université
PARIS-SACLAY

Higher-Order Confluence and Universe Embedding in the Logical Framework

Confluence d'ordre supérieur et encodage d'univers dans le Logical Framework

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580, sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique
Unité de recherche: Université Paris-Saclay, Inria, Inria Saclay-Île-de-France, 91120, Palaiseau, France
Référent: ENS Paris-Saclay

Thèse présentée et soutenue à ....., le .... 202X, par

Gaspard FÉREY

Composition du Jury

Maribel Fernández Professeure à King's College London, Londres	Rapporteure
Nicolas Tabareau Directeur de Recherche à Inria, Nantes	Rapporteur
Véronique Benzaken Professeure à Université Paris Saclay	Examinatrice
Femke van Raamsdonk Professeure associée à Vrije Universiteit, Amsterdam	Examinatrice
Tobias Nipkow Professeur à Université Technique de Munich	Examinateur
Jean-Pierre Jouannaud Professeur émérite à Université Paris Saclay	Invité encadrant

Direction de la thèse

Gilles Dowek Directeur de Recherche à Inria, Professeur attaché à ENS Paris-Saclay	Directeur

Thèse de doctorat

NNT: 2021UPASG032

Contents

Introduction

Chapter 1

Introduction

1.1 Formal proof systems

Mathematics have long been a matter of pen, paper, and some form of perseverance many would call stubbornness. The technological progress of the past decades has extended this toolbox with a new powerful tool: computers. *Proof assistants* are a wide variety of programs allowing to write theorems and develop their proofs using dedicated languages designed to be easily interpreted by computers. Proving a theorem using such a language is quite similar to writing a computer program: it is time consuming, requires to be very specific and precise, and, all too often, involves long desperate stares at alarming error messages. Yet proof systems offer many advantages. They usually come with the practical conveniences of modern development environments, such as syntax and debugging tools. More importantly the digital format of proof allows them to be easily shared and published, therefore promoting large scale research collaboration through the development of extensive libraries of proofs. Besides the recent drastic increase in available computing power has now allowed computers to directly assist mathematicians with the process of proof construction. Many tools, such as SAT or SMT solvers, are able to automatically iterate over numerous cases or to quickly try out several relatively straightforward strategies to handle particularly tedious roadblocks in proofs. Finally algorithms allow to *check* the developed proofs in real time and pinpoint errors in the logical reasoning. This considerably reduces the risk of writing an incorrect proof and therefore improves the trust that can be put in these formally verified theorems.

1.1.1 Practical use

Formal proof systems have met quite some success both in the fields of formalization of mathematics and certification of programs.

A famous example is the proof of the *four color theorem*, stating that the regions of any planar map can be colored with only four colors so that no two adjacent regions are assigned the same color. The original proof of this theorem involved heavy calculations that could not be directly checked and required instead to rely on a computer. The

mathematical community was therefore reluctant to recognize the validity of this proof until it was formalized by Gonthier and Werner [Gon05, Gon08] using the Coq proof assistant. Along with the other results that formal methods helped to prove, such as the Feit-Thompson theorem [GAA+13] and Kepler's conjecture [Hal05], it is worth mentioning the many efforts to formalize simpler but larger libraries of well-known textbook results and proof strategies.

Proof systems are also used, together with other formal methods such as model checking and abstract interpretation, to formally verify that programs are bug-free and correctly implement the algorithm they are supposed to. This is particularly important in the case of software controlling safety-critical systems which malfunction can potentially have great economical or even human consequences. While test-driven development prevents *most* simple bugs, formal software verification offers the much safer guarantee that *all* possible corner cases have been checked. An example of such verified software is the CompCert compiler for a subset of the C language. This program was proven, using Coq, to preserve the semantic of C programs when translating them into executable code in the assembly language. Other notable examples include the automation of Paris Métro line 14 using the B method or the verification of aeronautical systems by NASA using PVS and many more in the fields of hardware and security.

1.1.2 Type theoretical foundations of mathematics

Before developing a formal proof system, it is necessary to define the underlying *logic*: the mathematical objects and rules upon which all subsequent theoretical developments will be based. Set theory has long been universally accepted as such a foundation of mathematics. However, in the turn of the last century, several paradoxes have led to rethink the proper ways to define mathematics. These paradoxes usually rely on self-referencing issues such as the well-known paradox "This sentence is a lie." paradox. This grammatically correct sentence states its own negation and therefore cannot be assigned a truth value. Other paradoxes allow to contradict the existence of the ordinal number of all ordinals (Buralli Forti, 1897), the least indefinable ordinal (Richard, 1905), or the set of all set (Russell, 1901).

A work-around these paradoxes was suggested by Russell [Rus08] who introduced an alternative foundation: type theories. The objects of a type theory are called *terms* and are assigned a *type* which characterizes their *nature* in a somewhat more abstract and fundamental way than the sets they belong to or the properties they satisfy. For instance the natural number 0 can be assigned the type of natural number $\mathbb{N}$ which we write $0 : \mathbb{N}$. In many type theories, terms are built from variants of Church's λ-calculus [Chu40]. Functions may be assigned a *product type* written $\mathbb{N} \to \mathbb{N}$ and the application of a functional term f to a term t is only valid if the type of t is the same as the domain of the type of f. This *inference rule* corresponds, following the *Curry-Howard correspondence* [Cur34, dB83, How80], to its logical counter part: the well-known *modus ponens* deduction rule.

$$\frac{A \Rightarrow B \qquad A}{B}\ \textsc{ModusPonens} \qquad\qquad \frac{f : A \to B \qquad t : A}{f\ t : B}\ \textsc{Application}$$

The Curry-Howard correspondence identifies the usual notion of theorem with certain types in a type theory. For instance the theorem $A \Rightarrow B$ corresponds to the product type $A \to B$ and a proof of that theorem corresponds to a function taking an argument of type A and defining a term of type B.

Variables must also be assigned a type and, unless that type can be inferred, they are therefore usually annotated in quantifications $\forall x, P(x)$ or functions $x \mapsto f(x)$ which are represented with *dependent product* $\Pi x : \mathbb{N}. P\ x$ and *lambda abstractions* $\lambda x : \mathbb{N}. f\ x$ respectively. Based on these ideas, many type theories have emerged since then. In particular, types themselves can be represented as terms which have a special "type of all types": `Type`. This allows to have *higher-order* quantifications over properties or predicates. Just like the set of all sets, which, as Girard showed [Gir72], cannot be a set, neither can type theories allow `Type` : `Type`. A way around this limitation is to introduce an extra *universe* of types, `Kind`, lying above `Type` such that `Type` : `Kind`. Proceeding this way, it is then possible to successively define an infinite ascending hierarchy of universes to express ever more complex mathematical concepts.

1.2 A unifying logical framework

Since the work of Whitehead and Russell on type theory for foundation of mathematics [WR27], many proof assistants have been developed.

The huge number of these proof systems reflects the many different "ways to consider mathematics", each of them being adapted to their respective culture, traditions, fields of mathematics, automated tools and applications. A proof assistant based on the homotopy of type theory principles and an other one designed for hardware verification are bound to make different choices in the definition of their core logic. However many of these systems usually share similar features, such as term representation of proofs, programs, theorems and types. Systems may sometimes differ only by a small variation in some of these choices, but when it comes to logic, a slight variation may be enough to define a fundamentally different system. Because of the diversity of available proof systems, similar proofs of similar theorems are re-developed in many proof systems.

Translation is a solution to avoid this duplication of work and allow some form of *interoperability* between proof systems. Developing translators is however costly since it requires, in order to be implemented, an in-depth technical understanding of both the target and origin logics and systems. Besides, the number of required translators to achieve full interoperability between any two systems grows quadratically with the number of systems which is not sustainable. When the translated systems share some elementary features, such as λ-abstractions or logical operators, with the system it is encoded in, it is convenient and efficient to reuse this feature. However, differences in formalism mean that such a *shallow* representation is not always possible and translation must often either be partial or rely on a *deep* implementation of the missing feature using an explicit encoding which must be proven equivalent.

A different solution would be to specify and agree on a common language and logical framework in which all developments from other systems such as theorems, proofs,

definitions and so on, can be expressed, automatically translated and easily manipulated. Such a framework should remain simple enough to be easily understood and trusted but be expressive enough to represent all the various features from other logics. It should also preserve the usual key properties of formal systems, such as consistency and decidability of type checking, and allow the re-checking of libraries expressed in it with the same level of confidence as in the original system. Finally a logical framework is a tool to study the various logic themselves, compare the features they implement and design simpler or shorter proof that, in a context of interoperability, would also be easier to express in other partially compatible formalisms.

1.2.1 The lambda-Pi-calculus modulo

In this manuscript, we focus on the *lambda-Pi-calculus modulo*, written $\lambda\Pi_{\equiv}$. This system extends the well-known simply typed λ-calculus with *dependent types* and *higher-order term rewriting*. It is an extension of the lambda-Pi-calculus, $\lambda\Pi$, also known as LF, for "Logical Framework" [HHP93a]. Relying on the λ-calculus seems natural when defining a logical framework for type theories. It provides a direct representation for variables, abstractions and quantifications as well as a native notion of computation for functional terms. Proofs are represented as λ-terms which act as fully explicit witnesses that a theorem holds. Finally, the simplicity of this system provides efficiency and some enjoyable typing and syntactical guarantees such as a stratification of terms as *objects*, *types* or *kinds*.

Despite its simplicity, this system is expressive enough to represent a large class of different logics, including feature-rich type systems, by means of *encodings*. The power of $\lambda\Pi$ relies both on its dependent types which allow to represent more complex type relations, such as hierarchy of universes, and on its conversion relation which allows computational embeddings where computations in the original systems are represented with similarly computing terms. In particular it allows a shallow representation of the usual functional β-reduction which is a key component of most computational type theories based on the Curry–Howard correspondence.

1.2.2 Higher-order term rewriting

Most logical systems rely on a notion of implicit "sameness" between objects. This relation, written $\equiv$ in this manuscript, allows to identify objects that are considered identical or "equivalent". For instance, even though they have different syntactical representations, the sets $\{a, b\}$, $\{b, a\}$ and $\{a\} \cup \{b\}$ are considered identical in set theory without the need for any proof. This sameness can sometimes become quite confusing. For instance much larger explicit finite sets may require a lot more work to be decided equivalent. One way to check that two sets are identical is to check that each element in the first one also belongs to the second and vice-versa. This technique to check that two sets are equivalent is called an algorithm. A more complex but faster algorithm would be to sort the elements in each set and to check that both resulting sets are syntactically equal.

This notion of implicit sameness can be local, "Assume $e := \{\varnothing\}$. Then ..." implicitly means that the symbol e is now considered identical to its definition $\{\varnothing\}$ in the directly

following few sentences but not necessarily in the rest of the development. It can depend on the semantics: the η-expansion rule states that $t \equiv \lambda x.\, t\ x$ if and only if the term t has a product type, which is a semantic property.

In type theory it is usually desirable that the sameness relation $\equiv$ is *computable* so that mathematical developments relying on it can be automatically checked. In order to define an algorithm for checking sameness it is convenient to ensure $\equiv$ can be defined by an oriented *reduction relation* customarily represented with an arrow, $\longrightarrow$.

Term rewriting is a relation on terms defined with the instances of *rewrite rules*. For instance, the rule $\texttt{plus}\ X\ \texttt{0} \longrightarrow X$ allows to reduce $\texttt{plus}\ \texttt{0}\ \texttt{0}$ to $\texttt{0}$, written $\texttt{plus}\ \texttt{0}\ \texttt{0} \longrightarrow \texttt{0}$. A finite set of rewrite rules, called a *rewrite system*, therefore allows to define an infinite relation on the infinite set of terms. Rewrite systems are quite convenient since they allow to define general computational conversion relations that can be parameterized to fit the various needs of encoded logical systems.

In order to better represent other complex computational mechanisms and to make explicit some mechanisms of encoded systems, the conversion rule of $\lambda\Pi_{\equiv}$ is based on *higher-order term rewriting*. Several type checkers for $\lambda\Pi_{\equiv}$ have been implemented, including Dedukti which our work is based upon. A substantial part of our contribution lies in the formalization and study of higher-order rewriting in the lambda-Pi-calculus. Besides, we contributed to the practical implementation and the recent improvements of the Dedukti tool to meet the ever evolving expectations of its users.

1.2.3 Embedding higher-order logics

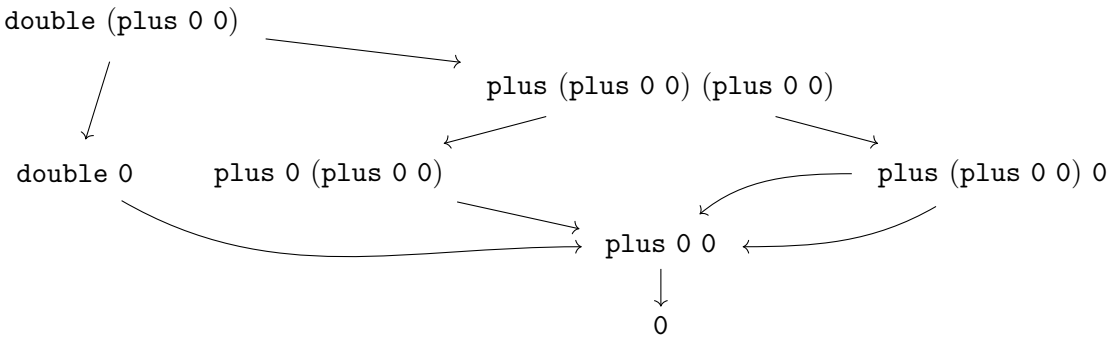
In order to properly embed a system into $\lambda\Pi_{\equiv}$, it is usually necessary to design an encoding system which consists of type assignments, $\texttt{a} : T$, and rewrite rules, $\texttt{f}\ X \longrightarrow X$, as well as a translation mechanism which uniquely defines a syntactical term representation, $[\![t]\!]$ in $\lambda\Pi_{\equiv}$ for any term t well-typed in the original system. This translation may depend on more than the strict syntactical representation of t. We show in this manuscript that it makes sense to have the translation mechanism consider the type of t, the context in which it is well-typed and even the corresponding typing derivation.

The theoretical part of the embedding usually requires to prove that the rewrite system of the encoding is both confluent and terminating. These two properties are a typical first step to show that the defined system has the key properties of subject reduction and decidability of type checking. It is then necessary to show that symbol type assignations define a logic consistent with the original system, a property called *conservativity*. Eventually, the translation function must be defined on all well-typed terms and preserve their well-typedness, a property called *correctness*.

In this manuscript we chose to address two essential issues in the field of logic encoding using term rewriting. In a first part we focus on the confluence property of higher-order rewrite systems in an untyped setting. Many confluence results already exist but the use of term rewriting to define logics often requires tailored criteria for confluence. In a second part we introduce embedding techniques for infinitely sorted type systems with subtyping and prove the correctness of an embedding of the particular case of universe polymorphism in Coq.

1.3 Confluence

Unlike equivalence relations, term rewriting is inherently oriented and is meant to be played exclusively forwards. It corresponds to a term reduction procedure which is computed using pattern matching on terms. Furthermore, term rewriting is often non deterministic. A term may have several reducts depending on the position where a rewrite step is performed, as illustrated by the following reduction diagram which assumes a unary symbol `double` defined by means of the rewrite rule `double` $X \longrightarrow$ `plus` X X.



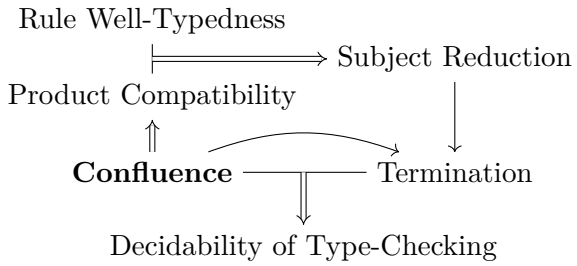
In order for this conversion procedure to be complete, it is critical that rewrite steps are never required to be played backwards. The two terms `double 0` and `plus (plus 0 0) 0` are equivalent since they are both reducts of `double (plus 0 0)`, however this can also be more conveniently checked by computing that they both reduce to `0`.

Therefore, a critical property of rewrite systems is the *Church-Rosser property*, also known as *confluence*, which states that any two equivalents terms can be *joined*, that is to say, reduced to a common term. This property guarantees that when checking whether two terms are convertible, it is sufficient to consider, and compute, their reducts only. In rewrite systems satisfying this property, it is possible to rely on a reduct-enumerating semi-decision procedure for conversion. In terminating systems, this procedure becomes a decision procedure.

1.3.1 Confluence of untyped rewriting

The two essential properties of a computational type theory, consistency and decidability of type checking, usually follow from three simpler ones: type preservation, strong normalization and confluence of the relation on terms defining computation. In dependent type theories however, confluence is often needed to prove type preservation and strong normalization, making all three properties interdependent if termination is used in the confluence proof.

This circularity can be broken in two ways: by proving all properties together within a single induction [Gog94]; or by proving confluence on untyped terms first, and then successively type preservation, confluence on typed terms, and eventually strong normalization.



We develop the latter way here, focusing on untyped confluence. There are several obstacles to confluence of higher-order rewrite systems which we address in this manuscript:

- The untyped setting means that confluence must be proven on all terms of the pure λ-calculus where, in particular, the β-reduction is non-terminating.
- Rewrite rules may overlap, creating *critical peaks* which must be proven joinable.
- Higher-order rewriting may *nest* redexes inside others forcing us to consider simultaneous reduction steps.
- Non-left-linear rewrite rules, such as the computational reflexivity $\texttt{eq}\ X\ X \longrightarrow \texttt{true}$, may be necessary.

In the context of the embedding of ever richer logics, it is in general required to consider complex encoding systems which confluence quickly becomes quite complicated to show. We provide several criteria for confluence of higher-order rewrite systems together with the β rule in the untyped setting. These criteria focus on the complex systems used, in particular, to encode universes in higher-order logics. They are often based on left-linear rules having critical pairs which forbids the use of most existing confluence criteria which rely on the orthogonality property. We show that confluence still holds if higher-order critical pairs have *decreasing diagrams*. Non-left-linear rewrite rules are usually avoided in higher-order settings since it was shown that they easily break confluence, [Klo80]. We show that a syntactical stratification of terms allows to retrieve confluence on a constrained subset of terms.

1.4 Embedding higher-order theories

Some proof systems, such as fully automated theorem provers, ACL2, Mizar or the HOL family of proof assistants, made the choice to rely on a rather simple and well understood core logic which provides some enjoyable logical guarantees at the cost of a somewhat verbose and explicit representation of proofs. What they lack in the expressiveness of their logic they make up for with efficient proof assisting tools such as SAT and SMT solvers. This strategy is close to that of a logical framework and therefore is more easily embedded and used in a context of interoperability. Indeed, proofs can be processed so as to be expressed in a rather minimal logic which relies on as few axioms and logic rules as possible. Such proofs hold in all the extensions of this minimal logic and are therefore more likely to be re-usable in other, more specialized, contexts.

Other systems such as PVS, Coq or Agda decided instead to extend their underlying logic as much as possible while guaranteeing a handful of key properties such as logical

consistency. They do not hesitate to introduce very high-level features such as infinite hierarchy of universes and even *universe polymorphism* in order to increase readability and simplify practical proof development at the cost of a more complex theory. These systems are much more difficult to express in simpler theories and harder to plug together with other theories. Yet, in our endless course towards universality of proof checking, it is crucial to be able to represent them.

In the second part of this manuscript, we focus on the translation of constructs from Coq. The many advanced and complex features of this system make it particularly worthy of interest. Indeed faithfully encoding these features in $\lambda\Pi_{\equiv}$ represents a significant challenge and therefore allows to showcase its expressiveness.

1.4.1 Subtyping in the cumulative hierarchy of universes

A first, already challenging, step is to embed its core logic, the Calculus of Constructions with an infinite hierarchy of universes, introduced by Coquand [Coq85] and studied with Huet [CH85, CH86]. Previous work from Cousineau and Dowek [CD07] relied on a dual representation of terms, as objects and as types of the $\lambda\Pi_{\equiv}$. This technique was originally designed for finitely sorted sets of universes but it can be extended to infinitely hierarchies using a term representation for the universes themselves.

The structure of Cumulative Type System introduces some form of subtyping stemming from cumulativity between universe levels and extending to product types. Assume a type T in the i-th hierarchy of universes, $T : \mathcal{U}_i$, then cumulativity states that T belongs to the universe above as well, $T : \mathcal{U}_{i+1}$. The translation $[\![T]\!]$ in $\lambda\Pi_{\equiv}$ is such that $[\![T]\!] : [\![\mathcal{U}_i]\!]$, however the uniqueness of type property of the encoding system forbids the image of translation to have the same cumulativity property. The embedding of such type systems was studied by Assaf [Ass15b] who relied on an explicit *lifting* operator on object representations : $\uparrow [\![T]\!] : [\![\mathcal{U}_{i+1}]\!]$. Despite having multiple representations for a single term, the correctness of the system is guaranteed by some conversion conditions which ensure that the type T considered in a given universe level j has a single representation up to conversion.

This explicit subtyping operator is extended in this manuscript to product types by means of a more general casting operator. The translation provides fully annotated terms in which the implicit subtyping steps required to derive its well-typedness have all been made explicit. This explicit representation can therefore, in fact, be seen as a representation of its typing derivation. The encoding introduced in this manuscript relies on term rewriting to ensure that all such representations of a single term share a common canonical representation. Since this representation does not necessarily correspond to a canonical typing derivation, it is not in the image of the translation and is kept *private*. Private terms serve as “hidden” unsafe intermediate representations considered only to implement properties such as *proof irrelevance* using term rewriting. In particular private symbols are kept separated from the exposed ”public“ part of the encoding and the translation function does not rely on them.

1.4.2 Universe polymorphism

The Calculus of Constructions with algebraic universes and universe polymorphic definitions was first introduced and studied by Harper and Pollack [HP91, HP89] and implemented in the LEGO system. Universe polymorphism was later introduced in Coq by Sozeau and Tabareau [ST14] though it remains a rather experimental feature. Variants of Coq, such as Matita, chose not to support any explicit form of universe polymorphism. The Agda system also supports universe polymorphic constant declarations where levels are first-class constructions, may explicitly be quantified over and level expressions are objects in the syntax that can be typed. This system however does not feature cumulativity, although it may rely on explicit lifting of universes to express some form of genericity.

For instance the polymorphic identity $\mathtt{id} : \Pi A : \mathtt{Type}_0.\, A \to A$ is of type $\mathtt{Type}_1$ which prevents it to be applied to itself. Using universe polymorphism, it is however possible to define $\mathtt{id}$ at an abstract universe level i, $\mathtt{id}[i] : \Pi A : \mathtt{Type}_i.\, A \to A$, therefore allowing to have: $\mathtt{id}_1\ (\Pi A : \mathtt{Type}_0.\, A \to A)\ \mathtt{id}_0 : \Pi A : \mathtt{Type}_0.\, A \to A$. Universe polymorphism is a prenex and constrained quantification on universe levels. It is a convenient way to handle polymorphic symbols in a more general way than already allowed by cumulativity.

In Coq, universe polymorphism is kept implicit in term representations. Universe levels and constraints are rather inferred and checked on the fly while performing the usual type checking procedure on terms. Naturally, the translation of universe polymorphic definitions requires to make the universe variables and constraints explicit in $\lambda\Pi_{\equiv}$. This is made possible by a representation of algebraic universe levels and parameterized versions of the symbols encoding products, sorts and subtyping. We prove the correctness of the provided encoding which relies on the translation of derivation trees, therefore making all typing steps fully explicit.

1.5 In practice

The main contributions of this thesis are not exclusively theoretical. Many practical developments were permitted by our theoretical results and allowed to put them to the test. We mention here some of the achievements deeply connected to our work. Most of them resulted from collaborations, sometimes with researchers from different fields.

- We adapted and improved the Dedukti tool in order to check our encodings and the sizable libraries of proofs embedded in them. Various new features were implemented such as
  - The optimization of the reduction engine with decision trees and improved term representation and sharing;
  - A subject reduction checker extension to better process the typing constraints;
  - Rewriting modulo associativiy-commutativity which was implemented to work together with the already in-place decision trees.

- Our theoretical work allowed to improve the CoqInE tool, a Coq plugin exporting modules to Dedukti encodings of the Calculus of Inductive Constructions. Admittedly much remains to do in terms of documentation, bug fixing and catching up with the fast evolving latest version of Coq. Thanks to our improvements, this tool is however now able, to check a sizable set of proofs and definitions and is already used in interoperability projects. Some of the added features are:
  - support for universe polymorphism;
  - support for monomorphic, template polymorphic and "true" polymorphic inductive types;
  - a redefined translation of fixpoints as first-class constructions;
  - modules and functors translation;
  - several output encoding, each tailored for a specific need: interoperability, fast type checking, expressiveness, readability, etc;
  - integration in the Logipedia project.
- The CoqInE tool was also used, in collaboration with the team in charge of the `GeoCoq` project, to translate and recheck a sizable library of Coq proofs to Dedukti in an dedicated encoding suitable for these particular interoperability needs.

