signatures are in $\mathcal{L}_{\Pi}$-normal form. This restriction is possible because the $\mathcal{L}_{\Pi}$-calculus is strongly normalizing (Proposition 3.11) and typing is preserved under reductions (Theorem 6.20).

Lemma 7.46 *Kinds and constructors in* $\mathcal{L}_{\Pi}$*-normal form are described by the following grammar:*

$$\begin{array}{llll} \textbf{Kinds} & K & ::= & \mathit{Type} \mid \Pi_M.K \mid X \mid [S]X \\ \textbf{Constructors} & A & ::= & \mathbf{1} \mid [\uparrow^{Suc(n)}]\mathbf{1} \mid \Pi_K.A \mid \Pi_{A'}.A \mid \\ & & & \lambda_K.M \mid \lambda_A.M \mid (M\ M') \mid X \mid [S]X \end{array}$$

where M *and* S *are terms and substitutions in* $\mathcal{L}_{\Pi}$*-normal form, and* X *is a meta-variable in* Σ.

Proof. Firstly we show that the above terms are $\mathcal{L}_{\Pi}$-normal forms, and then we prove by induction on typing derivation that other cases are not possible. □

Remark 7.47 *For a given signature* Σ *and context* Γ, *it is possible to know if a well-typed term* M *is a kind or a constructeur.*

Definition 7.48 *For a signature* Σ, *the* set-interpretation *of a kind* K, *denoted by* $\mathcal{V}_{\Sigma}(K)$, *is defined inductively as follows:*

$$\begin{array}{lll} \mathcal{V}_{\Sigma}(\mathit{Type}) & = & \mathbf{SAT} \\ \mathcal{V}_{\Sigma}(x) & = & \mathbf{SAT} \quad \text{if } x = X \text{ or } x = [S]X, \text{ where } X \text{ is a meta-variable in } \Sigma \\ \mathcal{V}_{\Sigma}(\Pi_{K'}.M) & = & \{f \mid f \in \mathcal{V}_{\Sigma}(K') \rightarrow \mathcal{V}_{\Sigma}(M)\} \quad \text{if } K' \text{ is a kind} \\ \mathcal{V}_{\Sigma}(\Pi_N.M) & = & \mathcal{V}_{\Sigma}(M) \quad \text{if } N \text{ is not a kind} \end{array}$$

The collection of all set-interpretations is denoted by $\mathbf{SAT}^*$.
We assume that set-interpretations are defined in a global signature Σ.

Definition 7.49 *The* constructor valuations *of* Γ, *denoted by* $[\![\Gamma]\!]_\Box$, *is a collection of lists of sets in* $\mathbf{SAT}^*$, *defined inductively on* Γ *as follows:*

$$\begin{array}{lcl} [\![nil]\!]_\Box & = & \{nil\} \\ [\![\Gamma'.K]\!]_\Box & = & \{\xi.\alpha \mid \xi \in [\![\Gamma]\!]_\Box \wedge \alpha \in \mathcal{V}_\Sigma(K)\} \quad \textit{if } K \textit{ is a kind} \\ [\![\Gamma'.M]\!]_\Box & = & \{\xi.\mathcal{SN}_\mathcal{L} \mid \xi \in [\![\Gamma]\!]_\Box\} \quad \textit{if } M \textit{ is not a kind} \end{array}$$

Remark 7.50 *If* $\xi \in [\![\Gamma]\!]_\Box$, *then* ξ *and* Γ *has the same length.*

Definition 7.51 *For* ξ, *a constructor valuation of* Γ, *i.e.* $\xi \in [\![\Gamma]\!]_\Box$, *the* type interpretation function, *denoted by* $[\![.]\!]_\xi$, *is a map from constructors in* $\mathcal{L}_\Pi$*-normal form to sets in* $\mathbf{SAT}^*$. *It is defined inductively as follows:*

$$\begin{array}{lcl} [\![x]\!]_\xi & = & \mathcal{SN}_\mathcal{L} \quad \textit{if } x \in \{\textit{Kind}, \textit{Type}\} \\ [\![x]\!]_\xi & = & \mathcal{SN}_\mathcal{L} \quad \textit{if } x = X \textit{ or } x = [S]X, \textit{ where } X \textit{ is a meta-variable} \\ [\![(M\ A)]\!]_\xi & = & [\![M]\!]_\xi([\![A]\!]_\xi) \quad \textit{if } A \textit{ is a constructor} \\ [\![(M\ N)]\!]_\xi & = & [\![M]\!]_\xi \quad \textit{if } N \textit{ is not a constructor} \\ [\![\lambda_K.M]\!]_\xi & = & \alpha \in \mathcal{V}_\Sigma(K) \longmapsto [\![M]\!]_{\xi.\alpha} \quad \textit{if } K \textit{ is a kind} \\ [\![\lambda_A.M]\!]_\xi & = & [\![M]\!]_\xi \quad \textit{if } A \textit{ is a constructor} \\ [\![\Pi_K.N]\!]_\xi & = & [\![K]\!]_\xi \to \cap_{\alpha \in \mathcal{V}_\Sigma(K)} [\![N]\!]_{\xi.\alpha} \quad \textit{if } K \textit{ is a kind} \\ [\![\Pi_A.N]\!]_\xi & = & [\![A]\!]_\xi \to [\![N]\!]_\xi \quad \textit{if } A \textit{ is a constructor} \\ [\![\underline{1}]\!]_{\xi'.\alpha} & = & \alpha \quad \textit{if } \xi = \xi'.\alpha \\ [\![\underline{n+1}]\!]_{\xi'.\alpha} & = & [\![\underline{n}]\!]_{\xi'} \quad \textit{if } \xi = \xi'.\alpha \\ [\![\underline{1}]\!]_{nil} & = & \mathcal{SN}_\mathcal{L} \\ [\![\underline{n+1}]\!]_{nil} & = & \mathcal{SN}_\mathcal{L} \end{array}$$

Lemma 7.52 (Soundness for $[\![.]\!]_\xi$) *Let* ξ *be a constructor valuation of* Γ, *i.e.* $\xi \in [\![\Gamma]\!]_\Box$,

- *if* $\Sigma :\cdot \Gamma \vdash M : K$ *and* $\Sigma :\cdot \Gamma \vdash K : \textit{Kind}$, *then* $[\![M]\!]_\xi \in \mathcal{V}_\Sigma(K)$, *and*
- *if* $\Sigma :\cdot \Gamma \vdash K : \textit{Kind}$, *then* $[\![K]\!]_\xi \in \mathbf{SAT}$.

Proof. By simultaneous structural induction on M and K. □

Lemma 7.53 *Let* ξ *be a constructor valuation of* Γ, *i.e.* $\xi \in [\![\Gamma]\!]_\Box$, *if* $\Sigma :\cdot \Gamma \vdash M : N$, *then* $[\![N]\!]_\xi \subseteq \mathcal{SN}_\mathcal{L}$.

Proof. By sort soundness theorem (Theorem 6.12), we have three cases:

- $N = \textit{Kind}$. By Def. 7.51, $[\![\textit{Kind}]\!]_\xi = \mathcal{SN}_\mathcal{L}$.
- $\Sigma :\cdot \Gamma \vdash N : \textit{Type}$ and $\Sigma :\cdot \Gamma \vdash \textit{Type} : \textit{Kind}$. By Lemma 7.52, $[\![N]\!]_\xi \in \mathcal{V}_\Sigma(\textit{Type}) = \mathbf{SAT} \subseteq \mathcal{SN}_\mathcal{L}$.
- $\Sigma :\cdot \Gamma \vdash N : \textit{Kind}$. By Lemma 7.52, $[\![N]\!]_\xi \in \mathbf{SAT} \subseteq \mathcal{SN}_\mathcal{L}$.