1.6 Outline of this manuscript

The first part of this manuscript is dedicated to the property of confluence of higher-order term rewriting together with the functional β-reduction in the context of defining encoding theories in the lambda-Pi-calculus modulo.

- In Chapter 2, we define higher-order pattern-matching and position-annotated *term rewriting* in a way fit to be studied together with the usual functional β-reduction. Single-step reduction is extended to simultaneous multi-steps relations, such as *parallel*, *orthogonal* or *sub*-rewriting. In non-terminating systems, studying these extensions is essential in order to prove the confluence of the initial relation. We take a particular care here in annotating rewriting steps with the position where they occur in the term.
- In Chapter 3, we introduce several general purpose confluence criteria for confluence of left-linear higher-order rewrite systems together with β. A first criterion is that a confluent higher-order rewrite system $\mathcal{R}$ remains confluent when considered together with β. This means that techniques relying, for instance, on termination or decreasing diagrams can be used to prove confluence of $\mathcal{R}$ provided critical pairs do not require β-steps to join. Otherwise a second criterion requires to show the existence of decreasing diagrams for all *orthogonal critical pairs* to retrieve confluence with β. In that case β steps are considered of low weight so they can be used for free in the closing diagrams.

- In Chapter 4, we provide confluence criteria for non-left-linear sets of rewrite rules together with β, assuming rewriting is restricted to a class of terms satisfying some syntactical layering conditions. These conditions guarantee a stratification of terms which allows to consider rules which non-linear variables belong to a level strictly lower than that of the left-hand side. Furthermore, the lowest level forbids all β-redexes. This constraint, called *confinement*, freely allows non-left-linear rules within the confined level while locally forbidding interactions with β. As in the previous chapter, we rely on techniques based on van Oostrom's decreasing diagrams [vO94] that reduce confluence proofs to the checking of critical pairs for higher-order rewrite rules. We also discuss practical applications of these results in this chapter.

The second part of this manuscript is dedicated to the definition of an encoding and translation function of the universe polymorphic calculus of inductive constructions in the lambda-Pi-calculus modulo with the practical application of translating and type checking sizable proof libraries from Coq to Dedukti.

- In Chapter 5, we introduce pure type systems and several extensions featuring some form of subtyping stemming from a cumulativity relation on sorts. We introduce and discuss previous work on embedding these systems into $\lambda\Pi_{\equiv}$. Finally we introduce new encoding paradigms which we argue allow to faithfully represent more complicated features such as non-functionality, floating universes or universe polymorphism.

- In Chapter 6, we describe how the Calculus of Constructions can be conservatively extended with an infinite set of sorts relying on (constrained) algebraic levels. We investigate how these levels can be encoded in a way compatible with the natural notions of level equality, inequality and instantiation, allowing their embedding in $\lambda\Pi_{\equiv}$ with the previously defined CTS encoding.

- In Chapter 7, we introduce a type system that conservatively extends the Calculus of Constructions with universe polymorphic definitions and declarations in a signature. We argue that this system remains close enough to a subset of the one implemented in our target system Coq and make safe assumptions to simplify the definition of our translation to $\lambda\Pi_{\equiv}$ which is defined on typing derivations rather than on terms.

- In Chapter 8, we introduce an encoding of $\mathrm{CC}^{\forall}_{\omega}$ into $\lambda\Pi_{\equiv}$ relying on a private representation of lambda terms to ensure preservation of conversion. The translation function is defined on typing derivations rather than on terms. We prove the preservation of typing and conversion of the translation mechanism.

- In Chapter 9, we describe how the previously defined embedding can be adapted and extended to represent other features of Coq such as inductive types and fixpoints. These practical techniques allowed to implement a Coq plugin exporting proofs and definitions into Dedukti which was used to translate several Coq developments, in particular from the `GeoCoq` library.

Part I

Confluent Rewriting in the Logical Framework

Chapter 2

Higher-Order Term Rewriting in the Lambda-Pi-calculus

We start by introducing basic concepts and notations at the core of type theories, as well as several known properties referred to in the rest of the manuscript. Our presentation relies on Klop's Combinatory Reduction Systems (CRS) [Klo80, KvOvR93] which is particularly well-suited for the study of higher-order rewrite rules together with the usual β-reduction. In the context of shallow embedding of logics, which is the main focus behind the development of the lambda-Pi-calculus modulo ($\lambda\Pi_{\equiv}$), term-rewriting is often required to be higher-order and support rewriting of functional terms called λ-abstractions. However, since term rewriting allows to define the type system, it is difficult, in its study, to rely on the well-typedness of the terms it operates on. Therefore it should instead be considered on all terms, enforcing a merely syntactical study of its properties.

In that endeavor, we take a particular care in annotating rewrite steps with the position at which they occur in a term and use this annotation to help characterize rewrite steps. We provide several known properties of terms and of this annotated higher-order rewriting relation on them.

In Section 2.3, the $\lambda\Pi_{\equiv}$ type system is formally defined and we recall the usual properties of this system introduced by Cousineau and Dowek [CD07] and later studied and successively improved by Saillard [Sai15a, Sai15b], Assaf [Ass15b], Thiré [Thi20] and Genestier [Gen20b].

Finally, we discuss several other formalisms fit to extend $\lambda\Pi$ with higher-order term rewriting while preserving the properties of the type system. The main challenges are that term rewriting needs to be type preserving, confluent and considered in an untyped setting before it can be used to extend the typing relation of $\lambda\Pi$. In particular we argue that the choice of Klop's CRS allows to better rely on positional rewriting.

2.1 Terms

We define in this section the usual notions of terms of the λ-calculus, positions, α-conversion and substitution.

2.1.1 Syntax

We consider the terms of the pure λ-calculus enriched with a set $\mathcal{F}$ of *symbols* called a *signature*.

Definition 2.1.1 (Term)**.** Terms *are generated from an infinite set $\mathcal{X}$ of variables and a set $\mathcal{F}$ of symbols. The set of terms $\mathcal{T}_{\mathcal{X},\mathcal{F}}$ (usually written $\mathcal{T}$) is inductively defined as follows:*

$$s, t, u, v, \ldots \quad := \quad x \in \mathcal{X} \quad | \quad \mathtt{f} \in \mathcal{F} \quad | \quad u\ v \quad | \quad \lambda x : t.\, u$$

As mentioned, even though they are meant to be dependently typed, terms shall not rely on the type of symbols as it is defined independently and properties like confluence or product compatibility are often needed before defining a usable type system. Therefore no typing relation is considered in the definition of terms which is purely syntactical. In particular all symbols in $\mathcal{F}$ have arity 0 and can only be applied using an application construction to form a currified algebraic expression. For instance the term $\lambda x : s\ t.\, u\ v = \lambda x : (s\ t).\, (u\ v)$ is an abstraction and $t = \mathtt{f}\ x\ (\mathtt{g}\ y)\ \lambda x : t.\, x$ is the application: $t = ((\mathtt{f}\ x)\ (\mathtt{g}\ y))\ (\lambda x : t.\, x)$. To ease readability, we may, on some occasions rely on the traditional function application notation, $\mathtt{f}(t, u) := (\mathtt{f}\ t)\ u = \mathtt{f}\ t\ u$, or use infix notations, $t{+}u = (+\ t)\ u$, or even more complex notations for symbols with many arguments: if ${}^{\square}_{\square}{\uparrow}^{\square}_{\square} \in \mathcal{F}$ then we write ${}^{s_2}_{s_1}{\uparrow}^{t_2}_{t_1}$ for the term $(((\uparrow\ s_1)\ s_2)\ t_1)\ t_2 = \uparrow\ s_1\ s_2\ t_1\ t_2$.

Definition 2.1.2 (Subterm)**.** *The* subterm *relation, $\lhd$, on terms is defined as the smallest reflexive, transitive relation such that: $u \lhd u\ v$, $v \lhd u\ v$, $t \lhd \lambda x : t.\, u$ and $u \lhd \lambda x : t.\, u$. It is a well-founded partial order on terms.*

Definition 2.1.3 (Free Variables)**.** *We write $\mathcal{V}ar(t) := \{x \in \mathcal{X} | x \lhd t\}$ the set of variables occurring in a term t. The finite set $\mathcal{FV}ar(t) \subseteq \mathcal{V}ar(t)$ of variables* free *in a term $t \in \mathcal{T}$ is inductively defined as follows:*

$$\mathcal{FV}ar(x) := \{x\} \qquad \mathcal{FV}ar(u\ v) := \mathcal{FV}ar(u) \cup \mathcal{FV}ar(v)$$
$$\mathcal{FV}ar(\mathtt{f}) := \varnothing \qquad \mathcal{FV}ar(\lambda x : t.\, u) := \mathcal{FV}ar(u) \setminus \{x\} \cup \mathcal{FV}ar(t)$$

Terms without free variables are closed*. A variable x occurring in the body u of the corresponding binder $\lambda x : t.\, u$ is said to be* bound *(or* bound above*).*

Definition 2.1.4 (α-equivalence)**.** Renaming *of the variable x with y in a term t, written $t\{y/x\}$, is the syntactical replacement of all occurrences of x with y, ignoring capture:*

$$\mathtt{f}\{y/x\} := \mathtt{f} \qquad (u\ v)\{y/x\} := (u\{y/x\}\ v\{y/x\})$$
$$x\{y/x\} := y \qquad (\lambda x : t.\, u)\{y/x\} := \lambda x : t\{y/x\}.\, u$$
$$z\{y/x\} := z \qquad (\lambda z : t.\, u)\{y/x\} := \lambda z : t\{y/x\}.\, u\{y/x\} \quad \text{if } z \neq x$$

The α-equivalence, written $\equiv_\alpha$ is the smallest reflexive, symmetric and transitive relation on terms such that for all terms t and $y \notin \mathcal{V}ar(t)$ $\lambda x.t \equiv_\alpha \lambda y.t\{y/x\}$ and if $u \equiv_\alpha v$, then $t\ u \equiv_\alpha t\ v$, $u\ t \equiv_\alpha v\ t$, $\lambda z : t.\, .u \equiv_\alpha \lambda z : t.\, .v$ and $\lambda z : u.\, .t \equiv_\alpha \lambda z : v.\, .t$.

From now on, all terms are considered equal when α-equivalent. This means that functions and relations defined on terms are all compatible with α-equivalence even though we usually choose to omit the proof of this property. We could show, by induction, that α-equivalent terms have the same set of free variables or that if $t \equiv_\alpha r \lhd v$, then there is some u such that $t \lhd u \equiv_\alpha v$. From this last implication we easily deduce that $\lhd$ extends to a well-founded partial order modulo α.

2.1.2 Meta-terms

Meta-terms are terms enriched with *meta-variables* from an infinite set $\mathcal{Z}$. Meta-variables are equipped with arities in $\mathbb{N}$ and may be applied to meta-terms.

Definition 2.1.5 (Meta-terms)**.** Meta-terms *extend terms with a family $\mathcal{Z}^n$ of infinite sets of* meta-variables *indexed by their arity n. The set of meta-terms, $\mathcal{MT}_{\mathcal{X},\mathcal{F},\mathcal{Z}}$, usually written $\mathcal{MT}$, is defined inductively as follows:*

$$\mathcal{MT}_{\mathcal{X},\mathcal{F},\mathcal{Z}} \quad : \qquad t,u,v \;:=\; x \in \mathcal{X} \mid \mathtt{f} \in \mathcal{F} \mid u\ v \mid \lambda x\!:\!t.\,u \mid Z[t_1,\ldots,t_n] \qquad \text{for } Z \in \mathcal{Z}^n$$

If $X \in \mathcal{Z}^0$, we write X instead of $X[\,]$. Note that $\mathcal{T} \subseteq \mathcal{MT}$.

Definition 2.1.6. *We define the finite set $\mathcal{MV}ar(t) := \{Z \in \mathcal{Z} \mid Z[\overline{u}] \lhd t\}$ of meta-variables in a meta-term $t \in \mathcal{MT}$. The notions of subterm, free variables and closedness extend to meta-terms.*

2.1.3 Substitutions

Definition 2.1.7 (Substitute)**.** *An n-*ary substitute *is an expression $\underline{\lambda}x_1 \ldots \underline{\lambda}x_n.u$ (or $\underline{\lambda}x_1 \ldots x_n.u$ or $\underline{\lambda}\overline{x}.u$) where u is a meta-term and $x_1,\ldots,x_n$ are pairwise distinct variables.*

Note that the new symbol $\underline{\lambda}$ means that substitutes are neither terms nor meta-terms but different objects. In other presentations such as Nipkow's HORS, substitutes are included in the meta-terms and substitution is seen as the (meta) β-reduction.

Definition 2.1.8 (Substitution)**.** *A* substitution *σ is defined as a function mapping variables $x \in \mathcal{X}$ to meta-terms and meta-variables $X \in \mathcal{Z}^n$ to n-ary substitutes, such that its* domain *$\mathcal{D}om(\sigma) := \{x \in \mathcal{Z} \cup \mathcal{X} \mid \sigma(x) \neq x\}$ is finite. Substitutions are usually written $\sigma = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$, or $\sigma = \{\overline{x} \mapsto \overline{u}\}$.*

The definitions of (free) (meta-)variables and closedness extend to substitutes and substitutions in a natural way, $\mathcal{FV}ar(\underline{\lambda}\overline{x}.t) := \mathcal{FV}ar(t)\backslash\overline{x}$ and $\mathcal{FV}ar(\sigma) := \mathcal{FV}ar(\sigma(\mathcal{D}om(\sigma)))$.

The range *of a substitution σ is defined as $\mathcal{R}an(\sigma) := \mathcal{V}ar(\sigma) \uplus \mathcal{MV}ar(\sigma)$.*

A meta-substitution *is a substitution Θ such that $\mathcal{D}om(\Theta) \subseteq \mathcal{Z}$.*

A valuation *is a meta-substitution which range contains no meta-variables.*

A term-substitution *is a substitution σ such that $\mathcal{D}om(\sigma) \subseteq \mathcal{X} \supseteq \mathcal{R}an(\sigma)$.*

Definition 2.1.9. *A substitution σ inductively defines a* capture-avoiding *homomorphism on meta-terms.*

$$\begin{array}{rcl} \mathtt{f}\sigma & := & \mathtt{f} \\ x\sigma & := & \sigma(x) \\ (u\ v)\sigma & := & (u\sigma\ v\sigma) \end{array} \qquad \begin{array}{rcl} (\lambda x\!:\!t.\,u)\sigma & := & \lambda y\!:\!t\sigma.\,u\sigma_{y/x} \\ Z[\overline{u}]\sigma & := & \begin{cases} t\{\overline{x} \mapsto \overline{u}\sigma\} & \text{if } \sigma(Z) = \underline{\lambda}\overline{x}.t \\ Z[\overline{u}\sigma] & \text{otherwise} \end{cases} \end{array}$$

with $\mathcal{D}om(\sigma_{y/x}) := \mathcal{D}om(\sigma) \cup \{x\}$, $\sigma_{y/x}(x) := y$ and $\forall z \neq x, \sigma_{y/x}(z) := \sigma(z)$ for some fresh variable y of same arity than x outside $\mathcal{R}an(\sigma) \cup \mathcal{V}ar(t)$.

Note that whenever $x \notin \mathcal{R}an(\sigma)$, $(\lambda x\!:\!t.\,u)\sigma = \lambda x\!:\!t\sigma.\,u\sigma$ and $(\underline{\lambda}x.t)\sigma = \underline{\lambda}x.(t\sigma)$.

A term $t\Theta \in \mathcal{T}$ is an instance *of the meta-term t if Θ is a meta-substitution.*

This homomorphism naturally extends to sequences of (meta-)terms as well as to substitutions themselves:

$$\sigma\tau := \biguplus_{x \in \mathcal{D}om(\sigma) \cup \mathcal{D}om(\tau)} \left\{ x \mapsto \begin{cases} \sigma(x)\tau & \text{if } x \in \mathcal{D}om(\sigma) \\ \tau(x) & \text{otherwise} \end{cases} \right\}$$

Meta-terms implicitly represent the set of their instances. Note that valuations are enough to define the instances of a meta-term t.

The application of substitutions is inductive well-defined since meta-terms syntactically forbid non-terminating substitutions such as the infamous $X[X]\{X \mapsto \underline{\lambda}x.X[x]\}$ which is invalid since no arity can be assigned to the variable X.

Lemma 2.1.10. *The application of a substitution Θ is well-defined.*

Proof. We define the order of Θ, $ord(\Theta)$ to be 0 if $\mathcal{D}om(\Theta) \subseteq \mathcal{X} \cup \mathcal{Z}^0$ and 1 otherwise. We label every application $t\Theta$ with the pair $(ord(\Theta), t)$ in $\{0,1\} \times \mathcal{T}$ which is equipped with the well-founded order $(\leq, \lhd)_{lex}$. The definition of the application $t\Theta$ of a substitution relies exclusively on applications of lower label: for the first two rules it relies on nothing. For the following two and the very last, it relies on the application of a substitution of the same order to strict subterms. Finally for the remaining second to last rule, it relies on both the application of the same substitution to a strict subterm and the application of a substitution or order 0 while Θ is of order 1. □

Lemma 2.1.11. *Given u, σ, τ, $(u\sigma)\tau = u(\sigma\tau)$ (written $u\sigma\tau$).*

As an example, assume $t = \lambda x\!:\!\mathtt{f}\ y.\,Z[x]$, $\Theta = \{Z \mapsto \underline{\lambda}u.u\ x\ y\}$ and $\sigma = \{y \mapsto x\}$, then

$$\begin{array}{lll} (t\Theta)\sigma & = (\lambda z\!:\!\mathtt{f}\ y.\,z\ x\ y)\sigma & = \lambda z\!:\!\mathtt{f}\ x.\,z\ x\ x \\ (t\sigma)\Theta & = (\lambda z\!:\!\mathtt{f}\ x.\,Z[z])\Theta & = \lambda z\!:\!\mathtt{f}\ x.\,z\ x\ y \\ t(\Theta\sigma) & = t\{Z \mapsto \underline{\lambda}z.z\ x\ x, y \mapsto x\} & = \lambda z\!:\!\mathtt{f}\ x.\,z\ x\ x \end{array}$$

Lemma 2.1.12. *Term-substitutions define capture-avoiding homomorphisms of terms: $t \in \mathcal{T} \Rightarrow t\sigma \in \mathcal{T}$. Besides if σ, τ are term-substitutions so is $\sigma\tau$.*

2.1.4 Positions

The subterms of a term t can be referenced by the *position* at which they occur inside t. We define here usual sets of positions.

Definition 2.1.13 (Position)**.** *We define* positions *as usual position in a tree.*

Positions *are words on the alphabet $\mathbb{N}$ of natural numbers.*

The empty word is written Λ and is called the root*.*

Positions are naturally equipped with a well-founded order $\leq_{\mathcal{P}}$ such that $p \leq_{\mathcal{P}} p \cdot q$ and its strict part is written $<_{\mathcal{P}}$.

Their inverse versions are respectively written $\geq_{\mathcal{P}}$ and $>_{\mathcal{P}}$.

When $p \leq_{\mathcal{P}} q$, we write that q is below *p or that p is* above *q which may seem odd but is natural as bigger positions in a term correspond to deeper subterms.*

This order is not total. We write $p\#q$ when neither $p \leq_{\mathcal{P}} q$ nor $q \leq_{\mathcal{P}} p$, in which case we say that p and q are parallel *or* incomparable*.*

This means that for all p and q either $p = q$ or $p <_{\mathcal{P}} q$ or $q <_{\mathcal{P}} p$ or $p\#q$.

$p \not\leq_{\mathcal{P}} q$ means that $p \leq_{\mathcal{P}} q$ does not hold, so that $p \not\leq_{\mathcal{P}} q \Rightarrow p\#q \vee q <_{\mathcal{P}} p$.

Definition 2.1.14 (Sets of positions)**.** *A set of positions P is said to be* parallel *if $\forall p, q \in P, p \neq q \Rightarrow p\#q$. Concatenation extends to sets of positions to the right, $p \cdot Q := \{p \cdot q \mid q \in Q\}$ and preserves parallelism. We also extend the $\leq_{\mathcal{P}}$ order to sets of positions:*

- *$P \leq_{\mathcal{P}} Q$ iff $\forall q \in Q, \exists p \in P, p \leq_{\mathcal{P}} q$; Note that this does not mean that all words in P have an extension in Q. In particular we have $P \leq_{\mathcal{P}} \varnothing$.*
- *$p \leq_{\mathcal{P}} Q$ iff $\{p\} \leq_{\mathcal{P}} Q$ i.e. $\forall q \in Q, p \leq_{\mathcal{P}} q$;*
- *$P \leq_{\mathcal{P}} q$ iff $P \leq_{\mathcal{P}} \{q\}$ i.e. $\exists p \in P, p \leq_{\mathcal{P}} q$;*
- *$P\#Q$ iff $\forall p \in P, \forall q \in Q, p\#q$.*
- *$P \not\leq_{\mathcal{P}} Q$ iff $\forall p \in P, \forall q \in Q, p \not\leq_{\mathcal{P}} q$. This implies but is not equivalent to $\neg(P \leq_{\mathcal{P}} Q)$;*
- *$P \not\leq_{\mathcal{P}} q$ and $p \not\leq_{\mathcal{P}} Q$ are shortcuts for $P \not\leq_{\mathcal{P}} \{q\}$ and $\{p\} \not\leq_{\mathcal{P}} Q$ respectively.*

All definitions extend to their strict version, $<_{\mathcal{P}}$. We sometimes write $Q \geq_{\mathcal{P}} P$ to mean $P \leq_{\mathcal{P}} Q$. Note that this does not correspond to the above extension of the $\geq_{\mathcal{P}}$ relation to sets of positions.

Definition 2.1.15. *Given a position p and a set of positions Q, we define the set of positions $p^{-1}(Q) := \{q \mid p \cdot q \in Q\}$.*

Definition 2.1.16 (Orthogonal Union)**.** *Given sets of positions P and Q such that P is parallel and $Q >_{\mathcal{P}} P$, the disjoint union $P \uplus Q$ is called* orthogonal *and written $P \otimes Q$.*

Lemma 2.1.17. *If $O = P \otimes Q$, then $P \leq_{\mathcal{P}} O$ and $Q = \biguplus_{p \in P} p \cdot (p^{-1}(O) \setminus \{\Lambda\}) = O \setminus P$.*

Proof. Straightforward since $Q >_{\mathcal{P}} P$ by definition. □

Lemma 2.1.18 (Orthogonal Decomposition)**.** *Given a set of positions O there exists a unique (P, Q) such that $O = P \otimes Q$.*

Proof. We define $P := \{p \in O \mid \forall q \in O, q \not<_{\mathcal{P}} p\}$ and $Q := \biguplus_{p \in P} p \cdot (p^{-1}(O) \setminus \{\Lambda\})$. We check that $P \otimes Q = O$. Let $P' \otimes Q' = O$ be another candidate, then, by Lemma 2.1.17, $P' \geq_{\mathcal{P}} P \geq_{\mathcal{P}} P'$. Assume $p \in P$ then $p = p' \cdot o$ for some $p' \in P'$ and $p' = p'' \cdot o'$ for some $p'' \in P$. Since P is parallel and $p \geq_{\mathcal{P}} p''$, we conclude $p = p''$, $o = \Lambda$, $p = p' \in P'$ and finally $P \subseteq P'$. We conclude $P = P'$ and by Lemma 2.1.17 we deduce $Q = Q'$. □

P is called the *parallel* part of O, written $\underline{O}$, while Q is called the *residual* part of O, written $\overline{O}$. Note that $O = \underline{O} \uplus \overline{O}$ and that whenever $O \neq \emptyset$, $\underline{O} \neq \emptyset$ and $\overline{O} \subsetneq O$.

Lemma 2.1.19. *Orthogonal decomposition satisfies the following properties:*

- *If $O_1 = P_1 \otimes Q_1$ and $O_2 = P_2 \otimes Q_2$ with $P_1 \# P_2$ then $O_1 \uplus O_2 = (P_1 \uplus P_2) \otimes (Q_1 \uplus Q_2)$*
- *If $O_1 \# O_2$ then $\underline{O_1 \uplus O_2} = \underline{O_1} \uplus \underline{O_2}$ and $\overline{O_1 \uplus O_2} = \overline{O_1} \uplus \overline{O_2}$*
- *If $S >_{\mathcal{P}} \underline{O}$ then $\underline{O \cup S} = \underline{O}$ and $\overline{O \cup S} = \overline{O} \cup S$*

Proof. All are straightforward by definition. □

Definition 2.1.20. *The set of positions $\mathcal{P}os(t)$ in a term t is defined as follows:*

$$\begin{aligned} \mathcal{P}os(x) = \mathcal{P}os(\mathtt{f}) &:= \{\Lambda\} \\ \mathcal{P}os(u\ v) &:= \{\Lambda\} \uplus 1 \cdot \mathcal{P}os(u) \uplus 2 \cdot \mathcal{P}os(v) \\ \mathcal{P}os(\lambda x\!:\!t.\, u) &:= \{\Lambda\} \uplus 1 \cdot \mathcal{P}os(t) \uplus 2 \cdot \mathcal{P}os(u) \end{aligned}$$

This definition naturally extends to meta-terms, $\mathcal{P}os(Z[t_1, \ldots, t_n]) := \{\Lambda\} \uplus \biguplus_i i \cdot \mathcal{P}os(t_i)$.