□

Remark 7.54 *Type interpretations functions do not depend on type annotations in substitutions, thus if M and N are two $\mathcal{L}_\Pi$-normal terms such that $|M| = |M'|$, then $[\![M]\!]_\xi = [\![N]\!]_\xi$. For the sake of simplicity, we write $[\![[S]M]\!]_\xi$ to denote the interpretation function of elements in $([S]M)\downarrow_{\Pi_{\mathcal{L}}}$.*

Definition 7.55 *For ξ a constructor valuation of Γ, i.e. $\xi \in [\![\Gamma]\!]_\Box$, the* set $\mathrm{CC}_{\mathcal{L}}$-valuations *of Γ with respect to ξ, denoted by $[\![\Gamma]\!]_\xi$, is a set of substitutions in $\mathcal{NF}_{\mathcal{L}}$ defined inductively on Γ as follows:*

$$[\![nil]\!]_{nil} = \{\uparrow^n \mid \textit{for any natural } n\}$$
$$[\![\Gamma'.N']\!]_{\xi'.\alpha} = [\![nil]\!]_{nil} \cup \{S \cdot M_{:N} \in \mathcal{NF}_{\mathcal{L}} \mid \begin{array}{l} S \in [\![\Gamma']\!]_{\xi'}, \\ M \in [\![[\uparrow^{Suc(0)}]N']\!]_{\xi'.\alpha}, \\ N \in \mathcal{SN}_{\mathcal{L}}, \\ [\![N]\!]_{\xi'} = [\![N']\!]_{\xi'}\} \end{array}$$

Lemma 7.56 *For any Γ and $\xi \in [\![\Gamma]\!]_\Box$, $[\![\Gamma]\!]_\xi \subseteq \mathcal{SN}_{\mathcal{L}}$.*

Proof. Just as in the proof of Lemma 7.13 (and Lemma 7.36), we show by structural induction on S that if $S \in [\![\Gamma]\!]_\xi$, then $S \in \mathcal{SN}_{\mathcal{L}}$. □

Definition 7.57 *Let M, S be in $\mathcal{NF}_{\mathcal{L}}$, we define*

1. *Γ satisfies that M is of type N, denoted by $\Gamma \models M : N$, if and only if $([T]M)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N]\!]_\xi$ for any $\xi \in [\![\Gamma]\!]_\Box$ and $T \in [\![\Gamma]\!]_\xi$.*
2. *Γ satisfies that S is of type Δ, denoted by $\Gamma \models S \triangleright \Delta$, if and only if $(T \circ S)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Delta]\!]_\xi$ for any $\xi \in [\![\Gamma]\!]_\Box$ and $T \in [\![\Gamma]\!]_\xi$.*

7.2.4 Weak Normalization Proof for $\mathrm{CC}_{\mathcal{L}}$

First, we show some technical lemmas.

Lemma 7.58 *If $\xi \in [\![\Gamma]\!]_\Box$, $S \in [\![\Gamma]\!]_\xi$, $M \in [\![N]\!]_\xi$ and $N \in \mathcal{SN}_{\mathcal{L}}$, then $(S \cdot M_{:N})\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Gamma.N]\!]_\xi$.*

Proof. We proceed as in the proof of Lemma 7.38. □

Lemma 7.59 *Let ξ be in $[\![\Gamma]\!]_\Box$,*

1. *if $\Sigma :\cdot \Gamma \vdash M : N_1$ and $\Sigma :\cdot \Gamma \vdash M : N_2$, then $[\![N_1]\!]_\xi = [\![N_2]\!]_\xi$, and*
2. *if $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta_1$ and $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta_2$, then $[\![\Delta_1]\!]_\xi = [\![\Delta_2]\!]_\xi$.*

Proof. We verify that if M is a constructor, N an object and K a constructor or a kind in Γ, then $[\![[M' \cdot MS]K]\!]_\xi = [\![K]\!]_{\xi.[\![M]\!]_\xi}$ and $[\![[M' \cdot NS]K]\!]_\xi = [\![K]\!]_\xi$. And we proceed as in the proof of Lemma 7.40. Note that we use the fact that $\mathrm{CC}_{\mathcal{L}}$ has the typing soundness property (Theorem 6.22). □

The weak normalization proof is based on the following proposition.

Proposition 7.60 (Soundness of $\models$)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, *then* $\Gamma \models M : N$,
2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* $\Gamma \models S \triangleright \Delta$.

Proof. By simultaneous structural induction on M and S. We detail some cases, the other cases are solved as in Proposition 7.43.

- $M = \mathbf{1}$. Let ξ be in $[\![\Gamma]\!]_\Box$ and $T \in [\![\Gamma]\!]_\xi$, there are three cases:
  - $T = \uparrow^0$. Therefore, $([T]\mathbf{1})\downarrow_{\Pi_\mathcal{L}} = \{\mathbf{1}\}$. But $\mathbf{1}$ is a neutral $\beta\Pi_\mathcal{L}$-normal form, then by Corollary 7.26, $\mathbf{1} \in [\![N]\!]_\xi$.
  - $T = \uparrow^{Suc(n)}$. Therefore, $([T]\mathbf{1})\downarrow_{\Pi_\mathcal{L}} = \{[\uparrow^{Suc(n)}]\mathbf{1}\}$. But $[\uparrow^{Suc(n)}]\mathbf{1}$ is a neutral $\beta\Pi_\mathcal{L}$-normal form, then by Corollary 7.26, $[\uparrow^{Suc(n)}]\mathbf{1} \in [\![N]\!]_\xi$.
  - $T = S' \cdot M'_{:N_1}$. Therefore, $([T]\mathbf{1})\downarrow_{\Pi_\mathcal{L}} = \{M'\}$. By Def. 7.55, $\Gamma = \Gamma'.N_2$ and $M' \in [\![[\uparrow^{Suc(0)}]N_2]\!]_\xi$. From the typing rules we know that $\Sigma :\cdot \Gamma \vdash \mathbf{1} : [\uparrow^{Suc(0)}]N_2$. By Lemma 7.59, $[\![N]\!]_\xi = [\![[\uparrow^{Suc(0)}]N_2]\!]_\xi$. Hence, $M' \in [\![N]\!]_\xi$.
- $M = \lambda_{M_1}.M_2$ and $\Sigma :\cdot \Gamma \vdash M_1 : \mathit{Kind}$, by Lemma 6.4 we have $\Sigma :\cdot \Gamma.M_1 \vdash M_2 : N'$ and $\Sigma :\cdot \Gamma \vdash \lambda_{M_1}.M_2 : \Pi_{M_1}.N'$. Let ξ be in $[\![\Gamma]\!]_\Box$ and $T \in [\![\Gamma]\!]_\xi$, note that if $M' \in ([T](\lambda_{M_1}.M_2))\downarrow_{\Pi_\mathcal{L}}$, then $M' = \lambda_{M'_1}.M'_2$ where $M'_1 \in ([T]M_1)\downarrow_{\Pi_\mathcal{L}}$ and $M'_2 \in ([\Uparrow_{M_1}(T)]M_2)\downarrow_{\Pi_\mathcal{L}}$. By induction hypothesis, $M'_1 \in ([T]M_1)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![\mathit{Type}]\!]_= \mathcal{SN}_\mathcal{L}$. In particular, if $T = \uparrow^0$, alors $M'_1 \in \mathcal{SN}_\mathcal{L}$.