Definition 2.1.21. *Assume t is a (meta-)term and $p \in \mathcal{P}os(t)$. The* subterm of t at position p, *$t|_p$, is inductively defined as follows: $t|_\Lambda := t$, $(u\ v)|_{1 \cdot p} := u|_p$, $(u\ v)|_{2 \cdot q} := v|_p$, $(\lambda x\!:\!t.\, u)|_{1 \cdot p} := t|_p$, $(\lambda x\!:\!t.\, u)|_{2 \cdot p} := u|_p$ and $Z[t_1, \ldots, t_n]|_{i \cdot p} := t_i|_p$.*

We define the symbol at position p in t, *$t(p) \in \mathcal{F} \cup \mathcal{X} \cup \mathcal{Z} \cup \{@, \boldsymbol{\lambda}\}$, as the head of $t|_p$.*

As not all positions are valid in a term, writing $t|_p$ implicitly means that $p \in \mathcal{P}os(t)$.

Definition 2.1.22. *Assume u is a term. We define the following subsets of $\mathcal{P}os(u)$:*

- *$\mathcal{P}os(x, u) := \{p \in \mathcal{P}os(u) \mid u|_p = x\}$ for $x \in \mathcal{X}$, which is parallel;*
- *$\mathcal{VP}os(u) := \bigcup_{x \in \mathcal{X}} \mathcal{P}os(x, u)$, which is parallel;*
- *$\mathcal{P}os(Z, u) := \{p \in \mathcal{P}os(u) \mid u(p) = Z\}$ for $Z \in \mathcal{Z}$;*
- *$\mathcal{MP}os(u) := \bigcup_{Z \in \mathcal{Z}} \mathcal{P}os(Z, u)$.*

We define the set of variables bounded above p in u, *$bv(u, p) := \mathcal{FV}ar(u|_p) \setminus \mathcal{FV}ar(u)$.*

2.1.5 Splitting

Positions are useful to characterize several term transformations. In particular they allow to define the *splitting* of a term into an "above" contextual term and a "below" higher-order substitution.

Definition 2.1.23 (Replacement)**.** Replacement *at position p in a (meta-)term t, written $t[w]_p$, is defined as follows:*

$$\begin{aligned} t[w]_\Lambda &:= w & Z[u_1,\dots,u_n][w]_{i\cdot p} &:= Z[u_1,\dots,u_i[w]_p\dots,u_n] \\ (\lambda x:t.\,u)[w]_{1\cdot p} &:= \lambda x:t[w]_p.\,u & (u\ v)[w]_{1\cdot p} &:= (u[w]_p\ v) \\ (\lambda x:t.\,u)[w]_{2\cdot p} &:= \lambda x:t.\,u[w]_p & (u\ v)[w]_{2\cdot p} &:= (u\ v[w]_p) \end{aligned}$$

If we write $t[u]_p$, it is always assumed that $p \in \mathcal{P}os(t)$. Replacement at a position p is not an homomorphism and it is not compatible with α conversion. It should not be seen as an operation on terms but rather as a way to refer to syntactical changes in a term as it is done in the definition of splitting.

Definition 2.1.24 (Splitting)**.** *Let t be a meta-term and $p \in \mathcal{P}os(t)$. We define the* splitting *of t at position p as the pair $(\underline{t}_p, \overline{t}^p)$ formed with a meta-term $\underline{t}_p \in \mathcal{MT}$ called the* above splitting *of t at p and a meta-substitution $\overline{t}^p$ called the* below splitting *of t at p. Assume $\overline{x} = \mathcal{V}ar(t|_p) \setminus \mathcal{V}ar(t) \subseteq bv(t,p)$, then $\underline{t}_p := t[\,Z[\overline{x}]\,]_p$ and $\overline{t}^p := \{Z \mapsto \underline{\lambda}\overline{x}.t|_p\}$ for a fresh meta-variable $Z \in \mathcal{Z}^{|\overline{x}|}$.*

We assume a canonical choice of meta-variables Z and X_i such that the splitting $(\underline{t}_p, \overline{t}^p)$ is uniquely defined.

Our use of splitting in this manuscript will be systematic unless it alters readability for no good reason. This invention permitted by Klop's notion of meta-variable, is the only technique we know of which allows to manipulate terms with binders safely, in case renaming of variables needs to take place independently in a term and in its context, as will often be the case here.

Lemma 2.1.25 (Properties)**.** *For terms t, u, substitution σ and positions $p, q \in \mathcal{P}os(t)$:*

1. *$u = u[\,u|_p\,]_p$ and $(u[t]_p)|_p = t$*
2. *Assuming variables of σ are not bounded in t, $(t\sigma)|_p = t|_p\sigma$ and $t[u]_p\sigma = t\sigma[u\sigma]_p$*
3. *if $p\#q$, $t[u]_p[v]_q = t[v]_q[u]_p$*
4. *$t[u]_{p\cdot q} = t[\,t|_p[u]_q\,]_p$*
5. *$u = \underline{u}_p\overline{u}^p$*
6. *$\mathcal{FV}ar(\underline{u}_p) \cup \mathcal{FV}ar(\overline{u}^p) = \mathcal{FV}ar(u)$ and $\mathcal{MV}ar(\underline{u}_p) \cup \mathcal{MV}ar(\overline{u}^p) = \mathcal{MV}ar(u)$*
7. *if $p\#q$ then we have $\underline{\underline{u}_p}_q = \underline{\underline{u}_q}_p$, $\overline{t}^p\overline{u}^q = \overline{t}^q\overline{u}^p$, $\underline{u}_p|_q = u|_q$ and $\overline{\underline{u}_p}^q = \overline{u}^q$*
8. *$\underline{\underline{u}_{p\cdot q}}_p = \underline{u}_p$ and $\underline{u}_p\sigma = \underline{u}_{p\cdot q}$ for some σ.*

Proof. (1.) By a simple induction on p in both cases. (2.) and (3.) by induction on t. (4.) By induction on p. (5.) Since Z is fresh in t, by (2.), $\underline{t}_p\overline{t}^p = t[\,Z[\overline{x}]\,]_p\overline{t}^p = t[\,Z[\overline{x}]\overline{t}^p\,]_p$. We conclude by (1.) since $Z[\overline{x}]\overline{t}^p = t|_p\{\overline{x} \mapsto \overline{x}\} = t|_p$. (6.) $\supseteq$ follows from (5.) and $\subseteq$ by definition. (7.) Follows from (3.) and the fact that the introduced meta-variables are both fresh. (8.) Using (4.) and (5.), $\sigma := \overline{\underline{t}_{p\cdot q}}^p$. □

These properties allow to generalize splitting to parallel sets of positions unambiguously.

Definition 2.1.26 (Parallel Splitting)**.** *Given a term u and a set $P = \{p_1, \ldots, p_n\}$ of parallel positions in u, we define the splitting of u along P as the meta-term and meta-substitution $(\underline{u}_P, \overline{u}^P)$ such that $\underline{u}_P := u[Z_1[\overline{x_1}]]_{p_1} \ldots [Z_n[\overline{x_n}]]_{p_n}$ (u is cut above P) and $\overline{u}^P := \{Z_i \mapsto \lambda\overline{x_i}.u|_{p_i} \mid 1 \leq i \leq n\}$ (u is cut above P) where, for all i, $\overline{x_i}$ is the list of all variables of $u|_{p_i}$ bound in u above p_i and Z_i is a fresh meta-variable or arity $|\overline{x_i}|$.*

Lemma 2.1.27. *If $P\#Q$ both parallel, then $\underline{\underline{u}_P}_Q = \underline{u}_{P \uplus Q}$, $\overline{u}^Q \overline{u}^P = \overline{u}^{P \uplus Q}$ and $\overline{\underline{u}_P}^Q = \overline{u}^Q$.*

Example 1: Assume $u := \lambda x\!:\!\mathtt{T}.\,\mathtt{f}\ x\ \mathtt{0}$, $P = \{21\}$ and $Q = \{22\}$. We have:

$$\begin{array}{rcll} \underline{u}_P & = & \lambda x\!:\!\mathtt{T}.\,X[x]\ \mathtt{0} & \overline{u}^P = \overline{\overline{u}^Q}^P = \{X \mapsto \underline{\lambda}x.\mathtt{f}\ x\} \\ \underline{u}_Q & = & \lambda x\!:\!\mathtt{T}.\,\mathtt{f}\ x\ Y & \overline{u}^Q = \overline{\overline{u}^P}^Q = \{Y \mapsto \mathtt{0}\} \\ \underline{\underline{u}_P}_Q = \underline{\underline{u}_Q}_P & = & \lambda x\!:\!\mathtt{T}.\,X[x]\ Y & \overline{u}^{P \uplus Q} = \{X \mapsto \underline{\lambda}x.\mathtt{f}\ x, Y \mapsto \mathtt{0}\} \end{array}$$

2.2 Higher-order term rewriting

2.2.1 Abstract rewriting

Given a binary relation $\longrightarrow$ on an abstract set $\mathcal{A}$, we denote with $\longleftarrow$ its inverse, $\longleftrightarrow$ its symmetric closure, $\longrightarrow^=$ its reflexive closure, $\longrightarrow\!\!\!\!\!\rightarrow$ its reflexive and transitive closure (also called *derivation* or *rewrite sequence*) and $\twoheadleftarrow\!\!\!\!\!\rightarrow$ or $\equiv$ its closures by reflexivity, symmetry and transitivity (called *convertibility*). Relations are identified with a name below such as $\underset{\beta}{\longrightarrow}$, $\underset{L \to R}{\longrightarrow}$ or $\underset{\mathcal{R}}{\longrightarrow}$. A triple $u \longrightarrow v$ is called a *step* and chain of steps $u_1 \longrightarrow u_2 \longrightarrow \cdots \longrightarrow u_n$ is called a *sequence*.

A element $s \in \mathcal{A}$ is *normal*, or *in normal form*, if there is no t such that $s \longrightarrow t$. If t is normal and $s \longrightarrow\!\!\!\!\!\rightarrow t$ then s *has a normal form*, or is *weakly normalizing*. If there is no infinite rewriting sequence $t \longrightarrow t_1 \longrightarrow \ldots \longrightarrow t_n \longrightarrow \ldots$, we say that t is *strongly normalizing* or *terminating*. If for all $t \in \mathcal{A}$, t is weakly (resp. strongly) normalizing, then we say that $\longrightarrow$ is weakly (resp. strongly) normalizing.

A *local peak* is a triple of terms (s, u, t) such that $s \longleftarrow u \longrightarrow t$; u is the *source* and s, t are its *reducts*. Two terms s, t are *joinable*, written $s \downarrow t$, if and only if $s \longrightarrow\!\!\!\!\!\rightarrow v \twoheadleftarrow\!\!\!\!\!\leftarrow t$ for some v. If $\underset{a}{\longleftarrow}\underset{b}{\longrightarrow} \subseteq \underset{b}{\longrightarrow}^= {}^= \underset{a}{\longleftarrow}$ we say that $\underset{a}{\longrightarrow}$ and $\underset{b}{\longrightarrow}$ *sub-commute*. If $\longrightarrow$ sub-commutes with itself then it is said to have the *diamond property*. If $\longrightarrow\!\!\!\!\!\rightarrow_a$ and $\underset{b}{\longrightarrow\!\!\!\!\!\rightarrow}$ sub-commute, then we say that $\underset{a}{\longrightarrow}$ and $\underset{b}{\longrightarrow}$ *commute*. If $\twoheadleftarrow\!\!\!\!\!\rightarrow \subseteq \longrightarrow\!\!\!\!\!\rightarrow \twoheadleftarrow\!\!\!\!\!\leftarrow$ then we say that $\longrightarrow$ has the *Church-Rosser* property. If $\longrightarrow\!\!\!\!\!\rightarrow$ has the diamond property then $\longrightarrow$ is said to be *confluent* and if $\longleftarrow \longrightarrow \subseteq \longrightarrow\!\!\!\!\!\rightarrow \twoheadleftarrow\!\!\!\!\!\leftarrow$ we say that $\longrightarrow$ is *locally confluent*. A relation both confluent and strongly normalizing is *convergent*.

We assume well-known the following facts:

- Strong normalization implies weak normalization;
- The confluence and Church-Rosser properties are equivalent;

- Confluence implies the unicity of normal forms for every term;
- In confluent and weakly normalizing systems, (such as convergent systems), all $t \in \mathcal{A}$ have exactly one normal form written $t{\downarrow}$;
- If $\underset{a}{\longrightarrow} \subseteq \underset{b}{\longrightarrow} \subseteq \underset{a}{\longrightarrow\!\!\!\!\!\rightarrow}$ then $\underset{a}{\longrightarrow}$ is confluent if and only if $\underset{b}{\longrightarrow}$ is confluent;
- Local confluence and strong normalization imply confluence (Newman's Lemma, [New42]);
- If $\underset{a}{\longrightarrow}$ and $\underset{b}{\longrightarrow}$ commute and are both confluent, then $\underset{a}{\longrightarrow} \cup \underset{b}{\longrightarrow}$ is confluent (Hindley-Rosen, [Hin64, Ros73]);
- If $\underset{a}{\longleftarrow}\underset{b}{\longrightarrow} \subseteq \underset{b}{\longrightarrow}^{=}\underset{a}{\twoheadleftarrow}$ then $\underset{a}{\longrightarrow}$ and $\underset{b}{\longrightarrow}$ commute.

2.2.2 Positional term rewriting

We will consider in this manuscript the particular case of relations on the terms, $\mathcal{T}$ and meta-terms, $\mathcal{MT}$. These relations naturally extend to sets of terms, substitutes and substitutions:

Definition 2.2.1. *$\sigma \longrightarrow \tau$ iff $\mathcal{D}om(\sigma) = \mathcal{D}om(\tau)$ and $\forall X \in \mathcal{D}om(\sigma), \sigma(X) \longrightarrow \tau(X)$.*

Definition 2.2.2 (Monotonicity)**.** *A relation $\longrightarrow$ on terms is* monotonic *iff for all terms $s, t, u \in \mathcal{T}$ and position $p \in \mathcal{P}os(u)$, $s \longrightarrow t$ implies $u[s]_p \longrightarrow u[t]_p$.*

Monotonic relations on (meta-)terms will often be represented with arrow signs, $\longrightarrow$. Since monotonic relations have good properties with the positions of a term, they will often be decorated above by a position p, as in $\xrightarrow[\beta]{p}$, a set of positions P, as in $s \xrightarrow[R]{P} t$ or by a property that this position or set of positions satisfies, as in $u \xrightarrow[R]{\geq_{\mathcal{P}} p} v$. Mentioning a position in a rewrite step $u \xrightarrow[\beta]{p} v$ indicates that the reduction operates on the term u *at position* p and it is assumed that $p \in \mathcal{P}os(u)$.

Relations may also be annotated above with a *label*. When a relation is annotated with both a label and a (set of) position(s), we write the label first: $\xrightarrow{l,p}$. We allow to omit either or both if they are irrelevant or clear from context.

Before labeling steps with an exact position, it is possible to characterize, more generally, steps that occur *below* a position or parallel set of positions P.

Definition 2.2.3. *Given a term u such that $u \longrightarrow v$ and a parallel set $P \subseteq \mathcal{P}os(u)$. If $\overline{u}^P \longrightarrow \sigma$ and $v = \underline{u}_P\sigma$, we say that the rewrite step* occurs below *P and we write $u \xrightarrow{\geq_{\mathcal{P}} P} v$.*

Rewriting below a set of positions P leaves all positions above and incomparable untouched. This property will be particularly useful in the study of the *critical peak*.

Lemma 2.2.4. *If $u \xrightarrow{\geq_{\mathcal{P}} p} v$, then $q \leq_{\mathcal{P}} p \Rightarrow u|_q \longrightarrow v|_q$ and $q \# p \Rightarrow u|_q = v|_q$.*

Lemma 2.2.5. *If $p \leq_{\mathcal{P}} q$ then $\xrightarrow{\geq_{\mathcal{P}} q} \subseteq \xrightarrow{\geq_{\mathcal{P}} p} \subseteq \xrightarrow{\geq_{\mathcal{P}} \Lambda} = \longrightarrow$.*

Lemma 2.2.6. *Assume* $\underset{r}{\longrightarrow}$ *monotonic and* $s \overset{\geq_{\mathcal{P}} q}{\underset{r}{\longrightarrow}} t$. *Then* $\forall u \in \mathcal{T}, p \in \mathcal{P}os(u)$, $u[s]_p \overset{\geq_{\mathcal{P}} p \cdot q}{\underset{r}{\longrightarrow}} u[t]_p$.

Proof. Since $(u[s]_p)|_{p \cdot q} = s|_q$ and $u[s]_p = \underline{u}_p\{Z \mapsto \underline{\lambda}\overline{x}.s\}$. □

Definition 2.2.7 (Stability)**.** *A relation* $\underset{r}{\longrightarrow}$ *on terms is* stable *iff for all terms* $u, v \in \mathcal{T}$ *such that* $u \underset{r}{\longrightarrow} v$ *and for all substitution* σ, *we have* $u\sigma \underset{r}{\longrightarrow} v\sigma$.

Lemma 2.2.8. *If* $\underset{r}{\longrightarrow}$ *is stable, then so is* $\overset{\geq_{\mathcal{P}} p}{\underset{r}{\longrightarrow}}$.

In this thesis, we insist on using rewrite relations on terms that are both monotonic and stable. From here on, we will therefore reserve the arrow symbol for relations on terms that are proven to be both monotonic and stable. Besides, for all relations $\overset{P}{\underset{r}{\longrightarrow}}$ defined directly with a positional annotation, we write $\underset{r}{\longrightarrow} := \bigcup_P \overset{P}{\underset{r}{\longrightarrow}}$ and we check that $\overset{P}{\underset{r}{\longrightarrow}} \subseteq \overset{\geq_{\mathcal{P}} P}{\underset{r}{\longrightarrow}}$.

2.2.3 Functional rewriting

Two different kinds of reductions coexist, functional and higher-order reduction. Both are meant to operate on terms. However, rewriting meta-terms will sometimes be needed, in which case rewriting is intended to rewrite all their ground instances at once. *Functional reduction* is defined as follows

Definition 2.2.9 (β-reduction)**.** *The* β-reduction *at position* p, $\overset{p}{\underset{\beta}{\longrightarrow}}$, *is defined as the smallest monotonic relation on meta-terms such that* $(\lambda x : t.\, u)\ v \overset{\Lambda}{\underset{\beta}{\longrightarrow}} u\{x \mapsto v\}$.

Proof. To justify the use of the annotated arrow symbol, we need to prove that for all p, β-reduction at position p is stable (monotonicity is already assumed by definition).

By induction on p. If $u \overset{p}{\underset{\beta}{\longrightarrow}} v$ then, by definition, we have either:

- $p >_{\mathcal{P}} \Lambda$ and $u = u[s]_q \overset{p}{\longrightarrow} u[t]_q = v$ so that $s \overset{q^{-1}(p)}{\longrightarrow} t$ and, by induction hypothesis, $s\sigma \overset{q^{-1}(p)}{\longrightarrow} t\sigma$ so we can conclude $u\sigma = u\sigma[s\sigma]_q \overset{p}{\longrightarrow} u\sigma[t\sigma]_q = v\sigma$ by monotonicity;
- or $p = \Lambda$, $u = (\lambda x : t.\, s)\ w$, $v = s\{x \mapsto w\}$ and w.l.o.g. $x \notin \mathcal{D}om(\sigma) \cup \mathcal{R}an(\sigma)$. It is then clear, by definition, that $u\sigma \overset{\Lambda}{\longrightarrow} s\sigma\{x \mapsto w\sigma\} = v\sigma$. □

2.2.4 Higher-order reductions

The *root* of a meta-term refers to the root of the tree representing it. Since variables and symbols have arity 0 in our setting, they only occur at the root of themselves (seen as terms). We distinguish the root from the more interesting notion of *head* which refers to the first non-applicative node encountered while systematically searching the left branch of applications from the root.

Definition 2.2.10 (Head of a term)**.** *Assume a meta-term t. We define:*

- *its* head arity, $\mathbf{ha}(t)$*, the smallest natural number such that* $t(1^{\mathbf{ha}(t)}) \neq @$*;*
- *its* head position, $\mathbf{hp}(t) := 1^{\mathbf{ha}(t)}$*;*
- *its* head symbol *,* $\mathbf{hs}(t) := t(\mathbf{hp}(t)) \in \mathcal{F} \cup \mathcal{X} \cup \mathcal{Z} \cup \{\boldsymbol{\lambda}\}$*;*

For instance $t = \mathtt{f}\ x\ y\ z$ has a head arity of 3 and its head symbol is $\mathtt{f}$. Considering the head of a term plays a key role when defining efficient matching algorithms. The notion of head symbol is already required to define a safe notion of pattern.

Definition 2.2.11 (Pre-pattern)**.** *A* pre-redex *is a meta-term $Z[\overline{x}]$ which arguments $\overline{x}$ are pairwise distinct variables. A meta-term L is a* pre-pattern *if and only if its meta-variables occurrences are all pre-redexes.*

Example 1: The meta-terms $\mathtt{f}\ \lambda x : Y.\, X[x, x]$, $\mathtt{f}\ X[\mathtt{a}]$ and $\mathtt{f}\ X[Y]$ are no pre-patterns since one of their meta-variables is applied to meta-terms different from a variable or more than once to the same variable.

Pre-redexes in pre-patterns occur at parallel positions which set plays a key rôle.

Definition 2.2.12 (Fringe)**.** *The* fringe F_L *of a pre-pattern L is the set of parallel positions of its pre-redexes. We denote by $\mathcal{FP}os(L) = \{p \in \mathcal{P}os(L) \mid p \not\geq_{\mathcal{P}} F_L\}$ the set of* functional positions *of the pre-pattern L. We also define $F_\beta = \{11, 12, 2\}$ for convenience.*

Note that the functional positions coincide with the usual notion for first-order terms and corresponds to positions of potential overlap with other rules.

Lemma 2.2.13. *If L is a pre-pattern, σ a meta-substitution and $p \in \mathcal{P}os(L\sigma)$, then either $p \geq_{\mathcal{P}} F_L$ or $p \in \mathcal{FP}os(L)$ but not both.*

Closed pre-patterns, whose variables are all locally bounded, are a particularly convenient class of meta-terms to define unambiguous matching and unification:

Lemma 2.2.14. *Assume closed pre-patterns L and G and substitutions σ, τ.*

If $L\sigma = t$ then we say that t matches *L with σ. In that case there exists a unique γ such that $L\gamma = t$ and $\mathcal{D}om(\gamma) \subseteq \mathcal{MV}ar(L)$.*

If $L\gamma = G\gamma$ then we say that L and *G* are unifiable *and that γ is an* unifier*.*

If $L\sigma = G\tau$ and $\mathcal{MV}ar(L) \cap \mathcal{MV}ar(G) = \varnothing$, then there is a closed unifier γ of L and G such that $\mathcal{D}om(\gamma) \subseteq \mathcal{MV}ar(L) \cup \mathcal{MV}ar(G)$.

Proof. We can chose τ the restriction of σ to $\mathcal{MV}ar(L)$. Assuming $L\gamma = L\tau$ such that $\mathcal{D}om(\sigma) \subseteq \mathcal{MV}ar(L) \supseteq \mathcal{D}om(\tau)$ and $X \in \mathcal{MV}ar(L)$. Then $L|_p = X[\overline{x}]$ for some p. If $\sigma(X) = \underline{\lambda}\overline{z}.u$ and $\tau(X) = \underline{\lambda}\overline{z}.v$, we have $u\{\overline{z} \mapsto \overline{x}\} = L|_p = v\{\overline{z} \mapsto \overline{x}\}$ and since the $\overline{x}$ are distinct and locally bounded, we have $u = v = L|_p\{\overline{x} \mapsto \overline{z}\}$. Therefore $\gamma = \tau$.

Using the first point, we can assume σ and τ have disjoint domains and can be merged into $\gamma' := \tau \cup \sigma$. If $\mathcal{R}an(\gamma') = \overline{x}$, then we define $\gamma := \gamma'\{\overline{x} \mapsto \overline{X}\}$ which works. □

Computing a matching substitution for L and t (resp. a unifier for L and G) or deciding it does not exist can be done in polynomial time [FJ19a].

Note that subterms of a pre-pattern are pre-patterns and that if u is a term and $P \subseteq \mathcal{P}os(u)$ is parallel then $\underline{u}_P$ is a linear closed pre-pattern matching u with $\overline{u}^P$.