  Now, we can prove, just an in Proposition 7.43, that for any $\alpha \in \mathcal{V}_\Sigma(M_1)$ and for all $M'' \in [\![M_1]\!]_\xi$, $([\uparrow^0 \cdot M''_{:M'_1}]M'_2)\downarrow_{\Pi_\mathcal{L}} \in [\![N']\!]_{\xi.\alpha}$. And then we show that $[\![M']\!]_\xi \in [\![M_1]\!]_\xi \to \cap_{\alpha \in \mathcal{V}_\Sigma(M_1)} [\![N']\!]_{\xi.\alpha} = [\![\Pi_{M_1}.N']\!]_\xi = [\![N]\!]_\xi$.
- $M = (M_1\ M_2)$ where M_1 and M_2 are constructors. By Lemma 6.4 we have $\Sigma :\cdot \Gamma \vdash M_1 : \Pi_{N_1}.N_2$, $\Sigma :\cdot \Gamma \vdash M_2 : N_1$ and $\Sigma :\cdot \Gamma \vdash (M_1\ M_2) : [\uparrow^0 \cdot M_{2:N_1}]N_2$. Let ξ be in $[\![\Gamma]\!]_\Box$ and $T \in [\![\Gamma]\!]_\xi$, note that if $M' \in ([T](M_1\ M_2))\downarrow_{\Pi_\mathcal{L}}$, then $M' = (M'_1\ M'_2)$ where $M'_1 \in ([T]M_1)\downarrow_{\Pi_\mathcal{L}}$ and $M'_2 \in ([T]M_2)\downarrow_{\Pi_\mathcal{L}}$. By Lemma 7.59, $[\![N]\!]_\xi = [\![[\uparrow^0 \cdot M_{2:N_1}]N_2]\!]_\xi$. By induction hypothesis, $([T]M_1)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![\Pi_{N_1}.N_2]\!]_\xi = [\![N_1]\!]_\xi \to \cap_{\alpha \in \mathcal{V}_\Sigma(M_1)} [\![N_2]\!]_{\xi.\alpha}$, and $([T]M_2)\downarrow_{\Pi_\mathcal{L}} \subseteq [\![N_1]\!]_\xi$. Hence, $(M'_1\ M'_2) \in \cap_{\alpha \in \mathcal{V}_\Sigma(M_1)} [\![N_2]\!]_{\xi.\alpha}$. Furthermore, by Lemma 7.52, $[\![M_2]\!]_\xi \in \mathcal{V}_\Sigma(N_1)$, so $(M'_1\ M'_2) \in [\![N_2]\!]_{\xi.[\![M_2]\!]_\xi} = [\![N]\!]_\xi$.

□

Now, we show that $\beta\Pi_\mathcal{L}$ is strongly normalizing.

Lemma 7.61 *Let* M, S *be expressions in* $\mathcal{NF}_\mathcal{L}$,

1. *if* $\Sigma :\cdot \Gamma \vdash M : N$, *then* $M \in \mathcal{SN}_\mathcal{L}$, *and*
2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* $S \in \mathcal{SN}_\mathcal{L}$.

Proof. For any kind K we define the canonical set $c^K \in \mathcal{V}_\Sigma(K)$ as follows:

$$\begin{array}{lcl} c^{\mathit{Type}} & = & \mathcal{SN}_\mathcal{L} \\ c^x & = & \mathcal{SN}_\mathcal{L} \quad \text{if } x = X \text{ or } x = [S]X \\ c^{\Pi_K.M} & = & f \in \mathcal{V}_\Sigma(K) \longmapsto c^M \\ c^{\Pi_A.M} & = & c^M \end{array}$$

For any well-formed context Γ, we define the canonical constructor valuation $|\Gamma|_{\square}$ as follows:

$$\begin{array}{lcl} |nil|_{\square} & = & nil \\ |\Gamma'.K|_{\square} & = & |\Gamma'|_{\square}.c^K \\ |\Gamma'.A|_{\square} & = & |\Gamma'|_{\square}.\mathcal{SN}_{\mathcal{L}} \end{array}$$

We verify, by using Def. 7.49, that $|\Gamma|_{\square} \in [\![\Gamma]\!]_{\square}$. But also, by Def. 7.55, $\uparrow^0 \in [\![\Gamma]\!]_{|\Gamma|_{\square}}$. Hence,

1. By Proposition 7.60, $M \in ([\uparrow^0]M)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![N]\!]_{|\Gamma|_{\square}}$, and by Lemma 7.53(1), $[\![N]\!]_{|\Gamma|_{\square}} \subseteq \mathcal{SN}_{\mathcal{L}}$.
2. By Proposition 7.60, $S \in (\uparrow^0 \circ S)\downarrow_{\Pi_{\mathcal{L}}} \subseteq [\![\Delta]\!]_{|\Gamma|_{\square}}$, and by Lemma 7.56, $[\![\Delta]\!]_{|\Gamma|_{\square}} \subseteq \mathcal{SN}_{\mathcal{L}}$.

□

Finally, we prove weak normalization on typed $\lambda_{\mathcal{L}\Pi}$-expressions in $\mathrm{CC}_{\mathcal{L}}$.

Theorem 7.62 (Weak Normalization)

1. *If* $\Sigma :\cdot \Gamma \vdash M : N$, *then* M *is weakly normalizing, and*
2. *if* $\Sigma :\cdot \Gamma \vdash S \triangleright \Delta$, *then* S *is weakly normalizing.*

Therefore, M *and* S *have* $\lambda_{\mathcal{L}\Pi}$*-normal forms.*

Proof. By Proposition 3.11, there exists $M', S' \in \mathcal{NF}_{\mathcal{L}}$ such that $M \xrightarrow{\mathcal{L}_{\Pi}^*} M'$ and $S \xrightarrow{\mathcal{L}_{\Pi}^*} S'$. The subject reduction theorem (Theorem 6.20) says that typing is preserved under reductions, hence $\Sigma :\cdot \Gamma \vdash M' : N$ and $\Sigma :\cdot \Gamma \vdash S' \triangleright \Delta$. Therefore, by Lemma 7.61, M' and S' are both in $\mathcal{SN}_{\mathcal{L}}$. Finally, note that $\beta\Pi_{\mathcal{L}}$-normal forms in $\mathcal{NF}_{\mathcal{L}}$ are $\lambda_{\mathcal{L}\Pi}$-normal forms too. □

Chapter 8

Church-Rosser and Confluence

Church-Rosser property states that if two typed expressions are convertible, then they are joinable. The confluence property says that all the reductions of a typed expression are joinable.

We need the following lemma coined in [89].

Lemma 8.1 *Let x and y be $\lambda_{\mathcal{LP}}$-normal forms such that $x =_{\lambda_{\mathcal{LP}}} y$. Then, $x = y$ if*

- *x is a term, $\Sigma :\cdot \Gamma_1 \vdash x : N$ and $\Sigma :\cdot \Gamma_2 \vdash y : N'$, or*
- *x is a substitution, $\Sigma :\cdot \Gamma_1 \vdash x \triangleright \Delta_1$, $\Sigma :\cdot \Gamma_2 \vdash y \triangleright \Delta_2$ and $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_2$.*

Proof. By Corollary 5.9, $|x|$ and $|y|$ are $|\lambda_{\mathcal{LP}}|$-normal forms, and by Lemma 5.10(1), $|x| =_{|\lambda_{\mathcal{LP}}|} |y|$. Since $|\lambda_{\mathcal{LP}}|$ is confluent on semi-open expressions (Theorem 5.4), $|x| = |y|$. We proceed by structural induction on x. Remark that sub-terms of $\lambda_{\mathcal{LP}}$-normal forms are $\lambda_{\mathcal{LP}}$-normal forms too.

- $x = \mathit{Kind}$. This case is not possible by Remark 3.15.
- $x \in \{\mathbf{1}, \mathit{Type}\}$ or $x = \uparrow^n$ or x is a meta-variable. In these cases $x = y$, by definition of $|.|$.
- $x = \Pi_{M_1}.M_2$. By definition of $|.|$, $y = \Pi_{N_1}.N_2$ where $|M_1| = |N_1|$ and $|M_2| = |N_2|$. We have the hypotheses:
  1. $\Sigma :\cdot \Gamma_1.M_1 \vdash M_2 : s_1$, by Lemma 6.4 applied to the hypothesis of lemma.
  2. $\Sigma :\cdot \Gamma_1 \vdash M_1 : s_1'$, by Lemma 6.2(2) applied to (1).
  3. $\Sigma :\cdot \Gamma_2.N_1 \vdash N_2 : s_2$, by Lemma 6.4 applied to the hypothesis of lemma.
  4. $\Sigma :\cdot \Gamma_2 \vdash N_1 : s_2'$, by Lemma 6.2(2) applied to (3).
  5. (a) $M_1 =_{\lambda_{\mathcal{LP}}} N_1$ and (b) $M_2 =_{\lambda_{\mathcal{LP}}} N_2$, by Lemma 5.7 applied respectively to hypothesis $|M_1| = |N_1|$ and to hypothesis $|M_2| = |N_2|$.
  6. $M_1 = N_1$, by induction hypothesis applied to (2) (4) and (5-a).
  7. $M_2 = N_2$, by induction hypothesis applied to (1), (3) and (5-b).

  Therefore, $x = \Pi_{M_1}.M_2 = \Pi_{N_1}.N_2 = y$.
- $x = \lambda_{M_1}.M_2$. By definition of $|.|$, $y = \lambda_{N_1}.N_2$ where $|M_1| = |N_1|$ and $|M_2| = |N_2|$. We have the hypotheses:

1. $\Sigma :\cdot \Gamma_1.M_1 \vdash M_2 : N$, by Lemma 6.4 applied to the hypothesis of lemma.
2. $\Sigma :\cdot \Gamma_1 \vdash M_1 : s_1$, by Lemma 6.2(2) applied to (1).
3. $\Sigma :\cdot \Gamma_2.N_1 \vdash N_2 : N'$, by Lemma 6.4 applied to the hypothesis of lemma.
4. $\Sigma :\cdot \Gamma_2 \vdash N_1 : s_2$, by Lemma 6.2(2) applied to (3).
5. (a) $M_1 =_{\lambda_{\mathcal{L}\Pi}} N_1$ and (b) $M_2 =_{\lambda_{\mathcal{L}\Pi}} N_2$, by Lemma 5.7 applied respectively to hypothesis $|M_1| = |N_1|$ and to hypothesis $|M_2| = |N_2|$.
6. $M_1 = N_1$, by induction hypothesis applied to (2), (4) and (5-a).
7. $M_2 = N_2$, by induction hypothesis applied to (1), (3) and (5-b).

Therefore, $x = \lambda_{M_1}.M_2 = \lambda_{N_1}.N_2 = y$.

- $x = (M_1\ M_2)$. By definition of $|.|$, $y = (N_1\ N_2)$ where $|M_1| = |N_1|$ and $|M_2| = |N_2|$. We have the hypotheses:

  1. (a) $\Sigma :\cdot \Gamma_1 \vdash M_1 : \Pi_{M_1'}.N$, and (b) $\Sigma :\cdot \Gamma_1 \vdash M_2 : M_1'$, by Lemma 6.4 applied to the hypothesis of lemma.
  2. (a) $\Sigma :\cdot \Gamma_2 \vdash N_1 : \Pi_{N_1'}.N'$, and (b) $\Sigma :\cdot \Gamma_2 \vdash N_2 : N_1'$, by Lemma 6.4 applied to the hypothesis of lemma.
  3. (a) $M_1 =_{\lambda_{\mathcal{L}\Pi}} N_1$ and (b) $M_2 =_{\lambda_{\mathcal{L}\Pi}} N_2$, by Lemma 5.7 applied respectively to hypothesis $|M_1| = |N_1|$ and to hypothesis $|M_2| = |N_2|$.
  4. $M_1 = N_1$, by induction hypothesis applied to (1-a), (2-a) and (3-a).
  5. $M_2 = N_2$, by induction hypothesis applied to (1-b), (2-b) and (3-b).

  Therefore, $x = (M_1\ M_2) = (N_1\ N_2) = y$.

- $x = [S]M$. Since x is a $\lambda_{\mathcal{L}\Pi}$-normal form, we have two cases:

  - $M = \mathbf{1}$ and $S = \uparrow^{Suc(n)}$. In this case $x = y$, by definition of $|.|$.
  - $M = X$ (X is a meta-variable) and S is a $\lambda_{\mathcal{L}\Pi}$-normal substitution different to $\uparrow^0$. By definition of $|.|$, $y = [T]X$ where $|S| = |T|$. We have the hypotheses:
    1. (a) $\Sigma :\cdot \Gamma_1 \vdash S \triangleright \Delta_1$ and (b) $\Sigma :\cdot \Delta_1 \vdash X : N_1$, by Lemma 6.4 applied to the hypothesis of lemma.
    2. (a) $\Sigma :\cdot \Gamma_2 \vdash T \triangleright \Delta_2$ and (b) $\Sigma :\cdot \Delta_2 \vdash X : N_2$, by Lemma 6.4 applied to the hypothesis of lemma.
    3. $S =_{\lambda_{\mathcal{L}\Pi}} T$, by Lemma 5.7 applied to hypothesis $|S| = |T|$.
    4. $\Delta_1 =_{\lambda_{\mathcal{L}\Pi}} \Delta_2$, by Lemma 6.6 applied to (1-b) and (2-b).
    5. $S = T$, by induction hypothesis applied to (1-a), (2-a), (3) and (4).

    Therefore, $x = [S]X = [T]X = y$.

- $x = T \circ S$. It is not difficult to see that $T \circ S$ is not a $\lambda_{\mathcal{L}\Pi}$-normal form. Therefore this case does not apply.