Closed pre-patterns are enough to define higher-order term rewriting.

Definition 2.2.15 (Rewriting)**.** *Assume L a closed pre-pattern and R such that $\mathcal{V}ar(R) \subseteq \mathcal{V}ar(L)$. We define* rewriting with (L, R) at position p *such that* $u \xrightarrow[(L,R)]{p} v$ *if and only if* $u|_p = L\gamma$ *and* $v = u[R\gamma]_p$ *for some substitution* γ*. The pair (L, R) is called a* rule*, a set of rules $\mathcal{R}$ is called a* system *and if L is linear then (L, R) and $\underset{(L,R)}{\longrightarrow}$ are called* left-linear*.*

We write $\xrightarrow[\mathcal{R}]{p} := \bigcup_{(L,R)\in\mathcal{R}} \xrightarrow[(L,R)]{p}$.

Closedness of both side and non-introduction of meta-variables are both required to ensure compatibility with the α-conversion.

To make our splitting notations fully explicit, if $u \xrightarrow[i]{p} v$, we have:

- $\underline{u}_p = u[X[\overline{x}]]_p$ and $\overline{u}^p = \{X \mapsto \lambda\overline{x}.u|_p\}$ with $\overline{x}$ variables bound above p in u
- $u = \underline{u}_p\overline{u}^p = \underline{u}_p\{X \mapsto \lambda\overline{x}.u|_p\} = \underline{u}_p\{X \mapsto \lambda\overline{x}.L\gamma\}$
- $v = \underline{u}_p\{X \mapsto \lambda\overline{x}.R\gamma\}$, hence $\underline{v}_p = \underline{u}_p$, $\overline{v}^p = \{X \mapsto \lambda\overline{x}.R\gamma\}$ and $v|_p = R\gamma$.

It is clear by definition that $\xrightarrow[(L,R)]{\geq_{\mathcal{P}} p} = \bigcup_{q\geq_{\mathcal{P}} p} \xrightarrow[(L,R)]{q}$ which justifies our positional annotation.

Lemma 2.2.16 (Monotonicity/Stability)**.** *Rewriting $\underset{(L,R)}{\longrightarrow}$ is the smallest stable monotonic relation on meta-terms such that $L \underset{(L,R)}{\longrightarrow} R$.*

Proof. Let $s \xrightarrow[(L,R)]{p} t$ and σ be a substitution. We have both $(s\sigma)|_p = s|_p\sigma = L\gamma\sigma$ and $t\sigma = s[R\gamma]_p\sigma = s\sigma[R\gamma\sigma]_p$, therefore $s\sigma \xrightarrow[(L,R)]{p} t\sigma$ and $\underset{(L,R)}{\longrightarrow}$ is stable. If $s \xrightarrow[(L,R)]{p} t$ then, by Lemma 2.1.25 and definition, $(u[s]_q)|_{q\cdot p} = s|_p = L\gamma$ and $u[t]_q = u[s[R\gamma]_p]_q = (u[s]_q)[R\gamma]_{q\cdot p}$ and therefore $\underset{(L,R)}{\longrightarrow}$ is monotonic. Finally it is easy to see that $L \xrightarrow[(L,R)]{\Lambda} R$ using the identity substitution for γ.

Let $\rightsquigarrow$ be the smallest stable monotonic relation on meta-terms such that $L \rightsquigarrow R$. We've already checked that check that $\rightsquigarrow \subseteq \underset{(L,R)}{\longrightarrow}$.

If $u \underset{(L,R)}{\longrightarrow} v$, then, by definition, $u|_p = L\gamma$ and $v = u[R\gamma]_p$ for some p. We have $L \rightsquigarrow R$ and, by stability, $L\gamma \rightsquigarrow R\gamma$ and, by monotonicity, $u = u[\,u|_p\,]_p = u[L\gamma]_p \rightsquigarrow u[R\gamma]_p = v$. □

Note that functional reduction can be seen as a particular rewriting.

Lemma 2.2.17. *Assuming* $L_\beta := (\lambda x\!:\!T.\,U[x])\ V$ *and* $R_\beta = U[V]$*, we have* $\xrightarrow[\beta]{p} = \xrightarrow[(L_\beta,R_\beta)]{p}$.

We can now define proper *higher-order reduction* which results from rewriting of *rules* which left-hand sides are higher-order patterns in Miller's or Nipkow's sense [MN98], although they need not be typed. In contrast with HORS, in our setting, the root of a term is often an application which is not considered a symbol. In order to define the proper notion of pattern, forbidding overlaps with the β rule, we need the *head term* of a pre-pattern to be a symbol.

Definition 2.2.18 (Rewriting)**.** *A* pattern *is a $\mathcal{F}$-headed β-normal closed pre-pattern which pre-redexes are unapplied. If L is a pattern then (L, R) is written $L \to R$ and is called a (higher-order)* rewrite rule*. Systems of rewrite rules are called* rewrite systems*. The relation $\xrightarrow[L\to R]{}$ is called higher-order rewriting.*

Example 2: The term $L = \mathtt{f}\ (\lambda x\!:\!\mathtt{N}.\,\lambda y\!:\!\mathtt{N}.\,\lambda z\!:\!\mathtt{N}.\,\mathtt{g}\ X[x,z]\ X[x,y])$ is a pattern. Its pre-redexes are the terms $X[x,z]$ and $X[x,y]$. Its fringe is the set $F_L = \{2^4 12, 2^4 2\}$.

The term $L = \mathtt{f}\ (\lambda x\!:\!\mathtt{N}.\,\lambda y\!:\!\mathtt{N}.\,\lambda z\!:\!\mathtt{N}.\,\mathtt{g}\ X[x,y,z])\ (\mathtt{a}\ X)$ is also a pattern, $F_L = \{12^5, 2^2\}$.

The meta-terms $\lambda x\!:\!\mathtt{N}.\,X$, $\mathtt{f}\ ((\lambda x\!:\!\mathtt{N}.\,x)\ \mathtt{a})$, $\mathtt{f}\ x$, $\mathtt{f}\ \lambda x\!:\!Y.\,X[x,x]$, $\mathtt{f}\ X[\mathtt{a}]$, $\mathtt{f}\ X[Y]$ and $\mathtt{f}\ (X\ Y)$ are no patterns. The first one is headed with an abstraction, the following one is not β-normal, the one after is not closed, the next three are not even pre-patterns and the last one contains an applied pre-redex.

All the conditions on patterns are required to prevent overlaps of rewrite rules with β-reduction. Left-hand sides not in β-normal forms contain a β-redex within. If applied, $\boldsymbol{\lambda}$-headed pre-pattern overlap below β at position $1 \not\geq_{\mathcal{P}} F_\beta$. Finally, applied meta-variables create β redexes if instantiated with abstractions such as $(\mathtt{f}\ (X\ \mathtt{t}))\{X \mapsto \lambda z\!:\!\mathtt{N}.\,z\}$.

This presentation allows to clearly separate the object language of terms (which has no meta-variables), from the meta-language of meta-terms (which has meta-variables). Rules, critical pairs and splittings belong to the meta-language, which serves analyzing the properties of the object language. For instance, rules are pairs of closed terms. They may contain meta-variables, but do not admit free variables. The role taken by variables in first-order rules is therefore taken here by meta-variables of arity zero.

In the following, $\mathcal{R}$ (resp. $\mathcal{R}$ll) is always a finite set of (resp. left-linear) rewrite rules.

Example 3: Let $L = \mathtt{der}\ (\lambda x\!:\!\mathtt{R}.\,\mathtt{times}\ A\ F[x]) \to \mathtt{times}\ A\ (\mathtt{der}\ \lambda y\!:\!\mathtt{R}.\,F[y]) = R$ be a rule. If we chose $\sigma = \{A \mapsto \mathtt{2}, F \mapsto \underline{\lambda}x.x\}$ then we have $L\sigma = \mathtt{der}\ (\lambda x\!:\!\mathtt{R}.\,\mathtt{times}\ \mathtt{2}\ x)$ and $R\sigma = \mathtt{times}\ \mathtt{2}\ (\mathtt{der}\ \lambda y\!:\!\mathtt{R}.\,y)$, hence $\mathtt{der}\ (\lambda x.\mathtt{times}\ \mathtt{2}\ x) \xrightarrow[L\to R]{\Lambda} \mathtt{times}\ \mathtt{2}\ (\mathtt{der}\ \lambda y\!:\!\mathtt{R}.\,y)$.

Note that, for instance, $\mathtt{der}\ (\lambda x\!:\!\mathtt{R}.\,\mathtt{times}\ x\ x)$ is a normal form.

Definition 2.2.19 (Head reduction)**.** *Assume $t \xrightarrow{p} u$. We say that t rewrite* at the head *to u if $p \leq_{\mathcal{P}} \mathbf{hp}(t)$. Otherwise we say that t rewrite* below the head *to u.*

For instance, assuming $\mathcal{R} = \{\mathtt{f}\ X \to X\}$, then $\mathtt{f}\ (\lambda x.x)\ \mathtt{c}\ x \xrightarrow[\mathcal{R}]{11} (\lambda x.x)\ \mathtt{c}\ x \xrightarrow[\beta]{1} \mathtt{c}\ x$ both rewrite steps occurring at the head.

Lemma 2.2.20. *If a term t rewrites to u below the head, then t and u have the same head symbol and arity:* $\mathbf{ha}(t) = \mathbf{ha}(u)$ *and* $\mathbf{hs}(t) = \mathbf{hs}(u)$.

If a term t β-rewrites at the head, then $\mathbf{hs}(t) = \lambda$ *and* $\mathbf{ha}(t) > 0$.

If a term t rewrites at the head with rule $l \to r$, then $\mathbf{hs}(t) = \mathbf{hs}(l)$ *and* $\mathbf{ha}(t) \geq \mathbf{ha}(l)$.

Lemma 2.2.21. *If $\longrightarrow$ is stable and monotonic and $\sigma \longrightarrow \tau$ then $u\sigma \longrightarrow\!\!\!\!\!\rightarrow u\tau$.*

Proof. By induction on u. The $u = x$ and $u = \mathtt{f}$ cases are both straightforward. If $u = u_1\ u_2$, $u = \lambda x : u_1.\, u_2$ or $u = X[\overline{u}]$ with $X \notin \mathcal{D}om(\sigma)$ then, by induction hypothesis, $\forall i, u_i\sigma \longrightarrow\!\!\!\!\!\rightarrow u_i\tau$ and we conclude by monotonicity. Otherwise $u = X[\overline{u}]$ with $X\sigma = \lambda\overline{x}.w$, $X\tau = \lambda\overline{x}.w'$ and either $w = w'$ or $w \longrightarrow w'$. By induction hypothesis, $\overline{u}\sigma \longrightarrow\!\!\!\!\!\rightarrow \overline{u}\tau$ and $u\sigma = w\{\overline{x} \mapsto \overline{u}\sigma\} \longrightarrow\!\!\!\!\!\rightarrow w\{\overline{x} \mapsto \overline{u}\tau\}$ (by monotonicity) $\longrightarrow w'\{\overline{x} \mapsto \overline{u}\tau\}$ (by stability) $= u\tau$. □

Corollary 2.2.21.1. *Let $u \underset{\mathcal{R}_1}{\longrightarrow\!\!\!\!\!\rightarrow} v$ and $\sigma \underset{\mathcal{R}_2}{\longrightarrow\!\!\!\!\!\rightarrow} \tau$. Then, $u\sigma \underset{\mathcal{R}_1 \cup \mathcal{R}_2}{\longrightarrow\!\!\!\!\!\rightarrow} v\tau$.*

Proof. We have both $u\sigma \underset{\mathcal{R}_2}{\longrightarrow\!\!\!\!\!\rightarrow} u\tau \underset{\mathcal{R}_1}{\longrightarrow\!\!\!\!\!\rightarrow} v\tau$ and $u\sigma \underset{\mathcal{R}_1}{\longrightarrow\!\!\!\!\!\rightarrow} v\sigma \underset{\mathcal{R}_2}{\longrightarrow\!\!\!\!\!\rightarrow} v\tau$ by stability. □

2.2.5 Parallel rewriting

We define *parallel rewriting* as the simultaneous rewriting at a parallel set of positions. Note that it is weaker than Tait's notion of parallel rewriting which we call here *orthogonal rewriting*.

Lemma 2.2.22. *If $p \# q$ then $\overset{p}{\longrightarrow}$ and $\overset{q}{\longrightarrow}$ commute:* $\overset{p}{\longrightarrow}\overset{q}{\longrightarrow} = \overset{q}{\longrightarrow}\overset{p}{\longrightarrow}$.

Proof. By definition since $u[R\gamma]_p|_q = u|_q$ and $u[R\gamma]_p[D\theta]_q = u[D\theta]_q[R\gamma]_p$. Both properties are proven by induction on u and definition of replacement. □

Definition 2.2.23. *Assume $P = \{p_1, \ldots, p_n\}$ a parallel set of positions, we define* parallel rewriting at positions P *as the relation* $\overset{P}{\Longrightarrow} := \overset{p_1}{\longrightarrow} \cdots \overset{p_n}{\longrightarrow}$.

Proof. By Lemma 2.2.22, $\overset{P}{\Longrightarrow}$ does not depend on the ordering of the elements of P. □

Lemma 2.2.24. *Parallel rewriting satisfies the following properties:*

- *Monotonicity: if $s \overset{P}{\Longrightarrow} t$ and $q \in \mathcal{P}os(u)$ then $u[s]_q \overset{q \cdot P}{\Longrightarrow} u[t]_q$.*
- *Stability: if $s \overset{P}{\Longrightarrow} t$ then $s\sigma \overset{P}{\Longrightarrow} t\sigma$.*
- $\overset{\{p\}}{\Longrightarrow} = \overset{p}{\longrightarrow}$.
- *If $u \overset{P}{\Longrightarrow} v$ then $\overline{u}^P \overset{\Lambda}{\longrightarrow} \sigma$ such that $v = \underline{u}_P\sigma$.*
- *If $P \# Q$ then* $\overset{P}{\Longrightarrow}\overset{Q}{\Longrightarrow} = \overset{Q}{\Longrightarrow}\overset{P}{\Longrightarrow} = \overset{P \uplus Q}{\Longrightarrow}$.

Proof. By definition and simple properties in Lemma 2.1.25. □

Parallel rewriting can be done simultaneously with a set $\mathcal{R}$ of rewrite rules written $\underset{\mathcal{R}}{\Longrightarrow}$. Note that $\bigcup_{r\in\mathcal{R}} \underset{r}{\Longrightarrow} \subseteq \underset{\mathcal{R}}{\Longrightarrow}$ but the latter may mix several rules of $\mathcal{R}$ in a single rewrite step.

Definition 2.2.25. *A set $V \subseteq \mathcal{X} \cup \mathcal{Z}$ is* self-nested *in u if there is $p >_\mathcal{P} q$ such that $u(p) \in V$ and $u(q) \in V$. In that case we say that u itself is also self-nested.*

Example 4: The set $\{X, Y\}$ is self-nested in $\mathtt{f}\ X[\mathtt{g}\ Y]$ but not in $\mathtt{f}\ Z[X, Y]\ X[Z]$.

Lemma 2.2.26. *If $\sigma \Longrightarrow \tau$ and $\mathcal{D}om(\sigma)$ is not self-nested in u then $u\sigma \Longrightarrow u\tau$.*

Proof. By induction on u, similar to Lemma 2.2.21. The $u = x \in \mathcal{X}$ case is straightforward. In the $u = u_1\ u_2$ and $u = \lambda x : u_1.\, u_2$ cases, the $u_i\sigma \Longrightarrow u_i\tau$ steps from induction hypothesis can be grouped together in a single $u\sigma \Longrightarrow u\tau$. This also works for the case of $u = X[\overline{u}]$ with $x \notin \mathcal{D}om(\sigma)$. If $u = X[\overline{u}]$ and $X \in \mathcal{D}om(\sigma)$ then we have $\sigma(X) = \underline{\lambda}\overline{z}.t$ for some $t \longrightarrow s$. Since $\mathcal{D}om(\sigma)$ is not self-nested in u, $\mathcal{D}om(\sigma)$ and $\mathcal{V}ar(u) \cup \mathcal{MV}ar(u)$ are disjoint and $\overline{u}\sigma = \overline{u}$. Therefore, by stability, $u\sigma = t\{\overline{z} \mapsto \overline{u}\} \Longrightarrow s\{\overline{z} \mapsto \overline{u}\} = u\tau$. □

Corollary 2.2.26.1. *If σ a term-substitution such that $\sigma \Longrightarrow \tau$ then $u\sigma \Longrightarrow u\tau$.*

If L is a pre-pattern and σ a meta-substitution such that $\sigma \Longrightarrow \tau$ then $L\sigma \Longrightarrow L\tau$.

2.3 The lambda-Pi-calculus modulo

The lambda-Pi-calculus modulo ($\lambda\Pi_\equiv$) is an extension of the Logical Framework ($\lambda\Pi$) [HHP93b]. In $\lambda\Pi_\equiv$, typing judgments are considered on a pre-defined *signature*, Σ, which not only introduces globally defined typed symbols but also extends the conversion rule with type-preserving higher-order rewrite rules. This means that the conversion rule is replaced with a convertibility modulo $\beta\mathcal{R}$ and the axiom rule may introduce symbols from the signature (cf Figure 2.1).

Several formalisms were suggested to define an extension of the lambda-Pi-calculus with rewrite rules from Cousineau and Dowek's $\lambda\Pi$-modulo [CD07] where rules are first order (in a higher-order setting) and defined with a context of types, to Blanqui's Reduction Type Systems [Bla01] adapted by Saillard [Sai15b] and in more recent work by Assaf [Ass15b]. We stick here to Assaf's presentation which is burdened with fewer syntactical restrictions on the considered terms. Our criteria for signature well-formedness are however closer to the ones introduced by Saillard. Whenever our choices significantly vary from other presentations, we will be careful to point it out. The many subtleties concealed in these variants are very well covered in the in-depth work of Saillard which we strongly recommend.

The expressiveness of rewriting makes this extended logical framework ideal to encode other logics. An encoding of a system in $\lambda\Pi_\equiv$ is nothing more than a well-formed signature Σ. Whenever all rewrite rules in a signature Σ are type preserving and the corresponding rewrite system is confluent, then type checking in the signature Σ is decidable. In

particular, the Dedukti software, developed in the Deducteam (Inria) implements a type checking algorithm for this type system.

2.3.1 Typing in the Logical Framework

Definition 2.3.1 ($\lambda\Pi_{\equiv}$ Syntax)**.** *The syntax of* $\lambda\Pi_{\equiv}$ *is as follows:*

$$\begin{array}{rrcl}
\textit{(Variable)} & x, y & \in & \mathcal{X} \cup \mathcal{Z} \\
\textit{(Symbol)} & \mathtt{c} & \in & \mathcal{F} \\
\textit{(Sort)} & s & \in & \{*, \Box\} \\
\textit{(Term)} & t, u, A, l, r & := & x \mid s \mid t\ u \mid \lambda x : A.\, t \mid \Pi x : A.\, t \\
\textit{(Context)} & \Gamma & := & \varnothing \mid \Gamma, x : A \\
\textit{(Signature)} & \Sigma & := & \varnothing \mid \Sigma, \mathtt{c} : A \mid \Sigma, l \to r \\
\textit{(Typing Judgment)} & & := & \Sigma; \Gamma \vdash_{\mathcal{D}} t : A \\
\textit{(Context WF Judgment)} & & := & \Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}} \\
\textit{(Signature WF Judgment)} & & := & \Sigma\ \mathbf{WF}_{\mathcal{D}}
\end{array}$$

Judgments are identified with the property that they are derivable using the corresponding set of inference rules. The signature Σ can often be inferred from context in which case it may be omitted.

The list of rewrite rules $l \to r$ in a signature Σ is called its *rewrite system*, written $\mathcal{R}_\Sigma$ or $\mathcal{R}$ if Σ can be inferred. The list of all symbol declarations $(\mathtt{c} : A) \in \Gamma$ is called its *signature*, written $\mathcal{S}_\Sigma$ or $\mathcal{S}$.

For this system and all other featuring a dependent product, we allow the notations $A \to B := \Pi x : A.\, B$ if $x \notin \mathcal{V}ar(B)$ and $\Pi x_1\ \ldots\ x_n : A.\, B := \Pi x_1 : A. \cdots \Pi x_n : A.\, B$. Note that dependent product types was not syntactically incorporated in our definition of terms. It can either be seen as an applied special symbol, $\Pi x : A.\, t := \mathtt{pi}\ \lambda x : A.\, t$ or as an extra symbol of arity 2, like $\boldsymbol{\lambda}$ or @, $\Pi x : A.\, t := \boldsymbol{\pi}(A, (x)t)$. In either case, the symbol $\boldsymbol{\pi}$ representing product types may not be the head of rewrite-rules left-hand sides.

Definition 2.3.2 (Conversion)**.** *Assume* Σ *a signature and* $\mathcal{R}$ *the set of rewrite rules in* Σ*. Then two terms* t *and* u *are* convertible*, written* $t \equiv_{\beta\mathcal{R}} u$*, if* $t \underset{\beta\mathcal{R}}{\longleftrightarrow\!\!\!\!\twoheadrightarrow} u$*.*

We avoid the notation $\equiv_{\beta\Sigma}$ to emphasize the fact that only rewrite rules from the signature define the conversion, as opposed to other relations, such as η, where typing and conversion are somewhat intertwined.

Definition 2.3.3 (Well-Typed Term)**.** *A term* t has type A *in the signature* Σ *and context* Γ *if the judgment* $\Sigma; \Gamma \vdash_{\mathcal{D}} t : A$ *is derivable using the inference rules of Figure 2.1.*

Definition 2.3.4 (Well-Formed Context)**.** *A context* Γ *is* well-formed *with respect to a signature* Σ *if the judgment* $\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$ *is derivable by the inference rules of Figure 2.2.*

For a given signature Ω, for instance encoding another system, we define the system $\mathcal{D}[\Omega]$ as exactly $\mathcal{D}$ with a prefix signature Ω so that we have:

$$\begin{array}{rcl}
\Sigma; \Gamma \vdash_{\mathcal{D}[\Omega]} t : A & \iff & \Omega, \Sigma; \Gamma \vdash_{\mathcal{D}} t : A \\
\Sigma \vdash_{\mathcal{D}[\Omega]} \Gamma\ \mathbf{WF}_{\mathcal{D}[\Omega]} & \iff & \Omega, \Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}} \\
\Sigma\ \mathbf{WF}_{\mathcal{D}[\Omega]} & \iff & \Omega, \Sigma\ \mathbf{WF}_{\mathcal{D}}
\end{array}$$

$$\frac{}{\Sigma;\Gamma \vdash_{\mathcal{D}} * : \Box}\ \mathcal{P}_{*} \qquad \frac{(\mathsf{c} : A) \in \Sigma}{\Sigma;\Gamma \vdash_{\mathcal{D}} \mathsf{c} : A}\ \mathcal{P}_{\Sigma} \qquad \frac{(x : A) \in \Gamma}{\Sigma;\Gamma \vdash_{\mathcal{D}} x : A}\ \mathcal{P}_{\Gamma}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathcal{D}} A : * \qquad \Sigma;\Gamma, x : A \vdash_{\mathcal{D}} B : s \qquad s \in \{*, \Box\}}{\Sigma;\Gamma \vdash_{\mathcal{D}} \Pi x : A.\, B : s}\ \mathcal{P}_{\Pi}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathcal{D}} \Pi x : A.\, B : s \qquad \Sigma;\Gamma, x : A \vdash_{\mathcal{D}} t : B \qquad s \in \{*, \Box\}}{\Sigma;\Gamma \vdash_{\mathcal{D}} \lambda x : A.\, t : \Pi x : A.\, B}\ \mathcal{P}_{\lambda}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathcal{D}} t : \Pi x : A.\, B \qquad \Sigma;\Gamma \vdash_{\mathcal{D}} u : A}{\Sigma;\Gamma \vdash_{\mathcal{D}} t\ u : B\{x \mapsto u\}}\ \mathcal{P}_{@}$$

$$\frac{\Sigma;\Gamma \vdash_{\mathcal{D}} t : A \qquad \Sigma;\Gamma \vdash_{\mathcal{D}} B : s \qquad s \in \{*, \Box\} \qquad A \equiv_{\beta\mathcal{R}} B}{\Sigma;\Gamma \vdash_{\mathcal{D}} t : B}\ \mathcal{P}_{\equiv}$$

Figure 2.1: Inference rules for typing in $\lambda\Pi_{\equiv}$

$$\frac{}{\Sigma \vdash_{\mathcal{D}} \varnothing\ \mathbf{WF}_{\mathcal{D}}}\ \mathbf{WF}_{\varnothing} \qquad \frac{\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}} \qquad \Sigma;\Gamma \vdash_{\mathcal{D}} A : * \qquad x \notin \Gamma}{\Sigma \vdash_{\mathcal{D}} \Gamma, x : A\ \mathbf{WF}_{\mathcal{D}}}\ \mathbf{WF}_{\mathcal{X}}$$

Figure 2.2: Inference rules for context well-formedness in $\lambda\Pi_{\equiv}$

Lemma 2.3.5 (Weakening)**.** *The following holds for all* Σ, Σ', Γ, Γ', t *and* A.
If $\Sigma;\Gamma \vdash_{\mathcal{D}} t : A$ *then* $\Sigma;\Gamma,\Gamma' \vdash_{\mathcal{D}} t : A$.
If $\Sigma;\Gamma \vdash_{\mathcal{D}} t : A$ *then* $\Sigma,\Sigma';\Gamma \vdash_{\mathcal{D}} t : A$.
If $\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$ *then* $\Sigma,\Sigma' \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$.