- $x = S \cdot M_{1:M_2}$. By definition of $|.|$, $y = T \cdot N_{1:N_2}$ where $|S| = |T|$ and $|M_1| = |N_1|$. We have the hypotheses:

1. (a) $\Sigma :\cdot \Gamma_1 \vdash S \triangleright \Delta'_1$, (b) $\Sigma :\cdot \Gamma_1 \vdash M_1 : [S]M_2$, (c) $\Sigma :\cdot \Delta'_1 \vdash M_2 : s$ and (d) $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta'_1.M_2$, by Lemma 6.4 applied to the hypothesis of lemma.
2. (a) $\Sigma :\cdot \Gamma_2 \vdash T \triangleright \Delta'_2$, (b) $\Sigma :\cdot \Gamma_2 \vdash N_1 : [T]N_2$, (c) $\Sigma :\cdot \Delta'_2 \vdash N_2 : s$ and (d) $\Delta_2 =_{\lambda_{\mathcal{LP}}} \Delta'_2.N_2$, by Lemma 6.4 applied to the hypothesis of lemma.
3. (a) $S =_{\lambda_{\mathcal{LP}}} T$ and (b) $M_1 =_{\lambda_{\mathcal{LP}}} N_1$, by Lemma 5.7 applied respectively to hypothesis $|S| = |T|$ and to hypothesis $|M_1| = |N_1|$.
4. (a) $\Delta'_1 =_{\lambda_{\mathcal{LP}}} \Delta'_2$ and (b) $M_2 =_{\lambda_{\mathcal{LP}}} N_2$, by hypothesis $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_2$, (1-d) and (2-d).
5. $S = T$, by induction hypothesis applied to (1-a), (2-a), (3-a) and (4-a).
6. $M_1 = N_1$, by induction hypothesis applied to (1-b), (2-b) and (3-b).
7. $M_2 = N_2$, by induction hypothesis applied to (1-c), (2-c) and (4-b).

Therefore, $x = S \cdot M_{1:M_2} = T \cdot N_{1:N_2} = y$.

□

Theorem 8.2 (Church-Rosser) *Let x and y be such that $x =_{\lambda_{\mathcal{LP}}} y$. Then, x and y are $\lambda_{\mathcal{LP}}$-joinable, i.e. there exists w such that $x \xrightarrow{\lambda_{\mathcal{LP}}^*} w$ and $y \xrightarrow{\lambda_{\mathcal{LP}}^*} w$, if*

1. *x is a term, $\Sigma :\cdot \Gamma_1 \vdash x : N_1$ and $\Sigma :\cdot \Gamma_2 \vdash y : N_2$, or*
2. *x is a substitution, $\Sigma :\cdot \Gamma_1 \vdash x \triangleright \Delta_1$, $\Sigma :\cdot \Gamma_2 \vdash y \triangleright \Delta_2$ and $\Delta_1 =_{\lambda_{\mathcal{LP}}} \Delta_2$.*

Proof. By weak normalization theorem (Theorem 7.62), there exists $\lambda_{\mathcal{LP}}$-normal forms x' and y' such that $x \xrightarrow{\lambda_{\mathcal{LP}}^*} x'$ and $y \xrightarrow{\lambda_{\mathcal{LP}}^*} y'$. It is sufficient to show that $x' = y'$, which is a consequence of subject reduction theorem (Theorem 6.20) and Lemma 8.1. □

The above property is not valid when $\Delta_1 \neq_{\lambda_{\mathcal{LP}}} \Delta_2$. Take the context

$$\Gamma = \mathit{nil}.\ \mathit{nat}{:}\mathit{Type}.\ T{:}\mathit{nat}{\rightarrow}\mathit{Type}.\ l{:}(\Pi n{:}\mathit{nat}.(T\ n)).\ 0{:}\mathit{nat}.\ m{:}(T\ 0){\rightarrow}\mathit{nat}$$

and the two substitutions

$$S_1 = x := 0_{:\mathit{nat}} \cdot y := (l\ 0)_{:(T\ x)}$$

$$S_2 = x := 0_{:\mathit{nat}} \cdot y := (l\ 0)_{:(T\ 0)}$$

We have, by Lemma 5.7, $S_1 =_{\lambda_{\mathcal{LP}}} S_2$. But also,

$$\Gamma \vdash S_1 \triangleright \Gamma.\ x{:}\mathit{nat}.\ y{:}(T\ x)$$

and

$$\Gamma \vdash S_2 \triangleright \Gamma.\ x{:}\mathit{nat}.\ y{:}(T\ 0)$$

However, S_1 and S_2 are not $\lambda_{\mathcal{LP}}$-joinable.

Confluence is a consequence of Church-Rosser and subject reduction properties.

Corollary 8.3 (Confluence) *Let x be an arbitrary well-typed expression, and y and z be such that $x \xrightarrow{\lambda_{\mathcal{LP}}^*} y$ and $x \xrightarrow{\lambda_{\mathcal{LP}}^*} z$, then there exists w such that $y \xrightarrow{\lambda_{\mathcal{LP}}^*} w$ and $z \xrightarrow{\lambda_{\mathcal{LP}}^*} w$.*

Since well-typed terms enjoys both Church-Rosser and weak normalization, we have that $\lambda_{\mathcal{L}\Pi}$-normal forms of well-typed terms always exist and they are unique. Thus, conversion on well-typed expressions is decidable.

Corollary 8.4 (Decidability) *The equivalence $x =_{\lambda_{\mathcal{L}\Pi}} y$ is decidable if*

- *x is a term, $\Sigma :\cdot \Gamma_1 \vdash x : N_1$ and $\Sigma :\cdot \Gamma_2 \vdash y : N_2$, or*
- *x is a substitution, $\Sigma :\cdot \Gamma_1 \vdash x \triangleright \Delta_1$, $\Sigma :\cdot \Gamma_2 \vdash y \triangleright \Delta_2$ and $\Delta_1 =_{\lambda_{\mathcal{L}\Pi}} \Delta_2$.*

Chapter 9

Conclusion

Before summarizing the main contributions of this thesis, we discuss some works about meta-variables and proof construction issues based on other formalisms.

9.1 Related Works

Syntactic manipulations of expressions with binding operators are a frequent source of problems. Several sorts of variables and substitutions (each one with different scoping behavior) are difficult to conciliate with other theoretical requirements.

The Huet's higher-order unification algorithm [46] uses a technique to code unification variables by means of the variables of λ-calculus. This idea was generalized by Pfenning and Elliott [78] in their Higher-Order Abstract Syntax notation. In this approach, general binding operators are expressed in an underlying λ-calculus. Two systems that use this higher-order notation are λProlog [65] and Isabelle [77]. Felty and Howe [29] present a proof system written in λProlog that supports the construction and manipulation of tree-structured proofs that can contain meta-variables. Basin, Bundy, Kraan and Matthews [5] show that the activities of program verification and program synthesis can be unified if meta-variables are admitted into proofs. That work is illustrated with an example developed in the Isabelle system. In both cases, the meta-variables are inherited from the implementation languages.

Talcott [87] presents a theoretical approach to holes in binding structures. In opposition to the higher-order abstract syntax tradition, this work emphasizes a first-order, structural view of syntactic entities of the theory. In the same direction, Hashimoto and Ohori propose in [42] a typed λ-calculus with holes where the filling mechanism is explicit.

A method of proof construction which uses meta-variables to postpone some choices in natural deduction proofs is proposed by Clement [13]. For example, in the logical rule

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\,(\mathrm{Elim}^{\vee})$$

propositions A and B appear in the hypotheses but not in the conclusion. As they will be known a posteriori, we can consider them as meta-variables. Notice that Clement's meta-variables represent predicate symbols, not place-holders of incomplete proofs in the sense that we have used here.

Meta-variables are used also to develop programs by refinement steps. In particular, a refinement calculus that uses meta-variables and conditional refinements to facilitate goal-directed development of programs is described by Nickson and Groves [75].

Burstall [12] explains the construction of terms via interactive refinement of proofs. In his work, there is no an explicit usage of places-holders.

Recently, Ritter, Pym and Wallen [83] have prosed the $\lambda\mu\epsilon$-calculus —a variant of the Parigot's $\lambda\mu$-calculus with explicit substitutions— as a realizer of classical and intuitionistic logics. Proof construction issues are not considered there.

9.2 Summary and Perspectives

In this thesis we have developed a formal framework to represent proof-terms with place-holders, for dependent type systems, where the proof construction process can be seen as a procedure for term construction. In this way, the "propositions-as-types" principle is extended to an "incomplete-proofs as incomplete-terms" principle, or better a "proof-construction as term-construction" principle.

We propose a typed λ-calculus with meta-variables playing the role of holes. Thus, incomplete proofs are just λ-terms with meta-variables, and proof refinement steps correspond to instantiations of meta-variables. In order to guarantee the correction of the construction, the substitution operation and the renaming mechanism of λ-calculus must be made explicit in the theory.

Calculi where the substitution operation is part of the formal language, are known as calculi of explicit substitutions. There are several versions of λ-calculi with explicit substitutions, but unfortunately, most of them do not share the same properties as λ-calculus.

For a couple of years, the incompatibility between confluence (on open terms, i.e. on terms with meta-variables) and preservation of strong normalization of β-reduction, was conjectured for the calculi of explicit substitutions. In this thesis, we solve the conjecture by proposing the λ_ζ-calculus which enjoys both properties of confluence and preservation of strong normalization at the same time.

However, the λ_ζ-calculus lacks for composition operator. This operator was introduced in some calculi in order to gain local confluence. But also, it happens to be useful for other purposes, for instance the pruning of search space in unification algorithms. Actually, we do not use the λ_ζ-calculus to represent incomplete proof-terms. Proof synthesis methods for higher-order logics are based on unification where composition and simultaneous substitutions are very useful features. However, we think that λ_ζ is a first step toward a calculus of practical interest.

On the other hand, the $\lambda\sigma$-calculus is confluent on expressions with meta-variables of terms and it is weakly normalizing on typed expressions. These two properties are sufficient to decide the equivalence of well-typed terms with meta-variables. In addition, the $\lambda\sigma$-calculus uses general composition of substitutions and simultaneous substitutions. However, $\lambda\sigma$ is not left-linear, and this feature raises technical problems in some frameworks. We propose a left-linear variant of $\lambda\sigma$, namely $\lambda_\mathcal{L}$, which enjoys the same general properties as $\lambda\sigma$.

We have studied the meta-theoretical properties of the typed versions of the $\lambda_\mathcal{L}$-calculus. In particular, the system $\mathrm{CC}_\mathcal{L}$ is a formulation of $\lambda_\mathcal{L}$ for the calculus of constructions. The system $\mathrm{CC}_\mathcal{L}$ enjoys the usual typing properties: Type Uniqueness, Subject Reduction, Confluence and Weak Normalization, all of them on expressions with meta-variables of terms.

In $\mathrm{CC}_\mathcal{L}$, meta-variables and variables are declared in different list structures. We use signatures to keep track of meta-variable declarations, and contexts to keep track of variable declarations.

Each meta-variable in a signature is assigned with a unique type and a unique context (modulo conversions), and in this way we can guarantee the soundness of instantiation of meta-variables.

The meta-theoretical study of the properties of $\mathrm{CC}_{\mathcal{L}}$ mixes several techniques. For instance, the proof of confluence uses the Yokouchi-Hikita's lemmas (on abstract relations), the General Critical Pair lemma (on rewrite systems), and the Geuvers' lemma (on typed λ-calculus).

Finally, we present a method of proof synthesis for $\mathrm{CC}_{\mathcal{L}}$. This method merges a procedure of term enumeration, with a technique of higher-order unification via explicit substitutions where unification variables are encoded by meta-variables.

Our complete formalism is developed for the Calculus of Constructions, the Dependent Type theory and the Simple Type Theory, but the same ideas can be applied to other formal systems. In particular, work is in progress to extend the system $\mathrm{CC}_{\mathcal{L}}$ with inductive and co-inductive types.

Another interesting issue to be further studied, is the formalization, in a constructive framework, of the meta-theoretical development presented here. In fact, using the same proof-as-term paradigm in a "boot-strapping" way, we can imagine the automatic extraction of algorithms for type checking, and also the automatic generation of proof construction tools based on our method of proof synthesis.

Bibliography

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitution. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] T. Altenkirch, V. Gaspes, B. Nordström, and B. von Sydow. *A User's Guide to ALF*. Chalmers University of Technology, Sweden, May 1994.

[3] H. Barendregt and K. Hemerik. Types in lambda calculi and programming languages. In N. D. Jones, editor, *Proceedings of the Third European Symposium on Programming*, volume 432 of *LNCS*. Springer-Verlag, 1990.

[4] H. P. Barendregt. *The Lambda Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B.V. (North-Holland), 1984.

[5] D. A. Basin, A. Bundy, I. Kraan, and S. Matthews. A framework for program development based on schematic proof. In *7th Intern. Workshop on Software Specification and Design* (Redondo Beach, CA), pages 162–171, Los Alamitos, CA, 1993. IEEE Computer Society Press. Also available as Technical Report MPI-I-93-231.

[6] Z.-E.-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a calculus of explicit substitutions which preserves strong normalisation. Technical Report RR-2477, Unité de recherche INRIA-Lorraine, Janvier 1995.

[7] R. Bloo. Preservation of strong normalisation for explicit substitution. Technical Report 95-08, Eindhoven University of Technology, Department of Computing Science, March 1995.

[8] R. Bloo. *Preservation of Termination for Explicit Substitution*. PhD thesis, Eindhoven University of Technology, 1997.

[9] R. Bloo, F. Kamareddine, and R. Nederpelt. The Barendregt cube with definitions and generalised reduction. *Information and Computation*, 126(2):123–143, 1 May 1996.

[10] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN-95: Computer Science in the Netherlands*, November 1995.

[11] D. Briaud. Higher order unification as a typed narrowing. Technical report, CRIN report 96-R-112, 1996.

[12] R. Burstall. Terms, proofs, and refinement (extended abstract). In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 2–7, Paris, France, 4–7 July 1994. IEEE Computer Society Press.

[13] T. Clement. Using metavariables in natural deduction proofs. In *Proceedings of the 5th Refinement Workshop*, pages 255–271. Springer-Verlag, 1992.

[14] T. Coquand. *Une Théorie de Constructions.* Thèse de doctorat, U. Paris VII, 1985.

[15] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76:96–120, 1988.

[16] C. Cornes. *Conception d'un Langage de Haut Niveau de Représentation de Preuves: récurrence par filtrage de motifs, unification en présence de types inductif primitifs. synthèse de lemmes d'inversion.* Thèse de doctorat, Université Paris 7, 1997.

[17] C. Cornes, J. Courant, J.-C. Filliâtre, G. Huet, P. Manoury, C. Paulin-Mohring, C. Muñoz, C. Murthy, C. Parent, A. Saïbi, and B. Werner. The Coq Proof Assistant Reference Manual Version 5.10. Technical Report RT-0177, INRIA, July 1995.

[18] R. Di Cosmo and D. Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97) Warsaw, Poland*, July 1997.

[19] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Pitman, second edition, 1993. Available from John Wiley and Sons.