Proof. By simple inductions on the derivation. □

Lemma 2.3.6 (Substitution)**.** *Assume* $\Sigma;\Gamma \vdash_{\mathcal{D}} t : A$ *and* $\Sigma \vdash_{\mathcal{D}} \Gamma, x : A, \Gamma'\ \mathbf{WF}_{\mathcal{D}}$.
Then $\Sigma \vdash_{\mathcal{D}} \Gamma, \Gamma'\{x \mapsto t\}\ \mathbf{WF}_{\mathcal{D}}$ *and for all* u, B *such that* $\Sigma;\Gamma, x : A, \Gamma' \vdash_{\mathcal{D}} u : B$, *we also have* $\Sigma;\Gamma, \Gamma'\{x \mapsto t\} \vdash_{\mathcal{D}} u\{x \mapsto t\} : B\{x \mapsto t\}$.

Proof. We first prove the typing of $u\{x \mapsto t\}$ by induction on typing the derivation of u then deduce the well-formedness of $\Gamma, \Gamma'\{x \mapsto t\}$ by induction on the length of Γ'. The typing derivation of t provides, together with the weakening lemma, the base case $\mathcal{P}_{\Gamma}$ for the introduction of the x variable. The introduction of a variable $(y : B) \in \Gamma$ is straightforward since $\Sigma \vdash_{\mathcal{D}} \Gamma, x : A, \Gamma'\ \mathbf{WF}_{\mathcal{D}}$ ensures that $x \notin \mathcal{FV}ar(B)$. All other cases follow by induction hypothesis. □

2.3.2 Checking rules and signatures

Definition 2.3.7 (Product Compatibility)**.** *A signature* Σ *satisfies the* product compatibility *property, written* $\mathbf{PC}(\Sigma)$*, if for any well-formed context* Γ *and derivable judgments* $\Sigma;\Gamma \vdash_{\mathcal{D}} \Pi x : A_1.\, B_1 : s$ *and* $\Sigma;\Gamma \vdash_{\mathcal{D}} \Pi x : A_2.\, B_2 : s$ *such that* $\Pi x : A_1.\, B_1 \equiv_{\beta\mathcal{R}} \Pi x : A_2.\, B_2$*, we have* $A_1 \equiv_{\beta\mathcal{R}} A_2$ *and* $B_1 \equiv_{\beta\mathcal{R}} B_2$*.*

Lemma 2.3.8. *If* $\Pi x : A.\, B \underset{\beta\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow} t$*, then* $t = \Pi x : A'.\, B'$ *such that* $A \underset{\beta\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow} A'$ *and* $B \underset{\beta\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow} B'$*.*

Proof. By induction on the length of the reduction. Since no rewrite rule matches a product at the head, reduction steps occur in either its domain or its codomain. □

Lemma 2.3.9. *If* $\underset{\beta}{\longrightarrow} \cup \underset{\Sigma}{\longrightarrow}$ *is confluent, then* $\mathbf{PC}(\Sigma)$*.*

Proof. We assume confluence of $\underset{\beta}{\longrightarrow} \cup \underset{\Sigma}{\longrightarrow}$. Assuming $\Pi x : A.\, B \equiv_{\beta\mathcal{R}} \Pi x : C.\, D$, then $\Pi x : A.\, B \underset{\beta\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow} t \underset{\beta\mathcal{R}}{\longleftarrow\!\!\!\!\!\leftarrow} \Pi x : A'.\, B'$. By Lemma 2.3.8, $t = \Pi x : C.\, D$ such that $A \underset{\beta\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow} C \underset{\beta\mathcal{R}}{\longleftarrow\!\!\!\!\!\leftarrow} A'$ and $B \underset{\beta\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow} D \underset{\beta\mathcal{R}}{\longleftarrow\!\!\!\!\!\leftarrow} B'$ which allows to conclude. □

Definition 2.3.10 (Type Preservation)**.** *A rewrite rule* $l \to r$ *is* type preserving *or* well-typed *in a signature* Σ*, written* $\Sigma \Vdash_{\mathcal{D}} l \to r$*, if, for any valuation* σ*, any context* Γ *well-formed in* Σ *and any term* T*, if* $\Sigma;\Gamma \vdash_{\mathcal{D}} l\sigma : T$*, then* $\Sigma;\Gamma \vdash_{\mathcal{D}} r\sigma : T$*.*

Definition 2.3.11 (Well-Formed Signature)**.** *A signature* Σ *is* well-formed*,* $\mathbf{WF}(\Sigma)$*, iff*

- $\mathbf{PC}(\Sigma)$
- *for all* $\mathsf{c} : A \in \Sigma$*,* $\Sigma;\varnothing \vdash_{\mathcal{D}} A : s$ *for* $s \in \{*, \Box\}$
- *for all* $l \to r \in \Sigma$*,* $\Sigma \Vdash_{\mathcal{D}} l \to r$

Definition 2.3.12 (Subject reduction)**.** *A well-formed signature* Σ *has the* subject reduction *property, written* $\mathbf{SR}(\Sigma)$*, iff for all terms* u*,* v*,* A *and context* Γ *if* $u \underset{\beta\Sigma}{\longrightarrow} v$ *and* $\Sigma,\Gamma \vdash_{\mathcal{D}} u : A$ *then* $\Sigma,\Gamma \vdash_{\mathcal{D}} v : A$*.*

Definition 2.3.13 (Uniqueness of Types)**.** *A well-formed signature* Σ *has the* uniqueness of types *property, written* $\mathbf{UT}(\Sigma)$*, iff for all terms* u*,* A*,* B *and context* Γ *if* $\Sigma,\Gamma \vdash_{\mathcal{D}} u : A$ *and* $\Sigma,\Gamma \vdash_{\mathcal{D}} u : B$ *then* $A \equiv_{\beta\mathcal{R}} B$*.*

Theorem 2.3.14. *For all signature* Σ*, if* $\mathbf{WF}(\Sigma)$ *then* $\mathbf{UT}(\Sigma)$ *and* $\mathbf{SR}(\Sigma)$*.*

Lemma 2.3.15 (Decidability of Type-Checking)**.** *Assume* Σ *is a signature which set* $\mathcal{R}$ *of rewrite rules is such that* $\underset{\beta\mathcal{R}}{\longrightarrow}$ *is confluent and strongly normalizing when restricted to the subset of terms which are well-typed in a context well-formed in* Σ*. Then type checking is decidable.*

In practice, confluence is often required to prove the strongly-normalizing condition that allows type checking to be decidable.

Lemma 2.3.16 (Inversion)**.** *If* $\mathbf{WF}(\Sigma)$, $\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$ *and* $\Sigma;\Gamma \vdash_{\mathcal{D}} t : T$*, then:*

$$
\begin{array}{lll}
t = x \in \mathcal{X} & \Rightarrow \exists A. & (x : A) \in \Gamma \wedge A \equiv_{\beta\mathcal{R}} T \\
t = \mathsf{c} \in \mathcal{F} & \Rightarrow \exists A. & (\mathsf{c} : A) \in \Sigma \wedge A \equiv_{\beta\mathcal{R}} T \\
t = s \in \{*, \Box\} & \Rightarrow & s = * \wedge T \equiv_{\beta\mathcal{R}} \Box \\
t = \Pi x : A.\, B & \Rightarrow \exists s. & \Sigma;\Gamma \vdash_{\mathcal{D}} A : * \wedge \Sigma;\Gamma, x : A \vdash_{\mathcal{D}} B : s \wedge T \equiv_{\beta\mathcal{R}} s \\
t = \lambda x : A.\, t & \Rightarrow \exists B, s. & \Sigma;\Gamma, x : A \vdash_{\mathcal{D}} t : B \wedge \Sigma;\Gamma \vdash_{\mathcal{D}} \Pi x : A.\, B : s \wedge T \equiv_{\beta\mathcal{R}} \Pi x : A.\, B \\
t = u\ v & \Rightarrow \exists A, B. & \Sigma;\Gamma \vdash_{\mathcal{D}} u : \Pi x : A.\, B \wedge \Sigma;\Gamma \vdash_{\mathcal{D}} v : A \wedge T \equiv_{\beta\mathcal{R}} B\{x \mapsto v\}
\end{array}
$$

Lemma 2.3.17 (Subterm)**.** *Assume* Σ *well-formed,* $\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$, $\Sigma;\Gamma \vdash_{\mathcal{D}} t : A$ *and* $u \lhd t$*. Then* $\Sigma;\Gamma,\Theta \vdash_{\mathcal{D}} u : B$ *for some context extension* Γ,Θ *well-formed in* Σ*.*

Proof. By induction on t using Lemma 2.3.16. □

Our presentation differs from the usual presentation of the $\lambda\Pi_{\equiv}$ as it allows terms to be well-typed in an ill-typed context or signature. Considering typing judgment with an ill-typed context or signature is not usually useful however removing this constraint defines a system closer to the implementation, where contexts are *assumed* and *maintained* well-typed rather than *checked* at every occurrence of a variable.

Definition 2.3.18. *We define* $\lambda\Pi^{\mathrm{WF}}_{\equiv}$ *the variant of* $\lambda\Pi_{\equiv}$ *where the* $\mathcal{P}_*$, $\mathcal{P}_\Sigma$ *and* $\mathcal{P}_\Gamma$ *rules have the extra* $\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$ *premise and the* $\mathbf{WF}_\varnothing$ *rule has the extra* $\mathbf{WF}(\Sigma)$ *premise.*

Judgments in this system are written with $\vdash^{\mathrm{WF}}_{\mathcal{D}}$*.*

Lemma 2.3.19. $\Sigma \vdash^{\mathrm{WF}}_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$ *if and only if* $\mathbf{WF}(\Sigma)$ *and* $\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$*.*

$\Sigma;\Gamma \vdash^{\mathrm{WF}}_{\mathcal{D}} t : A$ *if and only if* $\mathbf{WF}(\Sigma)$, $\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$ *and* $\Sigma;\Gamma \vdash_{\mathcal{D}} t : A$*.*

Proof. $\Rightarrow$ is done by induction on the derivation of $\Sigma \vdash^{\mathrm{WF}}_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$. The base case is covered with the extra premise in $\mathbf{WF}_\varnothing$. $\Leftarrow$ is done by induction on the derivation of $\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$ using $\mathbf{WF}(\Sigma)$ for base case.

Both sides are again done by induction on the typing derivation. $\Rightarrow$ uses either induction hypothesis on typing premise or the extra premise together with previous point. □

We reuse the same notation for all other type systems featuring typing and well-formedness judgments.

2.3.3 Checking signature well-formedness

The type preservation property of a rewrite rule depends on the signature Σ in which this rule is considered. In order to check the well-formedness of a extended signature it is therefore not only necessary to check the added rules and declarations but also the recheck all previously introduced rewrite rules. This definition is therefore highly unpractical and calls for a stronger but more stable criteria to use in practice.

Definition 2.3.20 (Permanent Type Preservation)**.** *A rewrite rule* $l \to r$ *is* permanently well-typed *in a signature* Σ*, written* $\Sigma \vdash_{\mathcal{D}} l \to r$*, if it is well-typed for any signature extension that satisfies product compatibility:* $\forall \Sigma', \mathbf{PC}(\Sigma, \Sigma') \implies \Sigma, \Sigma' \vdash_{\mathcal{D}} l \to r$*.*

Note that permanently well-typed rules are also well-typed in the same signature if it satisfies product compatibility.

Lemma 2.3.21. *Both* $\Sigma; \Gamma \vdash_{\mathcal{D}} t : A$ *and* $\Sigma \vdash_{\mathcal{D}} l \to r$*, seen as properties of the signature* Σ*, are stable by signature extension.*

Proof. By definition for $\Sigma \vdash_{\mathcal{D}} l \to r$. By induction on the derivation for $\Sigma; \Gamma \vdash_{\mathcal{D}} t : A$. A signature extension extends the conversion relation, by adding new rules, and therefore extends the well-typedness relation, by extending the $\mathcal{P}_{\equiv}$ and $\mathcal{P}_{\Sigma}$ inference rules. □

Note however that the type preservation property, $\Sigma \vdash_{\mathcal{D}} l \to r$, is not preserved since the assumption that the left-hand side is well-typed gets weaker as the typing relation is extended.

Example 1: Consider the signature $\Sigma := \{\mathtt{A} : *, \mathtt{B} : *, \mathtt{a} : \mathtt{A}, \mathtt{f} : \mathtt{B} \to *, \mathtt{f}\ a \longrightarrow a\}$. No instance of the rewrite rule's left-hand side is well-typed hence the rule itself is well-typed and Σ is well-formed. However, the signature extension $\Sigma, \mathtt{B} \to \mathtt{A}$ is ill-formed. This ill-formedness is not due to the added rewrite rule, which is well-typed. It rather comes from the previously defined rule which becomes ill-typed by adding the new rule since its left-hand side now matches some well-typed terms.

$$\frac{}{\varnothing\ \mathbf{WF}_{\mathcal{D}}}\ \mathbf{WF}_{\varnothing} \qquad \frac{\Sigma\ \mathbf{WF}_{\mathcal{D}} \qquad \Sigma; \varnothing \vdash_{\mathcal{D}} A : s \qquad s \in \{*, \Box\} \qquad \mathtt{c} \notin \Sigma}{\Sigma, \mathtt{c} : A\ \mathbf{WF}_{\mathcal{D}}}\ \mathbf{WF}_{\mathcal{F}}$$

$$\frac{\Sigma\ \mathbf{WF}_{\mathcal{D}} \qquad \underset{\beta\Sigma\Xi}{\longrightarrow} \text{ confluent} \qquad \Xi = (l_i \to r_i)_i \qquad \forall i,\ \Sigma \vdash_{\mathcal{D}} l_i \to r_i}{\Sigma, \Xi\ \mathbf{WF}_{\mathcal{D}}}\ \mathbf{WF}_{\mathcal{R}}$$

Figure 2.3: Typing rules for signature strong well-formedness

Definition 2.3.22 (Strong Well-formedness)**.** *If* $\Sigma\ \mathbf{WF}_{\mathcal{D}}$ *can be derived using the rules of Figure 2.3, then* Σ *is said to be* strongly well-formed*.*

Lemma 2.3.23. *If* Σ *is strongly well-formed, then it is* well-formed*.*

Proof. Assume $\Sigma\ \mathbf{WF}_{\mathcal{D}}$ and $\mathcal{R}$ is the set of rewrite rules in Σ. Then $\underset{\beta\mathcal{R}}{\longrightarrow}$ is confluent and by Lemma 2.3.9, $\mathbf{PC}(\Sigma)$. By induction on the derivation of $\Sigma\ \mathbf{WF}_{\mathcal{D}}$, for all $\mathtt{c} : A \in \Sigma$ (resp. $l \to r \in \Sigma$), $\Delta; \varnothing \vdash_{\mathcal{D}} A : s$ for some $s \in \{*, \Box\}$ (resp. $\Delta \vdash_{\mathcal{D}} l \to r$) for some Δ prefix of Σ. By Lemma 2.3.21, we have $\Sigma; \varnothing \vdash_{\mathcal{D}} A : s$ (resp. $\Sigma \vdash_{\mathcal{D}} l \to r$ and therefore $\Sigma \vdash_{\mathcal{D}} l \to r$). □

Dedukti relies on these inference rules to check the well-formedness of signatures. This criterion is an approximation of the signature well-formedness property. It is further

approximated by the fact that rule (strong) well-formedness is undecidable and can only automatically be checked using an incomplete criteria.

The study of the (strong) well-typedness property of higher-order rewrite rules is a whole research area which we cannot describe in detail here. In Saillard's [Sai15b] and Assaf's [Ass15b] presentations, rewriting is defined using HORS which relies on a "meta" λ-calculus of substitution where meta-variables are treated quite similarly to usual variables. It is tempting, in that setting, to infer and assign (dependent product) types to the meta-variables of a rewrite rule, using the fact that we are considering exclusively well-typed instances of the left-hand sides. An occurrence $X[x, y]$ in the left-hand side of a rewrite rule can be seen as the application of X to locally bounded variables x and y, yielding a term $X\ x\ y$ which allow to infer a type for X from the declared types of x and y and the expected type of the pre-redex. These typing assumption could then be used to ensure the well-typedness of occurrences of these variables in the right-hand side, just like we would do for the first order case where matching variables from rewrite rules are little more than simple variables.

Lemma 2.3.24. *Let Σ be a signature such that $\xrightarrow[\beta\Sigma]{}$ is confluent and $l \to r$ a rewrite rule. If Δ is a context such that $\mathcal{D}om(\Delta) = \mathcal{MV}ar(l)$, $\Sigma \vdash_{\mathcal{D}} \Delta\ \mathbf{WF}_{\mathcal{D}}$, $\Sigma; \Delta \vdash_{\mathcal{D}} l : A$ and $\Sigma; \Delta \vdash_{\mathcal{D}} r : A$, then $\Sigma \vdash_{\mathcal{D}} l \to r$.*

This easy criteria is already enough to check the well-formedness of many signatures, including the encoding of most type systems. Saillard's implementation already offers a more complete criteria relying on *constraints* inferred from the well-typedness of left-hand sides and used to type-check the right-hand side while still heavily relying on the typing of meta-variables.

This criteria is far from complete for several reasons:

- First of all, the order of the arguments of a meta-variable is not supposed to be relevant, $X[x, y]$ and $X[y, x]$ are two equivalent pre-redex matching the same terms, yet one of them may be typable while the other is not.
- Second of all, substitutes are $\underline{\lambda}$-meta-terms, not λ-terms. In particular their abstractions are not annotated. It could be tempting, for instance, to allow partially applied meta-variables, either on the RHS or the (non-linear) LHS: $\mathtt{f}\ (\lambda x.\, X[x])\ X \to X$. However these extensions do not behave well with, respectively, the logic of $\lambda\Pi_{\equiv}$ and the meta theory of term rewriting.
- Finally, we need to consider the cases where a meta-variable is not applied to all the available locally bounded variables. In a local context, inside a pattern, where both x and y are locally bounded, the partially applied $F[y]$ is a valid pre-redex. However, because of dependent types, the type of y and $F[y]$ may both very well depend on x and therefore it is impossible to provide a type for F.

In most cases, however, meta-variables are used in their fully applied version and variables can always be assumed ordered by depth making the first and third point.

In all generality the correct way to assign types to meta-variable is to consider substi-

tutes: $X : (\underline{\lambda}\overline{x}.T)$ rather than (dependent) product types.

Example 2: Consider the following signature.

$$\Sigma := \begin{cases} \mathtt{A} : * \quad , \quad \mathtt{T} : \mathtt{A} \to \mathtt{A} \to * \quad , \quad \mathtt{a} : \mathtt{A} \quad , \quad \mathtt{b} : \mathtt{A} \quad , \\ \mathtt{f} : \Pi a : \mathtt{A}. (\Pi x : \mathtt{A}. \mathtt{T}\ a\ x \to \mathtt{T}\ x\ a) \to \mathtt{T}\ \mathtt{a}\ \mathtt{b} \to \mathtt{T}\ \mathtt{b}\ \mathtt{a} \\ \mathtt{f}\ \mathtt{a}\ (\lambda x : \mathtt{A}. \lambda y : \mathtt{T}\ a\ x. Z[x, y])\ T \quad \longrightarrow \quad Z[\mathtt{b}, T] \\ \mathtt{f}\ \mathtt{a}\ (\lambda x : X. \lambda y : Y[x]. Z[y] \quad)\ T \quad \longrightarrow \quad Z[T] \end{cases}$$

The first rewrite rule is well-typed by Lemma 2.3.24 using the context of meta-variables $\Delta := \{Z : \Pi x : \mathtt{A} : \mathtt{T}\ \mathtt{a}\ x \to \mathtt{T}\ x\ \mathtt{a}.\ ,\ T : \mathtt{T}\ \mathtt{a}\ \mathtt{b}\}$ well-typed in Σ and in which both sides of the rule are well-typed.

The second rule is trickier as the left-hand side is ill-typed in all context of meta-variables. We consider a well-typed instance of the left-hand side with valuation σ in a context Γ and signature extension Σ'. By inversion (Lemma 2.3.16) we are able to deduce the following typing relation in signature Σ' and context Γ:

$$\begin{array}{llll} \vdash_{\mathcal{D}} T\sigma : \mathtt{T}\ \mathtt{a}\ \mathtt{b} & x : X\sigma \vdash_{\mathcal{D}} Y[x]\sigma : * & X\sigma \equiv_{\beta\mathcal{R}} \mathtt{A} \\ \vdash_{\mathcal{D}} X\sigma : * & x : X\sigma, y : Y[x]\sigma \vdash_{\mathcal{D}} Z[y]\sigma : \mathtt{T}\ x\ \mathtt{a} & Y[x]\sigma \equiv_{\beta\mathcal{R}} \mathtt{T}\ \mathtt{a}\ x \end{array}$$

Using the conversion properties together with the substitution lemma 2.3.6 on the last typing condition, we get $y : \mathtt{T}\ \mathtt{a}\ \mathtt{b} \vdash_{\mathcal{D}} Z[y]\sigma : \mathtt{T}\ \mathtt{b}\ \mathtt{a}$ and then $\vdash_{\mathcal{D}} Z[T]\sigma : \mathtt{T}\ \mathtt{b}\ \mathtt{a}$ which is the same type as the left-hand side, therefore the rule is permanently type-preserving.

We implemented an extension of the existing criteria with Genestier (unpublished) in order to check the well-formedness of the encodings of Coq (this manuscript) and Agda for which the usual criteria was not powerful enough. All the above criteria are subsumed by the more general criteria found in the work of Blanqui [Bla20] but not yet implemented in Dedukti. The type preservation property of all the $\lambda\Pi_{\equiv}$ signatures considered from here on have been checked using the Dedukti tool and the proof of this property will therefore be left to the reader. While Lemma 2.3.24 is often too restrictive to apply, the proof using definition is seldom more complicated than Example 2.

2.3.4 Syntactical Stratification

Lemma 2.3.25 (Stratification)**.** *Assume* $\Sigma; \Gamma \vdash^{\mathrm{WF}}_{\mathcal{D}} t : A$*. Then either*

- $\Sigma; \Gamma \vdash_{\mathcal{D}} A : *$*, in which case we say that* t *is an* object*,*
- $\Sigma; \Gamma \vdash_{\mathcal{D}} A : \Box$*, in which case we say that* t *is a* type*,*
- $A = \Box$*, in which case we say that* t *is a* kind*.*

Proof. By induction on $\Sigma; \Gamma \vdash_{\mathcal{D}} t : A$.

- $\mathcal{P}_*$: $A = \Box$ and t is a kind.
- $\mathcal{P}_\Sigma$: $\Sigma = \Sigma_1; x : A; \Sigma_2$. Since $\mathbf{WF}(\Sigma)$, either $\Sigma_1; \varnothing \vdash_{\mathcal{D}} A : *$ or $\Sigma_1 k; \varnothing \vdash_{\mathcal{D}} A : \Box$. By weakening, t is either an object or a type.
- $\mathcal{P}_\Gamma$: Since $\Sigma \vdash_{\mathcal{D}} \Gamma\ \mathbf{WF}_{\mathcal{D}}$, by weakening, t is an object.

- $\mathcal{P}_\Pi$: Either $A = \Box$ and t is a kind or $A = *$ and t is a type.
- $\mathcal{P}_\lambda$, $\mathcal{P}_\equiv$: Either $\vdash_\mathcal{D} A : \Box$ and t is a type or $\vdash_\mathcal{D} A : *$ and t is an object.
- $\mathcal{P}_@$: By Lemma 2.3.16 (inversion), $\Sigma; \Gamma, x : A \vdash_\mathcal{D} B : s$ and by Lemma 2.3.6 (substitution), $\Sigma; \Gamma \vdash_\mathcal{D} B\{x \mapsto t\} : s \in \{*, \Box\}$ and t is either an object or a type. □

This classification of well-typed terms can actually be done syntactically. In fact some presentations, such as Saillard's [Sai15b], syntactically enforce it in the definition of terms by separating *object symbols* and *type symbols* of the signature.