[20] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, March 1996.

[21] N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34(5), 1972.

[22] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1-2):69–115, February-April 1987.

[23] G. Dowek. *Démonstration automatique dans le calcul des constructions.* Thèse de doctorat, U. Paris VII, 1991.

[24] G. Dowek. A complete proof synthesis method for type systems of the cube. *Journal of Logic and Computation*, 3(3):287–315, June 1993.

[25] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions (extended abstract). In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 366–374, San Diego, California, 26–29 June 1995. IEEE Computer Society Press.

[26] G. Dowek, T. Hardin, C. Kirchner, and F. Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, Bonn, Germany, September 1996. MIT Press.

[27] T. Ehrhard. *Une sémantique Catégorique des Types Dépendants. Application au Calcul des Constructions.* Thèse de doctorat, U. Paris VII, 1988.

[28] C. Elliott. Higher-order unification with dependent types. In N. Dershowitz, editor, *Rewriting Techniques and Applications*, volume 355 of *LNCS*, pages 121–136, Chapel Hill, North Carolina, April 1989. Springer-Verlag.

[29] A. Felty and D. Howe. Tactic theorem proving with refinement-tree proofs and metavariables. In Alan Bundy, editor, *Proceedings, 12th International Conference on Automated Deduction*, volume 596 of *LNAI*, pages 605–619, Nancy, France, June 1994. Springer-Verlag.

[30] M. C. F. Ferreira, D. Kesner, and L. Puel. λ-calculi with explicit substitutions and composition which preserve β-strong normalization. *LNCS*, 1139, 1996.

[31] Gerhard Gentzen. Investigations into logical deductions, 1935. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland Publishing Co., Amsterdam, 1969.

[32] H. Geuvers. The Church-Rosser property for $\beta\eta$-reduction in typed λ-calculi. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 453–460, Santa Cruz, California, 22–25 June 1992. IEEE Computer Society Press.

[33] H. Geuvers. A short and flexible proof of Strong Normalization for the Calculus of Constructions. In P. Dybjer and B. Nordström, editors, *Types for Proofs and Programs, International Workshop TYPES'94*, volume 996 of *LNCS*, pages 14–38, Båstad, Sweden, 1994. Springer.

[34] H. Geuvers and R. Bloo. Counter-example for subject reduction in calculi of explicit substitutions with dependent types. Personal communication, 1997.

[35] H. Geuvers and B. Werner. On the Church-Rosser property for expressive type systems and its consequences for their metatheoretic study. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 320–329, Paris, France, 4–7 July 1994. IEEE Computer Society Press.

[36] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proof and Types.* Cambridge University Press, 1989.

[37] J. Goubault-Larrecq. A proof of weak termination of typed $\lambda\sigma$-calculi. Manuscript, 1997.

[38] J. Goubault-Larrecq. Une preuve de terminaison faible du $\lambda\sigma$-calcul. Technical Report RR-3090, Unité de recherche INRIA-Rocquencourt, Janvier 1997.

[39] T. Hardin and A. Laville. Proof of termination of the rewriting system subst on CCL. *Theoretical Computer Science*, 46:291–342, 1986.

[40] T. Hardin, L. Maranget, and B. Pagano. Functional back-ends and compilers within the lambda-sigma calculus. In Thomas Johnsson, editor, *The Workshop on the Implementation of Functional Languages '95*. Bastad, Sweden, September 1995.

[41] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[42] M. Hashimoto and A. Ohori. A typed context calculus. Technical Report RIMS-1098, Research Institute for Mathematical Sciences, Kyoto University, 1996.

[43] H. Herbelin. *Séquents qu'on calcule, de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes.* Thèse de doctorat, U. Paris VII, 1994.

[44] H. Herbelin. Notes sur les $\lambda\sigma$-calculs. Preprint, 1995.

[45] G. Huet. *Constrained Resolution A Complete Method for Higher Order Logic.* PhD thesis, Case Western Reserve University, 1972.

[46] G. Huet. A unification algorithm for typed lambda calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.

[47] G. Huet. *Résolution d'Équations dans les Langages d'Ordre 1,2, ...,ω.* Thèse de doctorat, U. Paris VII, 1976.

[48] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J.A.C.M.*, 27(4), October 1980.

[49] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In *PLILP*. LNCS, 1995.

[50] F. Kamareddine and A. Ríos. The λ_J-calculus: its typed and its extended versions. Manuscript, June 1995.

[51] D. Kapur, P. Narendran, and F. Otto. On ground-confluence of term rewriting systems. *Information and Computation*, 86(1):14–31, May 1990.

[52] D. Kapur and H. Zhang. RRL: A rewrite rule laboratory-user's manual. Technical Report 89-03, Department of Computer Science, The University of Iowa, 1989.

[53] D. Kesner. Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions (extended abstract). In Harald Ganzinger, editor, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA-96)*, volume 1103 of *LNCS*, pages 184–199, New Brunswick, NJ, USA, 1996. Springer-Verlag.

[54] D. Kesner. Confluence of extensional and non-extensional λ-calculi with explicit substitutions. Preprint, 1997.

[55] C. Kirchner and C. Ringeissen. Higher order equational unification via explicit substitutions. In *Proceedings International Conference PLILP/ALP/HOA'97*, volume 1298 of *LNCS*, Southampton (England), September 1997. Springer.

[56] J.-W. Klop. Combinatory reduction systems. *Mathematical Center Tracts*, (27), 1980.

[57] J.-L. Krivine. *λ calcul, types et modèles.* Masson, 1990.

[58] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$ a journey through calculi of explicit substitutions. In *Proceedings of the 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 60–69, January 1994.

[59] P. Lescanne and J. Rouyer-Degli. The calculus of explicit substitutions $\lambda\upsilon$. Technical report, Centre de Recherche en Informatique de Nancy (CNRS) and INRIA-Lorraine, March 1994.

[60] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de Bruijn's levels. In J. Hsiang, editor, *Rewriting Techniques and Applications*, volume 914 of *LNCS*, pages 294–308, Chapel Hill, North Carolina, 1995. Springer-Verlag.

[61] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*, volume 11 of *International Series of Monographs on Computer Science*. Oxford Science Publications, 1994.

[62] L. Magnusson. *The Implementation of ALF—A Proof Editor Based on Martin-Löf's Monomorphic Type Theory with Explicit Substitution.* PhD thesis, Chalmers University of Technology and Göteborg University, January 1995.

[63] P.-A. Melliès. Exemple de non terminaison forte dans un $\lambda\sigma$-calcul typé où la priorité serait donnée aux deux règles *shiftcons* et *varcons*, modulo lois de monoïde. Preprint, 1995.

[64] P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In *Typed Lambda Calculi and Applications*, volume 902 of *LNCS*. Second International Conference TLCA'95, Springer-Verlag, 1995.

[65] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[66] C. Muñoz. Confluence and preservation of strong normalisation in an explicit substitutions calculus. Technical Report RR-2762, Unité de recherche INRIA-Rocquencourt, Decembre 1995.

[67] C. Muñoz. Confluence and preservation of strong normalisation in an explicit substitutions calculus (extended abstract). In *Proceedings, Eleven Annual IEEE Symposium on Logic in Computer Science*, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

[68] C. Muñoz. Proof representation in type theory: State of the art. In *Proceedings, XXII Latinamerican Conference of Informatics CLEI Panel 96*, Santafé de Bogotá, Colombia, June 1996.