Definition 2.3.26. *Assume a partition $\mathcal{F} = \mathcal{F}_O \uplus \mathcal{F}_T$ of symbols.*

A term t is syntactically well-formed, written $\mathbf{SWF}(t)$*, if either $t = \Box$ or t is represented by the following grammar:*

$$\begin{array}{lrcl} \textit{(Syntactical Object)} & \underline{t}, \underline{u}, \underline{v} & := & x \in \mathcal{X} \mid \mathtt{c} \in \mathcal{F}_O \mid \underline{u}\ \underline{v} \mid \lambda x : \underline{U}.\, \underline{t} \\ \textit{(Syntactical Type)} & \underline{T}, \underline{U}, \underline{V} & := & \mathtt{C} \in \mathcal{F}_T \mid \underline{U}\ \underline{v} \mid \lambda x : \underline{U}.\, \underline{T} \mid \Pi x : \underline{U}.\, \underline{T} \\ \textit{(Syntactical Kind)} & K & := & * \mid \Pi x : \underline{U}.\, K \end{array}$$

A context Γ is syntactically well-formed, written $\mathbf{SWF}(\Gamma)$*, if all its type annotations are syntactical types.*

A signature Σ is syntactically well-formed, written $\mathbf{SWF}(\Sigma)$*, if symbols in $\mathcal{F}_O$ have syntactical types, symbols in $\mathcal{F}_T$ have syntactical kinds and rewrite rules $(l \to r) \in \Sigma$ are such that l and r are both syntactical objects or both syntactical types.*

Lemma 2.3.27 (Level Reduction)**.** *If $t \xrightarrow[\beta]{p} u$ and t is a syntactical object (resp. type, resp. kind) then so is u. Besides,* $\mathbf{SWF}(t|_p)$ *and either*

- *$t|_p$ is a syntactical object and the reduction is at the* type level*, written $t \xrightarrow[\beta^*]{p} u$;*
- *$t|_p$ is a syntactical type and the reduction is at the* kind level*, written $t \xrightarrow[\beta^\Box]{p} u$.*

Proof. By a simple induction, the **SWF** property is stable by subterm and β-redexes can only be syntactical objects or types. □

It is possible to consider exclusively syntactically well-formed terms. This would define a restricted system in which both β-reduction and Σ-rewriting are restricted to syntactically well-formed terms. The corresponding conversion is a bit better behaved since pathological terms, such as $(\mathtt{max}\ *\ \mathtt{Nat})$ or $\lambda x : \mathtt{0}.\, \Box$ are syntactically forbidden. Such terms may therefore no longer interfere as middle-steps in the conversion between two well-typed terms.

Restricting the set of terms defines a more constrained system which may however be less expressive. However if $\xrightarrow[\beta\Sigma]{}$ is confluent, then both systems enjoy subject reduction, their conversion relations coincide and, therefore, they also define the same typing relation:

Lemma 2.3.28. *Assume $\xrightarrow[\beta\Sigma]{}$ confluent and* $\mathbf{WF}(\Sigma)$*, then the following holds for some symbol partition $\mathcal{F} = \mathcal{F}_O \uplus \mathcal{F}_T$.*

1. *If* $\mathbf{SWF}(\Sigma)$ *and* $\mathbf{SWF}(\Gamma)$ *and $\Sigma; \Gamma \vdash^{\mathrm{WF}}_\mathcal{D} t : A$ then* $\mathbf{SWF}(t)$ *and*

   (a) *if t is an object, then t is a syntactical object with relation to $\mathcal{F}_O$ and $\mathcal{F}_T$;*
   (b) *if t is a type , then t is a syntactical type with relation to $\mathcal{F}_O$ and $\mathcal{F}_T$;*
   (c) *if t is a kind , then t is a syntactical kind with relation to $\mathcal{F}_O$ and $\mathcal{F}_T$;*
2. *If* $\mathbf{SWF}(\Sigma)$ *and* $\Sigma \vdash^{\mathrm{WF}}_{\mathcal{D}} \Gamma$ $\mathbf{WF}_{\mathcal{D}}$ *then* $\mathbf{SWF}(\Gamma)$.
3. $\mathbf{SWF}(\Sigma)$.

Proof. We choose $\mathcal{F}_O$ and $\mathcal{F}_T$ such that for all $(\mathsf{c} : A) \in \Sigma$, $\Sigma; \varnothing \vdash_{\mathcal{D}} A : *$ if and only if $\mathsf{c} \in \mathcal{F}_O$. The three proofs are done by induction on the derivation of the $\lambda\Pi^{\mathrm{WF}}_{\equiv}$ judgment.

1. Using $\mathbf{SWF}(\Sigma)$ for $\mathcal{P}_\Sigma$, $\mathbf{SWF}(\Gamma)$ for $\mathcal{P}_\Gamma$. Case $\mathcal{P}_\equiv$ requires confluence, **SR** and Lemma 2.3.27. All other cases use induction hypothesis.
2. By induction hypothesis and using (1.).
3. By induction hypothesis and using (1.) and (2.). □

This means that in confluent and well-formed signatures, the restricted syntax of syntactically well-formed terms is actually large enough to encompass all possible well-typed terms and well-formed signatures and contexts of the (unconstrained) $\lambda\Pi_{\equiv}$.

Confluence of $\xrightarrow[\beta\mathcal{R}]{}$ usually needs to be proven first as it is required to ensure many critical properties of a $\lambda\Pi_{\equiv}$ signature. Since subject reduction is usually derived from confluence, the proof of confluence cannot rely on typing and must hold for the full untyped λ-calculus.

Even if we cannot rely on well-typedness to prove confluence, syntactical well-formedness is stable by (untyped) rewriting and β-reduction in syntactically well-formed contexts. Restricting the λ-calculus to syntactically well-formed terms allows to define a variant of $\lambda\Pi_{\equiv}$ just as well-behaved and offering more syntactical guarantees that can be used in the proof of properties of the rewrite system, such as confluence.

Lemma 2.3.29. *When restricted to syntactically well-formed terms:*

1. $\xrightarrow[\beta]{}$, $\xrightarrow[\beta^*]{}$ *and* $\xrightarrow[\beta^{\square}]{}$ *are confluent;*
2. $\xrightarrow[\beta]{p} = \xrightarrow[\beta^*]{p} \uplus \xrightarrow[\beta^{\square}]{p}$;

Proof. 1. By confluence of β on all terms, preservation of syntactical well-formedness and preservation of redexes syntactical category.

2. By case analysis on $t|_p$, since neither syntactical kinds nor $\square$ can be β-redexes. □

Lemma 2.3.30. *Syntactically well-formed terms are $\beta^{\square}$ strongly normalizing.*

Proof. The proof is done by induction on well-formed terms t. We assume, by contradiction, an infinite sequence of $\beta^{\square}$-reductions originating from t.

This sequence necessarily contains a step at the root, otherwise they would all take place in either immediate subterms of t allowing to extract an infinite sub-sequence of steps occurring exclusively in one of these strict subterms which are all strongly normalizing by induction hypothesis. This means that the assumed infinite sequence eventually reaches a term that is a $\beta^{\square}$-redex: $t = \underline{A}\ \underline{b} \xrightarrow[\beta^{\square}]{>_{\mathcal{P}}\Lambda\!\!\!\!\Rightarrow} (\lambda x\!:\!\underline{U}.\,\underline{T})\ \underline{w} \xrightarrow[\beta^{\square}]{\Lambda} \underline{T}\{x \mapsto \underline{w}\} = t'$. By induction

hypothesis, $\underline{A}$ and $\underline{b}$ are strongly normalizing. So are their respective reducts, $\lambda x : \underline{U}.\, \underline{T}$ and $\underline{w}$, and therefore so is $\underline{T}$.

We consider the set, $\mathcal{S} \ni t'$, of terms built from a $\beta^{\square}$-reduct, $\underline{T'}$, of $\underline{T}$ in which instances of x are substituted with (possibly distinct) $\beta^{\square}$-reducts of $\underline{w}$:

$$\mathcal{S} := \left\{ \underline{T'}[u_1]_{p_1} \cdots [u_n]_{p_n} \mid \underline{T} \underset{\beta^{\square}}{\longrightarrow\!\!\!\!\!\rightarrow} \underline{T'} \wedge \forall i, p_i \in \mathcal{P}os(\underline{T'}, x) \wedge \underline{w} \underset{\beta^{\square}}{\longrightarrow\!\!\!\!\!\rightarrow} u_i \right\}$$

Since the u_i are objects, even if they are λ-abstractions, they cannot create new $\beta^{\square}$-redexes at the kind level when substituted in $\underline{T'}$. Therefore, $\mathcal{S}$ is stable by $\beta^{\square}$-reduction. Assuming that $\mathcal{S} \ni \underline{T'}[u_1]_{p_1} \cdots [u_n]_{p_n} \underset{\beta^{\square}}{\xrightarrow{p}} s$, then either

- $p \geq_{\mathcal{P}} p_i$ for some i and $s = \underline{T'}[u_1]_{p_1} \cdots [v_i]_{p_i} \cdots [u_n]_{p_n}$ for some v_i such that $u_i \underset{\beta^{\square}}{\longrightarrow} v_i$;
- or $\forall i, p_i \# p$ or $p_i \geq_{\mathcal{P}} p \cdot F_\beta$ and $s = \underline{S}[v_1]_{q_1} \cdots [v_n]_{q_n}$ for some $\underline{S}$ such that $\underline{T'} \underset{\beta^{\square}}{\xrightarrow{p}} \underline{S}$.

In both cases, $s \in \mathcal{S}$.

Since $\underline{T}$ is strongly normalizing, there can only exists a finite number of steps of the latter kind which means that, starting from a certain term $s = \underline{T'}[u_1]_{p_1} \cdots [u_n]_{p_n}$, all subsequent steps occur in one of the n subterms $u_1, \ldots, u_n$ of s. We conclude since these subterms are all reducts of $\underline{w}$ and therefore strongly normalizing. □

Note that the number of $\beta^{\square}$-redexes in t may be non-decreasing due to upward-created β-redexes. For instance, assume the well-formed signature $\Sigma = \mathtt{A} : *, \mathtt{a} : \mathtt{A}$, we have $(\lambda x : \mathtt{A}.\, \lambda y : \mathtt{A}.\, \mathtt{A})\ \mathtt{a}\ \mathtt{a} \underset{\beta^{\square}}{\longrightarrow} (\lambda y : \mathtt{A}.\, \mathtt{A})\ \mathtt{a} \underset{\beta^{\square}}{\longrightarrow} \mathtt{A}$ and the first reduction step does not decrease the number of $\beta^{\square}$-redexes. In fact that number may greatly increase when considering the duplication of potential redexes in the lambda annotations of the substituted term.

2.4 Term rewriting formalisms

2.4.1 Existing formalisms

There exists several formalisms in the literature fit to properly represent term rewriting in a way compatible with the usual functional β-reduction, and rewrite reduction defined as the instances of *rewrite rules*. Terms of the lambda-Pi calculus modulo ($\lambda\Pi_{\equiv}$, see Section 2.3) are dependently typed but rewritten in an untyped setting. Therefore we focus here on formalisms where the definition and study of term rewriting is purely syntactical and does not rely on any form of typing condition of the rewritten terms. Besides, types, just like any other term, can be rewritten meaning that dependently typed symbols may not have a fixed arity. A symbol `f` may have a total number of expected arguments depending on the value of the first ones, which would therefore not be stable by substitution.

Example 1: Consider, for instance, a dependent type $\mathtt{T} : \mathtt{N} \to *$ and $\mathtt{f} : \Pi n : \mathtt{N}.\, \mathtt{T}\ n$. It may seem like `f` expects a single argument n and is therefore a symbol of arity 1. However the user may very well extend conversion with rewrite rules so that $\mathtt{T}\ (\mathtt{S}\ x) \twoheadleftarrow\!\!\!\!\!\longleftarrow \mathtt{N} \to \mathtt{T}\ x$.

In that case, the term $\mathtt{f}\ (\mathtt{S}\ x)$ actually expects an extra argument of type $\mathtt{N}$ and $\mathtt{f}\ (\mathtt{S}^k\ x)$ expects (at least) k more arguments. Finally there is no reason to forbid rewriting of partially applied symbols, such as the unapplied $\mathtt{f}$, or any other terms with a product type.

Saillard studied term rewriting in the lambda-Pi-calculus [Sai15a, Sai15b] using the formalism introduced by Müller [Mü92]. This formalism comes with some drawbacks in a dependently typed setting as it forces a fixed arity on the symbols. The first versions of Dedukti heavily relied on this property in its implementation as it allowed to get confluence of β together with rewriting from the confluence of rewriting alone which is better understood. This formalism forbids the use of β steps in the confluence of $\mathcal{R}$ and it forces rewritten symbols from the signature to have an arity. We we will show in the next chapter that similar confluence results and others can be achieved even when symbols are curried and can therefore be applied to arbitrarily many arguments.

Assaf [Ass15b] relies instead on Higher-Order Rewrite Systems (HRS), introduced by Nipkow [MN98]. As explained in [vO94], HRS and CRS are both particular cases of the more general HORS framework where only first order arities for meta-variables are considered. This presentation is quite expressive as it relies on the power of simply typed (meta-)lambda-calculus as so-called "substitution calculus". This typed meta-language is quite rich but does not clearly distinguish the respective roles of meta-variables, representing the set of their instances, and free or locally bounded term-variables, which are meant to bind names in expressions. It also requires matching, rewriting and even terms to be considered modulo the meta-β and meta-η conversions which can become cumbersome. The mechanism of substitution calculus is however quite expressive in general and allows, for instance, to simply define simultaneous reduction steps, called *orthogonal rewriting* in Chapter 3 (see Section 3.1), with a simple quite elegant substitution criterion: $u \otimes\!\!\Longrightarrow v$ iff $u =\!\!\downarrow^{\eta}_{\beta} t$, $v =\!\!\downarrow^{\eta}_{\beta} s$ and $t \longrightarrow s$.

Other formalisms were studied by Blanqui (IDTS) [Bla06] and Miller [Mil91] both developed in slightly different setting and for different purposes. In a joint work with Jouannaud, we introduced a particular formalism in [FJ19a] which is quite close to the one presented here except that it featured unannotated λ-abstractions. The results in this paper are quite similar to those developed in the following chapter but the differences between both settings forbid to simply reuse them here.

Our choice is to define the terms of the $\lambda\Pi_{\equiv}$ as a particular case of Combinatory Reduction Systems (CRS). CRS were introduced by Klop [Klo80, KvOvR93] as a general way to represent higher-order term rewriting. In this formalism, the functional β-reduction is a particular case of rewrite rule in an encoding of the λ-calculus. Our presentation adapts this formalism to represent annotated lambda-abstractions while keeping the binary application operator, @, which is therefore the root symbol of most terms even though it does not necessarily show. The link with Klop's CRS will not be made explicit everywhere in the following presentation however the reader should have no problem to infer the correspondence between constructions and properties from one setting to the other. In particular, adopting the notations of [KvOvR93] : $\lambda x : A.\, t := \boldsymbol{\lambda}(A, [x]t)$,

$t\ u := @(t,u)$, symbols $\mathtt{f} \in \mathcal{F}$ are symbols of arity 0, the notions of positions, subterm, free variables, substitutions all match and the β reduction corresponds to a particular rewrite rule: $@(\boldsymbol{\lambda}(A,[x]T(x)),U) \longrightarrow T(U)$. Our setting is in fact directly equivalent to a particular CRS in which $@$ and $\boldsymbol{\lambda}$ are two symbols of arity 2 and all other symbols are of arity 0.

Finally our notion of rewrite rules will rely on Miller's notion of pattern for the left-hand sides, preventing overlap between rewrite rules and β-reduction. This will ease the proofs of confluence by allowing to consider exclusively so-called *critical pairs* between rewrite rules.

2.4.2 Confluence of untyped rewriting

There are three main tools for analyzing confluence of a term rewrite relation: Newman's Lemma [New42], Hindley-Rosen's Lemma [Hin69], and van Oostrom's Theorem which generalizes the two previous ones [vO08]. Since β-reduction does not terminate in pure lambda calculus, Newman's Lemma does not apply. And if the rules have non-trivial critical pairs, then Hindley-Rosen's Lemma does not apply either. Even the subtle use of Hindley-Rosen's Lemma allowing development-closed critical pairs [vO97] is too restrictive for practical use. The way out is the use of van Oostrom's decreasing diagrams [vO94]. The fact that β reduction does not terminate on untyped lambda terms is no obstacle since that criterion does not rely on termination for showing confluence. A further reason for considering pure lambda terms is that it is then easy to deduce confluence for any type system, including dependent type systems, for which the rules enjoy type preservation.

Van Oostrom's theorem is abstract, its application to term rewriting relations conceals many difficulties, especially in an untyped higher-order setting where the usual properties of type systems, such as product compatibility, subject reduction, unicity of typing or termination, are unavailable. Further, neither confluence nor termination are preserved by adding a confluent and terminating set of rewrite rules to the pure λ-calculus. A counter-example to termination in the simply typed λ-calculus is given in [Oka89]. Numerous counter-examples to confluence in the pure λ-calculus are given in [Klo80].

It is therefore essential to develop confluence criteria for non-terminating systems. This is the subject of the coming Chapter 3, focusing on left-linear systems, and Chapter 4 including non-left-linear rules.

Chapter 3

Confluence of Left-Linear Systems

Term rewriting is seldom deterministic. Several *redexes* may coexist in the same term which therefore has several *reducts* depending on the chosen evaluation strategy. These redexes may be either at parallel positions, nested one below the other or overlapping. Parallel redexes do not interfere with one another, nested redexes are trickier but can still be handled in the left-linear case and overlapping redexes require some guarantees on the so-called *critical pairs* of the rewrite system. The *confluence* property allows to ignore this non-determinism. In a confluent system, evaluation may be considered forward only, removing the need for costly backtracks and all evaluation strategies will eventually reach the same normal form if it exists.

In this chapter, we introduce several general purpose confluence criteria for confluence of left-linear higher-order rewrite systems together with β.

A left-linear higher-order rewrite systems $\mathcal{R}$ that is confluent alone remains confluent when considered together with β, this is Theorem 3.5.3. Showing that $\mathcal{R}$ is confluent can be done using techniques relying, for instance, on better known first-order techniques, if the rules are simple enough, or on strong normalization. This was done for instance by Barbanera, Fernàndez and Geuvers [BFG94] to extend [GN91] and show strong normalization and confluence of rewriting together with β in all the pure type systems of the λ-cube. In the case of non-overlapping sets of rewrite rules, the result is immediate, Corollary 3.5.3.1. Otherwise, as in first order, it is necessary that all critical pairs have a closing sequence in order for rewriting to be even locally confluent. In the case of strongly normalizing systems, Newman's lemma applies and this condition is actually sufficient, see Theorem 3.5.5. Otherwise it is necessary to provide decreasing diagrams joining the critical pairs.

Van Oostrom decreasing diagrams [vO94] have allowed the analysis of confluence for first-order rewriting relations that are non-terminating, in the left-linear case first [Fel13], and then in the notoriously much more difficult non-left-linear case [LJO15]. More precisely, they have allowed to reduce the confluence property to the existence of decreasing diagrams for all (parallel [Fel13] or rational [LJO15]) critical pairs. Van Oostrom has used his technique to show the confluence of a particular higher-order calculus with explicit substitutions [vO08]. But to our knowledge, the only existing attempts to reduce

the confluence property of an arbitrary higher-order system to the existence of decreasing diagrams for its critical pairs have been by [ADJL16], [Dow19] and [FJ19a], which all assume left-linearity. Further, all these results apply to λ-calculi without type annotations, possibly making their application problematic, in particular to dependently typed theories for which type erasures are unsound. Our goal here is to generalize [FJ19a] to a λ-calculus with type annotations, more precisely the one defined in Chapter 2.

In first order setting, already, rules that are not right-linear may duplicate existing redexes occurring below and therefore it is natural to consider parallel rewrite steps in order to close peaks with a single (multi-)step. For example the rule $\mathtt{f}\ X \longrightarrow \mathtt{g}\ X\ X$ may duplicate redexes below: $\mathtt{f}\ (\mathtt{f}\ t) \longrightarrow \mathtt{g}\ (\mathtt{f}\ t)\ (\mathtt{f}\ t)$. In a higher-order setting, some rules, such as β, may nest redexes below other redexes by substituting locally bounded variables with terms. In particular, two redexes occurring at parallel positions and meant to be reduced simultaneously can be moved one below, or rather "inside", the other, therefore forbidding their simultaneous reduction. For instance, $(\lambda x.\, \mathtt{f}\ x)\ (\mathtt{f}\ t) \longrightarrow \mathtt{f}\ (\mathtt{f}\ t)$. Because of this, we often need to consider a stronger simultaneous multi-step reduction allowing to group together both

- parallel steps, played in any order: if $u \otimes\!\!\Longrightarrow u'$ and $v \otimes\!\!\Longrightarrow v'$, then $u\ v \otimes\!\!\Longrightarrow u'v'$;
- nested steps provided they do not overlap with the above steps (which is never the case for β): if $u \otimes\!\!\Longrightarrow u'$, $v \otimes\!\!\Longrightarrow v'$, then $(\lambda x : t.\, u)\ v \otimes\!\!\Longrightarrow u'\{x \mapsto v'\}$.

This relation is called *orthogonal* rewriting.

Since $\longrightarrow \subseteq \otimes\!\!\Longrightarrow \subseteq \twoheadrightarrow$, the confluence of $\longrightarrow$ is deduced from that of this extension of simple rewriting. In left-linear systems, $\otimes\!\!\Longrightarrow$ is particularly well-behaved. In particular, in the absence of non trivial critical pairs, it satisfies a stronger version of the diamond property: if $s \Longleftarrow\!\!\otimes\, u \otimes\!\!\Longrightarrow t$, then s,u and t all reduce in one step to a common reduct. This property allowed Cockx, Tabareau and Winterhalter [CTW20] to devise a very convenient practical confluence criteria requiring only that the ancestor of all critical pairs reduces in a single orthogonal step to a common *optimal* reduct. Besides the fact that this criteria can easily be checked, automatically, *completing* the system if necessary, it also allows some form of modularity of confluence between blocks of rewrite rules that have no pairwise critical pairs.

Using these two techniques, decreasing diagrams and multi-step rewriting, we show two general purpose confluence criteria. In the case where the systems is not self-nested and β-steps can be avoided in the decreasing diagrams, we only need to consider parallel critical pairs and ensure decreasing diagrams do not interfere with non-overlapping parts, Theorem 3.5.8. This last condition comes from Toyama [Toy81] and extends Huet's criteria for first-order TRS [Hue80] which requires the systems to be parallel closed: critical pairs (s, t) must satisfy $s \Longrightarrow t$. In some cases, $\mathcal{R}$ is not enough to close critical peaks which require β steps. Our last criterion requires then to show the existence of decreasing diagrams for all *orthogonal critical pairs* to retrieve confluence with β. In that case β steps are considered of low weight so they can be used for free in the closing diagrams.

Variants of most of the following results can be found in the recent work of Jouannaud and several other collaborators, [JL12a, LDJ14, ADJL16, Dow19, FJ19b, FJ19a] including the author but in a setting that forbid their application to the particular case of the

lambda-Pi-calculus modulo, for the most part because of the handling of type annotations in λ-abstractions. We adopt here a presentation better befitting our needs and take extra care in providing all the confluence criteria that we deem the most useful in order to prove the confluence of practical rewrite systems.

We prove that if one of the four following condition is satisfied, then the left-linear higher-order rewrite systems $\mathcal{R}$ is confluent together with β-reduction:

- Theorem 3.5.3: $\mathcal{R}$ has no critical pair ;
- Theorem 3.5.5: $\mathcal{R}$ is strongly normalizing and its critical pairs are joinable with $\mathcal{R}$;
- Theorem 3.5.8: $\mathcal{R}$ is not self-nested, satisfies Toyama's Variable Condition [Toy81] and is labeled so that all parallel critical pairs have a decreasing diagram using rules in $\mathcal{R}$;
- Theorem 3.5.9: $\mathcal{R}$ is labeled so that all orthogonal critical pairs have a decreasing diagram allowing β steps.

3.1 Orthogonal rewriting

In the case of left-linear rules, confluence proofs sometimes require to consider a safe multi-step extension of rewriting. As introduced in Tait's confluence proof of the lambda-calculus, we define *orthogonal reductions* (called parallel reductions in [Bar81]) as the grouping of steps both at parallel positions and nested below, provided they do not overlap with the above steps. Our definition is essentially Tait's, but makes the rewriting positions explicit. All positions orthogonally rewritten correspond to a redex in the rewritten term. Note that rather than defining simultaneous reduction, our definition relies on the grouping of compatible sequential steps: reductions at parallel subsets can be grouped in any order (horizontal grouping) while steps nested below a redex must be played before the redex itself (vertical grouping).