[69] C. Muñoz. Dependent types with explicit substitutions: A meta-theoretical development. In *Types for Proofs and Programs, Proceedings of International Workshop TYPES'96*, 1997.

[70] C. Muñoz. A left-linear variant of $\lambda\sigma$. In *Proceedings International Conference PLILP/ALP/HOA'97*, volume 1298 of *LNCS*, Southampton (England), September 1997. Springer.

[71] C. Muñoz. Meta-theoretical properties of λ_ϕ: A left-linear variant of $\lambda\sigma$. Technical Report RR-3107, Unité de recherche INRIA-Rocquencourt, Février 1997.

[72] C. Murthy. A new existential engine. Preprint, 1993.

[73] G. Nadathur. A fine-grained notation for lambda terms and its use in intensional operations. Technical Report TR-96-13, Department of Computer Science, University of Chicago, May 30 1996. Accepted for publication in Journal of Functional and Logic Programming.

[74] G. Nadathur. The (SCons) rule. Personal communication, 1996.

[75] R.G. Nickson and L.J. Groves. Metavariables and conditional refinements in the refinement calculus. In D. Till and R. G. F. Shaw, editors, *Proceedings of the 6th Refinement Workshop*. London, Springer-Verlag, 5–7 January 1994.

[76] B. Pagano. Confluent extensions of $\lambda_\Uparrow$. Personal communication, 1996.

[77] L.C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[78] F. Pfenning and C. 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.

[79] R. Pollak. On extensibility of proof checkers. In P. Dybjer, B. Nordström, and J. Smith, editors, *Proceedings of the International Workshop on Types for Proofs and Programs*, volume 996 of *LNCS*, pages 140–161, Båstad, Sweden, June 1994. Springer-Verlag.

[80] D. Pym. A unification algorithm for the $\lambda\Pi$-calculus. *International Journal of Foundations of Computer Science*, 3(3):333–378, September 1992.

[81] A. Ríos. *Contributions à l'étude de λ-calculs avec des substitutions explicites.* Thèse de doctorat, U. Paris VII, 1993.

[82] E. Ritter. Categorical abstract machines for higher-order lambda calculi. *Theoretical Computer Science*, 136(1):125–162, 1994.

[83] E. Ritter, D. Pym, and L. Wallen. Proof-terms for classical and intuitionistic resolution (extended abstract). In M.McRobbie and J.Slaney, editors, *Proceedings of CADE-13*, volume 1104 of *LNCS*, Springer, June 1996.

[84] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[85] M. Schmidt-Schauss. *Computational aspects of an order-sorted logic with term declarations*, volume 395 of *LNCS and LNAI*. Springer-Verlag, New York, NY, USA, 1989.

[86] P. Severi. *Normalisation in LAMBDA CALCULUS and its relation to type inference.* PhD thesis, Eindhoven University of Technology, 1996.

[87] C.L. Talcott. A theory of binding structures and applications to rewriting. *Theoretical Computer Science*, 112, 1993.

[88] A. Tasistro. Formulation of Martin-Löf's theory of types with explicit substitutions. Technical report, Chalmers University of Technology, University of Göteborg, Göteborg, Sweden, May 1993.

[89] B. Werner. *Une Théorie des Constructions Inductives.* Thèse de doctorat, U. Paris VII, 1994.

[90] H. Yokouchi and T. Hikita. A rewriting system for categorical combinators with multiple arguments. *SIAM Journal on Computing*, 19(1):78–97, February 1990.

[91] H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17(1):23–50, January 1994.

[92] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.

[93] H. Zantema. Termination of ϕ and Π_ϕ by semantic labelling. Personal communication, 1996.

Index

$\lambda_{\mathcal{L}\Pi}$-calculus 62
  Church-Rosser 141
  confluence 141
  decidability 142
  instantiation soundness 68
  inversion 105
  sort soundness 109
  subject reduction 117
  type uniqueness 111
  typing soundness 118
  weak normalization for $\mathrm{CC}_{\mathcal{L}}$ 137
  weak normalization for $\lambda\Pi_{\mathcal{L}}$ 132
$\lambda_{\mathcal{L}}$-calculus 43
  confluence 47
  in pre-expressions 62
  instantiation soundness 54
  inversion 55
  simply-typed version 52
  subject reduction 56
  type uniqueness 55
  weak normalization 124
$\lambda\upsilon$-calculus 29
  preservation of strong normalization . 31
$\lambda\sigma_{\Uparrow}$-calculus 29
  confluence 29
λ_{ζ}-calculus 35
  confluence 40
  preservation of strong normalization . 41
  reduction completeness 39
  reduction soundness 38
$\mathcal{L}_{\Pi}$-calculus 62
β-conversion 11
β-reduction
  big-step semantics 36
  one-step semantics 36
λ_{dB}-calculus 26
  β-reduction 27
λ-calculus 11
$\lambda\sigma$-calculus 27
`let-in` 70

Bachmair-Dershowitz
  commutation 32
  lemma 32

calculus of constructions 57
  $\mathrm{CC}_{\mathcal{L}}$-system 66
commutation
  instantiation and β-reduction 18
  typing and refinements 19
confluence 11
constraint 73
context 10
contexts 52
critical pair lemma 33
  generalized 33
  in sort-compatible systems 33
  on ground expressions 33
Curry-Howard isomorphism 9, 10
cut
  elimination theorem 12
  logic rule 12

dependent types 12
  $\lambda\Pi_{\mathcal{L}}$-system 65
  $\lambda\Pi$-system 57
domain 125

explicit substitutions 18, 25
expression
  pre-expression 62
  semi-open 28

Geuvers' lemma 102
grafting 76
  application of 76

composition of 80
elementary 81
elementary (in $\lambda\Pi_{\mathcal{L}}$) 90
sequential 77
valid 76

Hauptsatz 12
Heyting-Kolmogorov semantic 9

incomplete proof 16
instantiation 18
parallel 79

judgment
in logic 9
in type systems 10

logic
Classical 9
Intuitionistic 9

marks 36
meta-variables 18
declaration of 59

place-holder 16
preservation of strong normalization 28
product 12
proof-trees 16
propositions-as-types principle 10

saturated set
in $\lambda_{\mathcal{L}\Pi}$ 125
in $\lambda_{\mathcal{L}}$ 120
search tree 84
signature 59
constrained 73
derivation 79
failure 75
normal 74
restriction 79
search tree of 84
solution of 79
solvable 79
sort 13, 56
soundness of a type system 117
subject reduction 11
substitution 11
composition 27
cons 27
identity 27
lift 29
shift 27

tactic 16
tautology 9
term
λ-term 10
à la de Bruijn 26
abstraction 10
application 10
approximatively equal 88
atomic rigid 75
flexible 75
ground 28, 75
neutral 120
open 18
pure 36
pure in $\lambda_{\mathcal{L}\Pi}$ 68
pure in $\lambda_{\mathcal{L}}$ 54
purifiable 36
rigid 75
semi-open 28
well-typed 10
termination 11
type 10
inhabited 10
product 57

valuation
in $\mathrm{CC}_{\mathcal{L}}$ 135
in $\lambda\Pi_{\mathcal{L}}$ 128
in $\lambda_{\mathcal{L}}$ 122
of constructors in $\mathrm{CC}_{\mathcal{L}}$ 134
variable 10
variable declaration 10

Yokouchi-Hikita
commutation 31
lemma 32

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105,
78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS
Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399