Definition 3.1.1 (Orthogonal Rewriting)**.** *Assume a set $\mathcal{R}$ of left-linear rules and a set O of positions.* Orthogonal rewriting with $\mathcal{R}$ at positions O, *written* $u \overset{O}{\underset{\mathcal{R}}{\otimes\!\!\Longrightarrow}} v$, *is the smallest reflexive* ($u \overset{\varnothing}{\otimes\!\!\Longrightarrow} u$) *relation satisfying the following properties:*

- *if* $P \# Q$ *then* $\overset{P}{\underset{\mathcal{R}}{\otimes\!\!\Longrightarrow}} \overset{Q}{\underset{\mathcal{R}}{\otimes\!\!\Longrightarrow}} \subseteq \overset{P \uplus Q}{\underset{\mathcal{R}}{\otimes\!\!\Longrightarrow}}$;
- *if* $(L, R) \in \mathcal{R}$ *and* $O \geq_{\mathcal{P}} p \cdot F_L$ *then* $\overset{O}{\underset{\mathcal{R}}{\otimes\!\!\Longrightarrow}} \underset{(L,R)}{\overset{\{p\}}{\longrightarrow}} \subseteq \overset{\{p\} \uplus O}{\underset{\mathcal{R}}{\otimes\!\!\Longrightarrow}}$.

Parallel steps are played in any order however nested steps must be played before the above step to preserve the set of positions. The sets $\mathcal{R}$ of rules and O of positions are omitted if irrelevant or clear from context. We write $\overset{O}{\underset{L \to R}{\otimes\!\!\Longrightarrow}} := \overset{O}{\underset{\{L \to R\}}{\otimes\!\!\Longrightarrow}}$ and $\overset{O}{\underset{\beta}{\otimes\!\!\Longrightarrow}} := \overset{O}{\underset{\{(L_\beta, R_\beta)\}}{\otimes\!\!\Longrightarrow}}$.

3.1.1 Properties

Among many other key properties, sequential orthogonal rewriting steps can, under some conditions, be grouped together or split into orthogonal steps at compatible subsets of positions.

Lemma 3.1.2. *Orthogonal rewriting satisfies the following properties:*

- *Monotonicity: if* $s \overset{O}{\otimes\!\!\Longrightarrow} t$ *and* $q \in \mathcal{P}os(u)$ *then* $u[s]_q \overset{q\cdot O}{\otimes\!\!\Longrightarrow} u[t]_q$
- *Stability: if* $s \overset{O}{\otimes\!\!\Longrightarrow} t$ *then* $s\sigma \overset{O}{\otimes\!\!\Longrightarrow} t\sigma$
- $\overset{\{p\}}{\otimes\!\!\Longrightarrow} = \xrightarrow{p}$ *and if* P *parallel,* $\overset{P}{\otimes\!\!\Longrightarrow} = \overset{P}{\Longrightarrow}$
- $\longrightarrow \subseteq \Longrightarrow \subseteq \otimes\!\!\Longrightarrow \subseteq \twoheadrightarrow$
- $\overset{O}{\otimes\!\!\Longrightarrow} \subseteq \overset{\geq_{\mathcal{P}} O}{\otimes\!\!\Longrightarrow}$ *Therefore if* $s \overset{q\cdot O}{\otimes\!\!\Longrightarrow} t$ *then* $s|_q \overset{O}{\otimes\!\!\Longrightarrow} t|_q$ *and* $\underline{s}_q = \underline{t}_q$
- *If* $P \# Q$, $\overset{P}{\otimes\!\!\Longrightarrow} \overset{Q}{\otimes\!\!\Longrightarrow} = \overset{P\uplus Q}{\otimes\!\!\Longrightarrow}$
- *If* P *parallel and* $Q \geq_{\mathcal{P}} P \cdot F_L$ *then* $\overset{Q}{\otimes\!\!\Longrightarrow} \overset{P}{\Longrightarrow} = \overset{P\uplus Q}{\otimes\!\!\Longrightarrow}$

Proof. The first two are done by induction on O. The first case is easy, the second requires induction hypothesis and the last one the monotonicity and stability properties of simple rewriting. Third and fourth are by definition. Fifth is again by induction on O using monotonicity. Sixth's $\subseteq$ is by definition and $\supseteq$ by induction on $P \cup Q$. Seventh's $\subseteq$ is also by definition and previous points while $\supseteq$ is done by commuting parallel subsets of steps. If $s \overset{P\uplus Q}{\otimes\!\!\Longrightarrow} t$ for some $Q = \biguplus_{p\in P}(p \cdot Q_p)$, then by previous point $s \overset{p_1\cdot Q_{p_1}}{\otimes\!\!\Longrightarrow} \xrightarrow{p_1} \cdots \overset{p_n\cdot Q_{p_n}}{\otimes\!\!\Longrightarrow} \xrightarrow{p_n} t$ we then commute parallel steps $s \overset{p_1\cdot Q_{p_1}}{\otimes\!\!\Longrightarrow} \cdots \overset{p_n\cdot Q_{p_n}}{\otimes\!\!\Longrightarrow} \xrightarrow{p_1} \cdots \xrightarrow{p_n} t$ and conclude. □

Definition 3.1.3. *The set* O *is* compatible *with* F *if* $\forall p, q \in O, p <_{\mathcal{P}} q \Rightarrow p \cdot F \leq_{\mathcal{P}} q$.

Lemma 3.1.4. *If* $O \neq \varnothing$ *is compatible with* F_L *then* $\overset{\overline{O}}{\underset{L\to R}{\otimes\!\!\Longrightarrow}} \overset{O}{\underset{L\to R}{\Longrightarrow}} = \overset{O}{\underset{L\to R}{\otimes\!\!\Longrightarrow}}$.

For instance, assuming $F = F_\beta = \{11, 12, 2\}$, then the set $O = \{\Lambda, 1\}$ is not compatible with F. Therefore a "below" step at position $\overline{O} = \{1\}$ and an "above" step at position $\underline{O} := \{\Lambda\}$ cannot be merged into a single orthogonal step: $(\lambda x.\, \lambda y.\, y)\ s\ t \xrightarrow{\{1\}} (\lambda y.\, y)\ t \xrightarrow{\{\Lambda\}} t$.

Note that if O is compatible with F, then so are all its subsets. Besides if $s \overset{O}{\underset{L\to R}{\otimes\!\!\Longrightarrow}}$ then O is compatible with F_L. The compatibility condition guarantees that the residual orthogonal reduction below $\overset{\overline{O}}{\underset{\beta}{\otimes\!\!\Longrightarrow}}$ neither creates nor interfere with the above parallel redexes $\overset{\underline{O}}{\underset{\beta}{\Longrightarrow}}$.

A set of positions compatible with the fringe F_L of a rewrite rule $L \to R$ allows a simple criterion for orthogonal rewriting at position O with that rule. In particular, any

functional orthogonal rewriting step $\overset{O}{\underset{\beta}{\otimes\!\!\Longrightarrow}}$ can be decomposed this way if O is compatible with F_β.

The following properties of orthogonal step splitting and merging will be needed for the coming analysis of orthogonal ancestor peaks.

Lemma 3.1.5. *If $Q \not\geq_{\mathcal{P}} P$ and $P \cap Q \cdot \mathcal{FP}os(L) = \varnothing$ then $\overset{P}{\underset{(L,R)}{\otimes\!\!\Longrightarrow}} \overset{Q}{\underset{(L,R)}{\otimes\!\!\Longrightarrow}} = \overset{P \uplus Q}{\underset{(L,R)}{\otimes\!\!\Longrightarrow}}$.*

Proof. By induction on the size of $P \uplus Q$. The $P = \varnothing$ and $Q = \varnothing$ cases are immediate. Otherwise $\underline{Q} \not\geq_{\mathcal{P}} \underline{P}$ and therefore $\underline{P} = P_1 \uplus P_2$ with $P_1 >_{\mathcal{P}} Q$ and $P_2 \# Q$.

$$
\begin{aligned}
\overset{P}{\otimes\!\!\Longrightarrow}\overset{Q}{\otimes\!\!\Longrightarrow} &= \overset{\overline{P}}{\otimes\!\!\Longrightarrow}\overset{\underline{P}}{\Longrightarrow}\overset{\overline{Q}}{\otimes\!\!\Longrightarrow}\overset{\underline{Q}}{\Longrightarrow} && \text{(by Lemma 3.1.2)}\\
&= \overset{\overline{P}}{\otimes\!\!\Longrightarrow}\overset{P_1}{\Longrightarrow}\overset{P_2}{\Longrightarrow}\overset{\overline{Q}}{\otimes\!\!\Longrightarrow}\overset{\underline{Q}}{\Longrightarrow} && \text{(by Lemma 2.2.24)}\\
&= \overset{\overline{P}}{\otimes\!\!\Longrightarrow}\overset{P_1}{\Longrightarrow}\overset{\overline{Q}}{\otimes\!\!\Longrightarrow}\overset{P_2}{\Longrightarrow}\overset{\underline{Q}}{\Longrightarrow} && \text{(by definition } \overline{Q}\#P_2)\\
&= \overset{\overline{P} \uplus P_1 \uplus \overline{Q}}{\otimes\!\!\Longrightarrow}\overset{P_2}{\Longrightarrow}\overset{\underline{Q}}{\Longrightarrow} && \text{(by induction hypothesis } \overline{P} \not\leq_{\mathcal{P}} P_1 \not\leq_{\mathcal{P}} \overline{Q})\\
&= \overset{\overline{P} \uplus P_1 \uplus \overline{Q}}{\otimes\!\!\Longrightarrow}\overset{P_2 \uplus Q}{\Longrightarrow} && \text{(by Lemma 2.2.24)}\\
&= \overset{\overline{P \uplus Q}}{\otimes\!\!\Longrightarrow}\overset{P \uplus Q}{\Longrightarrow} = \overset{P \uplus Q}{\otimes\!\!\Longrightarrow} && \text{(by Lemma 2.1.19 and Lemma 3.1.2)}
\end{aligned}
$$

Since $P_2 \uplus \underline{Q}$ parallel and $P_2 \uplus \underline{Q} <_{\mathcal{P}} \overline{P} \uplus P_1 \uplus \overline{Q}$. □

The hypothesis simply states that P contains exclusively positions disjoint with Q and positions strictly below $Q \cdot F_L$, therefore guaranteeing that the disjoint union $P \uplus Q$ is compatible with F_L. For instance $(\lambda x.(\lambda y.y)\ x)\ t\ ((\lambda z.z)\ t) \overset{\{111,2\}}{\otimes\!\!\Longrightarrow} (\lambda x.x)\ t\ t \overset{\{1\}}{\otimes\!\!\Longrightarrow} t\ t$ can be merged into a single $(\lambda x.(\lambda y.y)\ x)\ t\ ((\lambda z.z)\ t) \overset{\{1,111,2\}}{\otimes\!\!\Longrightarrow} t\ t$ and split back into $(\lambda x.(\lambda y.y)\ x)\ t\ ((\lambda z.z)\ t) \overset{\{111\}}{\otimes\!\!\Longrightarrow} (\lambda x.x)\ t\ ((\lambda z.z)\ t) \overset{\{1,2\}}{\otimes\!\!\Longrightarrow} t\ t$. However, playing the above step first, $(\lambda x.(\lambda y.y)\ x)\ t\ ((\lambda z.z)\ t) \overset{\{1\}}{\otimes\!\!\Longrightarrow} (\lambda y.y)\ t\ ((\lambda z.z)\ t) \overset{\{1,2\}}{\otimes\!\!\Longrightarrow} t\ t$, yields a decomposition at positions which union is not the entire set of redexes.

This very general lemma ensures, for instance, that orthogonal β-steps can be split into subsets of positions played sequentially and merged back as long as the positions of the first step are not above the second (either below or incomparable). This lemma will be used extensively, sometimes without notice. In particular it allows to prove the following crucial lemma.

Lemma 3.1.6 (Orthogonal pasting)**.** *If $u \overset{O}{\otimes\!\!\Longrightarrow} v$ and $\sigma \otimes\!\!\Longrightarrow \tau$, then $u\sigma \overset{P}{\otimes\!\!\Longrightarrow} v\tau$ for some $P \geq_{\mathcal{P}} O \cup \{p \in \mathcal{P}os(u) \mid u(p) \in \mathcal{D}om(\sigma)\}$.*

Proof. It is clear by induction on u and monotonicity that if $\mathcal{D}om(\sigma) \subseteq \mathcal{X}$, then $u\sigma \overset{P}{\otimes\!\!\Longrightarrow} u\tau$ for some $P \geq_{\mathcal{P}} \mathcal{VP}os(u)$. We prove the result by induction on u and $|O|$.

If $\Lambda \notin O$, then in the $u = u_1\ u_2$, $u = \lambda x\!:\!u_1.\,u_2$, $u \in \mathcal{F}$ and $u \in \mathcal{X} \cup (\mathcal{Z} \setminus \mathcal{D}om(\sigma))$ cases, the result is straightforward by induction hypothesis and grouping of parallel steps.

In the $u = X[\overline{u}]$ with $X \in \mathcal{D}om(\sigma)$ case, we have by assumption $\sigma(X) = \underline{\lambda}\overline{z}.s$ and $\tau(X) = \underline{\lambda}\overline{z}.t$ with $s \otimes\!\!\Longrightarrow t$. By induction hypothesis, we have $\{\overline{z} \mapsto \overline{u}\} \otimes\!\!\Longrightarrow \{\overline{z} \mapsto \overline{v}\}$ and by our preliminary result and stability, $u\sigma = s\{\overline{z} \mapsto \overline{u}\} \overset{P}{\otimes\!\!\Longrightarrow} s\{\overline{z} \mapsto \overline{v}\} \otimes\!\!\Longrightarrow t\{\overline{z} \mapsto \overline{v}\} = u\tau$ which can be grouped into a single step by Lemma 3.1.5.

If $O = \{\Lambda\} \uplus O'$, then $u \overset{O'}{\otimes\!\!\Longrightarrow} u' \underset{(L,R)}{\overset{\Lambda}{\longrightarrow}} v$ and by induction hypothesis and stability, $u\sigma \overset{P'}{\otimes\!\!\Longrightarrow} u'\tau \underset{(L,R)}{\overset{\Lambda}{\longrightarrow}} v\tau$ for some $P' \geq_{\mathcal{P}} O' \cup \{p \in \mathcal{P}os(u) \mid u(p) \in \mathcal{D}om(\sigma)\}$. To conclude with Lemma 3.1.5, we only need to show $P' \cap \mathcal{F}\mathcal{P}os(L) = \varnothing$. By definition of orthogonal rewriting, we already have $O' \geq_{\mathcal{P}} F_L$. Assume $p \in \mathcal{F}\mathcal{P}os(L)$ and $u(p) \in \mathcal{D}om(\sigma)$. Since $p \not\geq_{\mathcal{P}} O'$, $p \in \mathcal{P}os(u')$ and since $u' = L\gamma$ and $p \not\geq_{\mathcal{P}} F_L$, $p \in \mathcal{P}os(L)$. Therefore $L(p) \in \mathcal{Z} \cup \mathcal{X}$ is either a meta-variable of L or a locally bound variable and $u(p) \notin \mathcal{D}om(\sigma)$. □

Lemma 3.1.7. *Assume $q \in \mathcal{P}os(u)$ and $O = P \uplus Q \uplus R \subseteq \mathcal{P}os(u)$ with $P\#q$, $q \leq_{\mathcal{P}} Q$ and $R <_{\mathcal{P}} q$. Then $u \overset{O}{\underset{\beta}{\otimes\!\!\Longrightarrow}} v$ iff $u \overset{P}{\underset{\beta}{\otimes\!\!\Longrightarrow}} \overset{Q}{\underset{\beta}{\otimes\!\!\Longrightarrow}} \overset{R}{\underset{\beta}{\otimes\!\!\Longrightarrow}} v$ iff $u \overset{Q}{\underset{\beta}{\otimes\!\!\Longrightarrow}} \overset{P}{\underset{\beta}{\otimes\!\!\Longrightarrow}} \overset{R}{\underset{\beta}{\otimes\!\!\Longrightarrow}} v$.*

Proof. Since $P\#Q$ and $P \uplus Q \not\leq_{\mathcal{P}} R$. □

Lemma 3.1.8. *$(\lambda x\!:\!A.\,t)\ u \overset{O}{\underset{\beta}{\otimes\!\!\Longrightarrow}} v$ with $\Lambda \in O$ if and only if $O = \{\Lambda\} \uplus 12{\cdot}P \uplus 2{\cdot}Q \uplus 11{\cdot}R$, $t \overset{P}{\underset{\beta}{\otimes\!\!\Longrightarrow}} t'$, $u \overset{Q}{\underset{\beta}{\otimes\!\!\Longrightarrow}} u'$ and $v = t'\{x \mapsto u'\}$.*

3.1.2 Left-linear systems

The most important property of left-linear pre-patterns is that they still match the splitting of their instances below their fringe.

Lemma 3.1.9. *Assume L a linear pre-pattern and $P \geq_{\mathcal{P}} F_L$ a parallel set of positions. Then $\underline{L\sigma}_P = L\sigma'$ for some σ' such that $\sigma = \sigma'\overline{L\sigma}^P$.*

Proof. We assume $P = \{o \cdot p\}$ for some $o \in F_L$. Then $L|_o = X[\overline{x}]$ for some meta-variable X and variables $\overline{x}$ bound above o in L. Necessarily $\sigma(X) = \underline{\lambda}\overline{x}.t$ such that $(L\sigma)|_o = t$ and we have $\underline{L\sigma}_{o\cdot p} = L\sigma[\,Z[\overline{z}]\,]_{o\cdot p} = L\sigma[\,t[\,Z[\overline{z}]\,]_p\,]_o = L\sigma'$ for σ' identical to σ except that $\sigma'(X) = \underline{\lambda}\overline{x}.t[\,Z[\overline{z}]\,]_p$. Since $\underline{L\sigma}_{o\cdot p} = \{Z \mapsto \underline{\lambda}\overline{z}.(L\sigma)|_{o\cdot p}\}$, we check that $(\sigma'\underline{L\sigma}_{o\cdot p})(X) = \underline{\lambda}\overline{x}.t[\,Z[\overline{z}]\,]_p\underline{L\sigma}_{o\cdot p}\underline{\lambda}\overline{x}.t[\,(L\sigma)|_{o\cdot p}\,]_p = \underline{\lambda}\overline{x}.t = \sigma(X)$. □

Lemma 3.1.10. *If L a linear pre-pattern and $L\sigma \overset{\geq_{\mathcal{P}} F_L}{\longrightarrow} u$ then $\sigma \longrightarrow \tau$ and $u = L\tau$.*

Proof. By definition, $\overline{L\sigma}^{F_L} \longrightarrow \gamma$ such that $u = \underline{L\sigma}_{F_L}\gamma$. By Lemma 3.1.9, $\underline{L\sigma}_{F_L} = L\sigma'$ such that $\sigma = \sigma'\overline{L\sigma}^{F_L}$. We conclude since, by stability, $\sigma \longrightarrow \sigma'\gamma =: \tau$. □

Note that we can choose $\longrightarrow$ to be any reduction, including orthogonal rewriting, $\underset{\mathcal{R}}{\otimes\!\Longrightarrow}$.

Lemma 3.1.11 (Preservation)**.** *If $u \xrightarrow[(L,R)]{p} v$ and $p \cdot \mathcal{FP}os(L) \not\geq_{\mathcal{P}} q$ (i.e. $q \# p$ or $q \geq_{\mathcal{P}} q \cdot F_L$), then $\underline{u}_q \xrightarrow[(L,R)]{p} w$ for some w such that $v = w\overline{u}^q$.*

Proof. We have $u|_p = L\gamma$. If $q \# p$, this is easy since $\underline{u}_q|_p = u_p = L\gamma$ and $w = \underline{u}_q[R\gamma]_p$ and therefore $w\overline{u}^q = u[Z[\overline{z}]]_q[R\gamma]_p\overline{u}^q = u[R\gamma]_p[Z[\overline{z}]]_q\overline{v}^q = v[Z[\overline{z}]]_q\overline{v}^q = \underline{v}_q\overline{v}^q = v$. Otherwise, $q \geq_{\mathcal{P}} p \cdot F_L$. Assuming $p = \Lambda$, by Lemma 3.1.9, we have $\underline{u}_q = \underline{L\gamma}_q = L\gamma' \underset{(L,R)}{\longrightarrow} R\gamma' = w$ and $w\overline{u}^q = R\gamma'\overline{L\gamma}^q = R\gamma = v$. We extend to the case $p \neq \Lambda$ by monotonicity. □

Corollary 3.1.11.1. *Assume L a linear pre-pattern and Q parallel such that $O \cdot F_L \not\geq_{\mathcal{P}} Q$. If $u \underset{(L,R)}{\overset{O}{\otimes\!\Longrightarrow}} v$, then $\underline{u}_Q \underset{(L,R)}{\overset{O}{\otimes\!\Longrightarrow}} w$ for some w such that $v = w\overline{u}^Q$.*

Proof. The case where O and Q are singleton sets is Lemma 3.1.11. We extend it to any parallel Q by induction on Q and then to any O by induction on O. □

Lemma 3.1.12 (Orthogonal splitting)**.** *Assume $u \underset{(L,R)}{\overset{O}{\otimes\!\Longrightarrow}} v$ such that $O = P \uplus Q$ with $P \not\geq_{\mathcal{P}} Q$. Then $\underline{u}_Q \underset{(L,R)}{\overset{P}{\otimes\!\Longrightarrow}} w$, $\overline{u}^Q \underset{(L,R)}{\otimes\!\Longrightarrow} \sigma$ and $v = w\sigma$.*

Proof. By Lemma 3.1.5, $u = \underline{u}_Q\overline{u}^Q \underset{(L,R)}{\overset{Q}{\otimes\!\Longrightarrow}} v' \underset{(L,R)}{\overset{P}{\otimes\!\Longrightarrow}} v$. By Lemma 3.1.10, $\overline{u}^Q \underset{(L,R)}{\otimes\!\Longrightarrow} \sigma$ such that $v' = \underline{u}_Q\sigma$. Since O is compatible with F_L and $P \not\geq_{\mathcal{P}} Q$, $P \cdot F_L \not\geq_{\mathcal{P}} Q$ and, by Corollary 3.1.11.1, $\underline{u}_Q \underset{(L,R)}{\overset{P}{\otimes\!\Longrightarrow}} w$ for some w such that $v = w\overline{u}^Q$. □

3.2 Decreasing Diagrams

When rewriting terminates, it is well-known that the joinability of all local peaks implies the confluence property, this is the so-called Newman's lemma. When it does not, it is then necessary to strengthen joinability, this is the rôle of *decreasing diagrams*, introduced by van Oostrom [vO94] then extended to the more general version that we present here. In the following, we consider abstract rewrite relations which elementary steps are equipped with a label belonging to some well-founded set which strict partial order is written $\rhd$.

Our definition of decreasing diagram was developed by Jouannaud and van Oostrom and generalizes the one introduced by van Oostrom [vO08] to allows bidirectional steps in the side and middle steps.

Definition 3.2.1 (Generalized Decreasing diagram [JvO09])**.** *Given a labeled relation $\longrightarrow$ on an abstract set, we denote by $DS(m,n)$ the set of* decreasing sequences *of the form $u \overset{\delta}{\twoheadleftarrow} v$ or $u \overset{\gamma}{\twoheadrightarrow} s \overset{n}{\longrightarrow} t \overset{\delta}{\twoheadleftarrow} v$ such that the labels in γ are strictly smaller than m*

and the labels in δ are strictly smaller than m or n. The steps labeled by γ, n and δ, are called the side *steps,* facing *step and* middle *steps of the decreasing sequence, respectively.*

Given a local peak $v \xleftarrow{m} u \xrightarrow{n} w$, a (locally) decreasing diagram *is a pair of derivations from v and w to some common term t, belonging to $DS(m,n)$ and $DS(n,m)$, respectively.*

Decreasing diagrams are represented in Figure 3.1 and abbreviated as DDs. Note that a facing step of a decreasing diagram may be missing, its side steps are then absorbed by the middle ones.

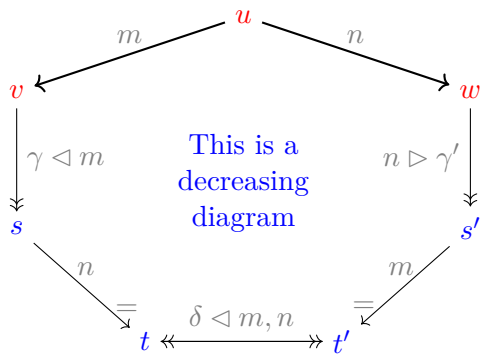


Figure 3.1: Decreasing diagram

Theorem 3.2.2 ([vO94])**.** *A labeled relation is Church-Rosser if all its local peaks have decreasing diagrams.*

The existence of decreasing diagrams is a very general criterion for confluence of abstract relations. It is even complete for weakly normalizing relations and relation on countable sets of objects, such as subsets of terms, [vO94]. For a more in-depth presentation of decreasing diagrams with relation to confluence and commutation properties, we highly recommend [EKO19].

The decreasing diagram technique allows to consider exclusively local peaks to prove the confluence of a relation. In the particular case of term rewriting, many kinds of local peaks $s \xleftarrow[\mathcal{R}]{p} u \xrightarrow[\mathcal{R}]{q} t$ correspond to rewrite steps at disjoint and compatible position(s) p and q. In order to prove the confluence of $\xrightarrow[\mathcal{R}]{}$, we need only to define a labeling of rewrite steps such that all local peaks have a decreasing diagram. The study of all possible local peaks of $\xrightarrow[\mathcal{R}]{} \cup \xrightarrow[\beta]{}$ is critical to find the correct labeling function which depends on properties of the set of rules $\mathcal{R}$. A labeling function may rely on the whole reduced term, on the position of the redex in this term, on the redex itself (independently of its position) or simply on the reducing rule.

Van Oostrom's theorem generalizes to rewriting modulo an equational theory $=_E$ in which case $\rhd$ must be compatible with the equational theory [JL12b]. This is for example the case when rewriting terms of the λ-calculus for which α conversion is built-in and must be compatible with the chosen definition of reduction over terms. Equational steps are considered to have a minimal label and often ignored.

3.3 Critical peaks

A key property of plain first-order rewriting is that there are three possible kinds of local peaks depending on the respective positions of the rewrites that define them. Either they occur at parallel positions, one is nested below the other, or they overlap and form a so-called critical peak. This property generalizes trivially to positional higher-order rewrites.

The most problematic case of local peak is when the redexes in u *overlap* in the sense that they both rely on the same part of u to be fired. Positional rewriting allows to simply characterize these overlaps: they occur if $q = p \cdot o$ for some $o \in \mathcal{FP}os(L)$, i.e. $o \not\geq_{\mathcal{P}} F_L$. This property can even be generalized to local peaks of orthogonal multi-step reductions.

3.3.1 Simple critical peaks

In the case of single step reductions, the critical peaks are generated by the existing overlaps between pairs of left-hand sides.

Definition 3.3.1. *If $o \in \mathcal{FP}os(L)$ (i.e. $o \not\geq_{\mathcal{P}} F_L$) we say that $s \underset{(L,R)}{\overset{p}{\leftarrow}} u \underset{(G,D)}{\overset{p\cdot o}{\rightarrow}} t$ is an* overlapping peak of (G, D) onto (L, R) at position o *and that the rules* overlap.

Lemma 3.3.2. *If $s \underset{(L,R)}{\overset{p}{\leftarrow}} u \underset{(G,D)}{\overset{q}{\rightarrow}} t$ then, there are three possibilities:*

- *$p \# q$ (*disjoint peak *case);*
- *$q \geq_{\mathcal{P}} p \cdot F_L$ or $p \geq_{\mathcal{P}} q \cdot F_L$ (*ancestor peak *case);*
- *$p = q \cdot o$ with $o \in \mathcal{FP}os(L)$ or $q = p \cdot o$ with $o \in \mathcal{FP}os(G)$ and the peak is overlapping. (*overlapping peak *case).*

Lemma 3.3.3. *A rewrite rule $L \to R$ does not overlap with β.*

Proof. A pattern L is not an abstraction so $L \to R$ may not overlap onto β at position 1. For all p, if $L|_p = u\ v$ then u is neither an abstraction nor a meta-variable, therefore β does not overlap onto $L \to R$ at position p. □

Definition 3.3.4. *Let $L \to R$ and $G \to D$ be two left-linear rules, $o \in \mathcal{FP}os(L)$ and $\overline{x}$ the variables bound above o in L. Consider $G^{\overline{x}} := G\sigma^{\overline{x}}$ for the* lifting *substitution $\sigma^{\overline{x}}$ such that $\forall X \in \mathcal{Z}^n$, $\sigma^{\overline{x}}(X) := \underline{\lambda}\overline{z}\underline{\lambda}\overline{x}.X'[\overline{z}, \overline{x}]$ for some fresh $X' \in \mathcal{Z}^{n+|\overline{x}|}$.*

If the unification of the closed pre-patterns $\lambda\overline{x}.L|_o = \lambda\overline{x}.G^{\overline{x}}$ has a most general solution σ, then the peak $R\sigma \underset{L\to R}{\overset{\Lambda}{\leftarrow}} L\sigma \underset{G\to D}{\overset{o}{\rightarrow}} L\sigma[D\sigma^{\overline{x}}\sigma]_o$ is called a critical peak *of $G \to D$ onto $L \to R$ at position o. Its associated* critical pair *is $\langle R\sigma, L\sigma[D\sigma^{\overline{x}}\sigma]_o\rangle$.*

Proof. Since $o \in \mathcal{FP}os(L)$, then $(L\sigma)|_o = L|_o\sigma = G\sigma_{\overline{x}}\sigma \longrightarrow D\sigma_{\overline{x}}\sigma$. □

Critical pairs of left-linear rewrite systems are in finite number and computing them is done in polynomial time [FJ19a]. Using standard techniques, we then get the analog of Nipkow's critical pair lemma developed for the case of simply typed higher-order rewrite rules:

Lemma 3.3.5 (Critical peak lemma)**.** *Assume* $s \underset{L\to R}{\overset{p}{\leftarrow}} u \underset{G\to D}{\overset{q}{\rightarrow}} t$ *is an overlapping peak at position* $o \in \mathcal{FP}os(L)$ *such that* $q = p \cdot o$*. Then, there is a critical peak* $s' \underset{i}{\overset{\Lambda}{\leftarrow}} u' \underset{j}{\overset{o}{\rightarrow}} t'$ *and a substitution* θ *such that* $u'\theta = u|_p$*,* $s'\theta = s|_p$ *and* $t'\theta = t|_p$*.*

Proof. As usual we assume that $p = \Lambda$ and $q = o$. By definition of higher-order rewriting, there exists some substitutions γ and σ such that $L\gamma = u$, $G\sigma = u|_o = (L\gamma)|_o$, $s = R\gamma$ and $t = u[D\sigma]_o$. Let $\overline{x}$ be the set of variables bound above o in L. We cannot merge γ and σ into a substitution that can be applied to L as the $\overline{x}$ variables may be in $\mathcal{R}an(\sigma)$. However, we can define σ' such that if $\sigma(X) = \underline{\lambda}\overline{z}.t$, then $\sigma'(X') = \underline{\lambda}\overline{z}\underline{\lambda}\overline{x}.t$ and we have both $\overline{x} \cap \mathcal{R}an(\sigma') = \varnothing$ and $G^{\overline{x}}\sigma' = G\sigma = (L\gamma)|_o$. Since σ' and γ have disjoint domains (note that even in the particular case of $G = L$, we consider the unification of $L|_o$ and $G^{\varnothing}$ which has fresh meta-variables) we can define $\gamma' := \gamma\sigma'$ and have both $L\gamma' = u|_p$ and $G^{\overline{x}}\gamma' = (L\gamma')|_o = u|_q$. Besides, since $o \in \mathcal{FP}os(L)$ and $\mathcal{R}an(\gamma') \cap \overline{x} = \varnothing$ we have $(L\gamma')|_o = L|_o\gamma'$. Therefore, γ' is a solution of the equation $L|_o = G^{\overline{x}}$. Let ω be its most general unifier, there exist a substitution θ such that $\omega\theta = \gamma'$. The critical peak is $s' = R\omega \underset{i}{\overset{\Lambda}{\leftarrow}} L\omega = u' = L\omega[G^{\overline{x}}\omega]_o \underset{j}{\overset{o}{\rightarrow}} L\omega[D\sigma^{\overline{x}}\omega]_o = t$. We check that we have indeed $s'\theta = R\omega\theta = R\gamma' = R\gamma = s$, $u'\theta = L\gamma = u$ and $t'\theta = L\gamma[D\sigma^{\overline{x}}\gamma']_o = u[D\sigma]_o = t$. □

3.3.2 Orthogonal critical peaks

We consider the generalization of overlaps between two single reduction steps to overlaps between two orthogonal reduction steps, as illustrated in Figure 3.2.

Definition 3.3.6. *If* L *is a pre-pattern, we write* $\mathcal{FP}os^*(L) := \mathcal{FP}os(L) \setminus \{\Lambda\}$*.*

Definition 3.3.7 (Orthogonal overlap)**.** *Assume two left-linear rules* (L, R) *and* (G, D) *and sets of positions* P *and* Q *compatible with* F_L *and* F_G *respectively.*

We say that P *and* Q overlap for these rules *iff there is a root position* $o \in P \cup Q$ *such that* $P \subseteq \{o\} \cup Q \cdot \mathcal{FP}os^*(G)$ *and* $Q \subseteq \{o\} \cup P \cdot \mathcal{FP}os^*(L)$*.*

An orthogonal local peak $s \underset{(L,R))}{\overset{P}{\Longleftarrow\!\!\otimes}} u \underset{(G,D)}{\overset{Q}{\otimes\!\!\Longrightarrow}} t$ *is an* overlapping peak at positions P and Q *iff* P *and* Q *are overlapping positions for these rules.*

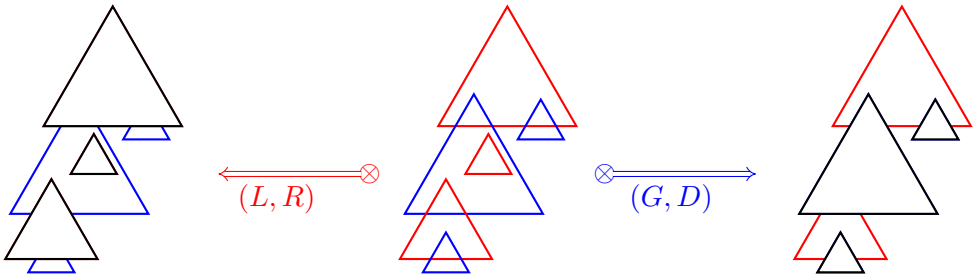


Figure 3.2: Orthogonal critical peak

If P and Q overlap, then any $p \in P$ is either the root, o, of the overlap, or there exists $q \in Q$ strictly above p such that $p \in q \cdot \mathcal{FP}os(G)$. Things are symmetrical for any $q \in Q$.

Note that the overlapping peaks at singleton positions $\{p\}$ and $\{q\}$ are exactly the overlapping peaks at these positions in which case they are called *simple*. If P and Q are both parallel, the overlapping peak is called *parallel*. If $Q = \varnothing$ then $P = \{p\}$ and the peak is *trivial*: it originates from any term matching L at position p.

Lemma 3.3.2 generalizes to orthogonal overlaps. Local peak can be split into either two disjoint steps which can be played in either order or into an above overlap and nested steps below its fringe. If the overlap is trivial, then we are actually in the case of an ancestor peak.

Lemma 3.3.8. *Assume two left-linear rules (L, R) and (G, D) and P and Q compatible with F_L and F_G respectively. Then $P = P_1 \uplus P_2$ and $Q = Q_1 \uplus Q_2$ such that either*

- *$(P_1 \cup Q_1)\#(P_2 \cup Q_2)$ and both sets are strictly smaller in size than $P \cup Q$;*
- *or $(P_2 \cup Q_2) \geq_{\mathcal{P}} (P_1 \cdot F_L \cup Q_1 \cdot F_G)$ and P_1 and Q_1 overlap.*

Proof. If $P \cup Q$ can't be split into parallel strict subsets then necessarily $P \cup Q \geq_{\mathcal{P}} o$ for some $o \in (P \cup Q)$. We define P_1 and Q_1 the smallest subsets of P and Q respectively such that $P \cap (\{o\} \cup Q_1 \cdot \mathcal{FP}os^*(G)) \subseteq P_1$ and $Q \cap (\{o\} \cup P_1 \cdot \mathcal{FP}os^*(L)) \subseteq Q_1$. We define $P_2 := P \setminus P_1$ and $Q_2 := Q \setminus Q_1$. It is clear by definition that P_1 and Q_1 overlap. Let $p \in P_2$. If $p \in P_1 \cdot \mathcal{FP}os(L)$ then P is not compatible with F_L. If $p \in Q_1 \cdot \mathcal{FP}os^*(G)$ then $p \in P_1$. If $p <_{\mathcal{P}} p_1 \in P_1$, we pick the smallest such p_1 and we have $p_1 \in q \cdot \mathcal{FP}os^*(G)$ for some $q \in Q_1$. If $q <_{\mathcal{P}} p$ then $p \in q \cdot \mathcal{FP}os^*(G)$, impossible. Otherwise $p \leq_{\mathcal{P}} q \in p_1' \cdot \mathcal{FP}os^*(L)$ for some $p_1' \in P_1$ in which case $p <_{\mathcal{P}} p_1' <_{\mathcal{P}} p_1$ contradict the minimal choice for p_1 and $p \geq_{\mathcal{P}} p_1'$ implies $p \in P_1 \cdot \mathcal{FP}os(L)$, impossible. This proves $p \geq_{\mathcal{P}} P_1 \cdot F_L$ and $P_2 \geq_{\mathcal{P}} P_1 \cdot F_L$. Similarly, we prove $P_2 \geq_{\mathcal{P}} Q_1 \cdot F_G$, $Q_2 \geq_{\mathcal{P}} P_1 \cdot F_L$ and $Q_2 \geq_{\mathcal{P}} Q_1 \cdot F_G$ and conclude. □

Example 1: Assume two rewrite rules $\mathtt{f}\ X \longrightarrow X$ and $\mathtt{g}\ (\mathtt{f}\ X) \longrightarrow \mathtt{f}\ (\mathtt{g}\ X)$. We have the following local peak: $\mathtt{h}\ x\ (\mathtt{g}\ x) \overset{\{12,22,222\}}{\Longleftarrow\!\!\otimes} \mathtt{h}\ (\mathtt{f}\ x)\ (\mathtt{g}\ (\mathtt{f}\ (\mathtt{f}\ x))) \overset{\{2\}}{\otimes\!\!\Longrightarrow} \mathtt{h}\ (\mathtt{f}\ x)\ (\mathtt{f}\ (\mathtt{g}\ (\mathtt{f}\ x)))$. The two sets of positions can be split into $\{22, 222\} \cup \{2\}$ and $\{12\} \cup \varnothing$ which are parallel and have both strictly less steps. The first pair of sets can in turn be decomposed into $\{22\} \cup \{2\}$ and $\{222\} \cup \varnothing$ such that $\{22\}$ and $\{2\}$ overlap since $22 = 2 \cdot 2$ with 2 a functional position in $\mathtt{g}\ (\mathtt{f}\ X)$ and the remaining steps are below the fringe of the above steps.

Definition 3.3.9. *An overlapping peak $s \underset{(L,R)}{\overset{P}{\Longleftarrow\!\!\otimes}} u \underset{(G,D)}{\overset{Q}{\otimes\!\!\Longrightarrow}} t$ is* minimal *if u is a closed pre-pattern and $\Lambda \in P$.*

Lemma 3.3.10. *If $s \underset{(L,R)}{\overset{O}{\Longleftarrow\!\!\otimes}} u \underset{(G,D)}{\overset{O'}{\otimes\!\!\Longrightarrow}} t$ is an overlapping peak then there exists a minimal overlapping peak $S \underset{(L,R)}{\overset{P}{\Longleftarrow\!\!\otimes}} U \underset{(G,D)}{\overset{Q}{\otimes\!\!\Longrightarrow}} T$ such that $O = o \cdot P$, $O' = o \cdot Q$, $u|_o = U\sigma$, $s = u[S\sigma]_o$ and $s = u[T\sigma]_o$ for some $o \in \mathcal{P}os(u)$ and substitution σ.*

Proof. By definition of local peak, $O = o \cdot P$, $O' = o \cdot Q$ such that $\Lambda \in P$ and since both steps are below o, we have $s|_o \underset{(L,R)}{\overset{P}{\Longleftarrow\!\!\otimes}} u|_o \underset{(G,D)}{\overset{Q}{\otimes\!\!\Longrightarrow}} t|_o$.

We consider the set of positions of free variables and meta-variables in $u|_o$: $R := \bigcup_{x \in \mathcal{V}ar(u|_o)} \mathcal{P}os(x, u|_o) \uplus \mathcal{MP}os(u|_o)$.

We have $P \cdot F_L \not\geq_{\mathcal{P}} R$ and $Q \cdot F_G \not\geq_{\mathcal{P}} R$ since L and G are pre-pattern and cannot match meta-variables or free variables. The meta-terms $U := \underline{u|_o}_R$, $S := \underline{s|_o}_R$ and $T := \underline{t|_o}_R$ are all closed pre-patterns and, by Corollary 3.1.11.1, $S \underset{(L,R)}{\overset{P}{\Longleftarrow\!\!\otimes}} U \underset{(G,D)}{\overset{Q}{\otimes\!\!\Longrightarrow}} T$. Besides, if $\sigma := \overline{u|_o}^R$, they satisfy $U\sigma = u|_o$, $S\sigma = s|_o$ and $T\sigma = t|_o$. □

Definition 3.3.11 (Orthogonal critical pair)**.** *Two rules that overlap at positions $P \ni \Lambda$ and Q have a most general minimal overlapping peak $s \underset{(L,R)}{\overset{P}{\Longleftarrow\!\!\otimes}} u \underset{(G,D)}{\overset{Q}{\otimes\!\!\Longrightarrow}} t$ at these positions called a* critical peak*. The pair (s, t) is the corresponding* orthogonal critical pair*.*

Proof. By induction hypothesis on $|P| + |Q|$.

If $Q = \varnothing$, then the critical peak is simply $R \underset{(L,R)}{\overset{\{\Lambda\}}{\Longleftarrow\!\!\otimes}} L \underset{(G,D)}{\overset{\varnothing}{\otimes\!\!\Longrightarrow}} L$.

Otherwise we choose a position $o \in P \cup Q$ such that $P \cup Q \not\geq_{\mathcal{P}} o \neq \Lambda$. We assume $o \in P$ (the case $o \in Q$ is symmetrical). If the rules overlap at positions P and Q, so do they at positions $P \setminus \{o\}$ and Q and by induction hypothesis, there is a most general minimal overlapping peak $s' \underset{(L,R)}{\overset{P\setminus\{o\}}{\Longleftarrow\!\!\otimes}} u' \underset{(G,D)}{\overset{Q}{\otimes\!\!\Longrightarrow}} t'$.

By assumption, there is an instance of that peak such that $s'\sigma \underset{(L,R)}{\overset{P}{\Longleftarrow\!\!\otimes}} u'\sigma \underset{(G,D)}{\overset{Q}{\otimes\!\!\Longrightarrow}} t'\sigma$ for some closed substitution σ. By definition of an overlapping peak, $o = q \cdot o' \in Q \cdot \mathcal{FP}os(G)$ and since $u'|_q$ matches G, $o \not\geq_{\mathcal{P}} F_{u'}$ and therefore $(u'\sigma)|_o = u'|_o\sigma = L\gamma$ for some substitution γ such that (w.l.o.g.) $\mathcal{V}ar(\gamma) \subseteq \mathcal{V}ar(u'|_o) = \overline{x}$ where $\overline{x}$ are the variables bound above o in u'. The closed pre-patterns $\lambda\overline{x}.u'|_o$ and $\lambda\overline{x}.L^{\overline{x}}$ are therefore unifiable. Their most general unifier θ is such that $s'[R\theta]_o \underset{(L,R)}{\overset{P}{\Longleftarrow\!\!\otimes}} u'\theta \underset{(G,D)}{\overset{Q}{\otimes\!\!\Longrightarrow}} t'\theta$ is a minimal overlapping peak.

It is most general since any overlap at positions P and Q is an instance of the critical peak at positions $P\setminus\{o\}$ and Q such that $u|_o$ and L are unifiable, therefore it is an instance of the built minimal overlapping peak. □

The proof provides an algorithm to enumerate all orthogonal critical pairs which may not be in finite number.

Example 2: The single rewrite rule $\mathtt{F}\ (\mathtt{F}\ X) \longrightarrow X$ has an infinite set of orthogonal

critical pairs with itself:

$$
\begin{array}{ccccc}
Z & \xleftarrow{\{\Lambda\}} & \mathtt{F}\ (\mathtt{F}\ Z) & \xrightarrow{\varnothing} & \mathtt{F}\ (\mathtt{F}\ Z) \\
\mathtt{F}\ Z & \xleftarrow{\{\Lambda\}} & \mathtt{F}\ (\mathtt{F}\ (\mathtt{F}\ Z)) & \xrightarrow{\{2\}} & \mathtt{F}\ Z \\
Z & \xleftarrow{\{\Lambda,22\}} & \mathtt{F}\ (\mathtt{F}\ (\mathtt{F}\ (\mathtt{F}\ Z))) & \xrightarrow{\{2\}} & \mathtt{F}\ (\mathtt{F}\ Z) \\
 & & \cdots & & \\
\mathtt{F}^{(1-(-1)^n)/2}\ Z & \xleftarrow{\{2^{2k} \mid 2k\leq n\}} & \mathtt{F}^n\ Z & \xrightarrow{\{2^{2k+1} \mid 2k+1\leq n\}} & \mathtt{F}^{1+(1+(-1)^n)/2}\ Z
\end{array}
$$

3.4 Non-overlapping local peaks

Even if rewrite rules have no overlapping peaks, we still need to consider the cases of ancestor and disjoints peaks, see Lemma 3.3.2. We prove in this section several commutation lemmas between β-reduction and $\mathcal{R}$-rewriting. While the disjoint case is easy (Lemma 3.4.1) the nested case requires more work. The main property we need to prove is that $\underset{\beta}{\Longleftarrow\!\!\otimes}\ \underset{L\to R}{\longrightarrow} \subseteq \underset{L\to R}{\Longrightarrow}\ \underset{\beta}{\Longleftarrow\!\!\otimes}$ (Lemma 3.4.7). To prove this property we will need to study the cases where β steps occur above (Lemma 3.4.6) and below (Lemma 3.4.3.1) the rewrite step.

There are two kinds of local ancestor peaks, *homogeneous* ones, where reductions are either both β or both higher-order rewriting, and *heterogeneous* ones, which mix both kinds. We analyze here which local ancestor peaks enjoy *decreasing diagrams for free*, and which do not.

In the case of plain rewriting, two non-overlapping rewrite steps issuing from a same term commute, a major component of any confluence proof. When the steps occur at disjoint positions, this property, which holds for any monotonic relation, remains true for rewriting modulo a theory, hence all disjoint peaks have decreasing diagrams for free.

Lemma 3.4.1. *If P and Q parallel sets of such that $P\#Q$, then* $\overset{\geq_{\mathcal{P}} P}{\longleftarrow}\ \overset{\geq_{\mathcal{P}} Q}{\longrightarrow} \subseteq \overset{\geq_{\mathcal{P}} Q}{\longrightarrow}\ \overset{\geq_{\mathcal{P}} P}{\longleftarrow}$.

Proof. Directly follows from Lemma 2.1.27 and monotonicity. □

This is not the case, however, when the steps occur at positions which one is an ancestor of the other, because the above rewrite may interact with and duplicate the rewrite below. Our definition of higher-order rewriting, however, enjoys a similar property, in the particular cases where the fringe of a rewrite step guards the positions of rewrites below it.

In the coming lemmas, "LAP" stands for *linear ancestor peak*. Both simple and **O**rthogonal β-step, **a**bove and **b**elow higher-order rewrite steps are considered.

Lemma 3.4.2 (LAPβa)**.** *Let u be a term, $p, q \in \mathcal{P}os(u)$ such that $q \geq_{\mathcal{P}} p \cdot F_\beta$ and $s \underset{\beta}{\xleftarrow{p}} u \xrightarrow{q} t$. Then $s \overset{Q}{\Longrightarrow} \underset{\beta}{\xleftarrow{p}} t$ for some set Q of parallel positions of s such that $Q \geq_{\mathcal{P}} p$.*

Proof. Note that $\longrightarrow$ can be any monotonic and stable relation on terms. Assuming $p = \Lambda$, we have $u = (\lambda x : A.\, M)\ N$, and $s = M\sigma$, where $\sigma = \{x \mapsto N\}$. There are three cases:

1. $q \geq_{\mathcal{P}} 2$ and $t = (\lambda x : A.\, M)\ P$ with $N \longrightarrow P$. This requires several rewrite steps at the parallel positions of x in M. Then $s = M\sigma \Longrightarrow M\{x \mapsto P\} \underset{\beta}{\overset{\Lambda}{\longleftarrow}} (\lambda x : A.\, M)\ P = t$.
2. $q = 12 \cdot q'$ and $M \xrightarrow{q'} P$. Then, by stability, $s = M\sigma \xrightarrow{q'} P\sigma$ while on the other hand, $t = (\lambda x : A.\, P)\ N \underset{\beta}{\overset{\Lambda}{\longrightarrow}} P\sigma$.
3. $q = 11 \cdot q'$, $A \xrightarrow{q'} A'$ and $t = (\lambda x : A'.\, M)\ N \underset{\beta}{\overset{\Lambda}{\longrightarrow}} M\sigma = s$.

In the case where $p > \Lambda$, we have by definition of positional rewriting $\underline{u}_p = \underline{s}_p = \underline{t}_p$ and $s|_p \underset{\beta}{\overset{\Lambda}{\longleftarrow}} u|_p \underset{L \to R}{\overset{p^{-1}(q)}{\longrightarrow}} t|_p$. We conclude using monotonicity of $\longrightarrow$ and β. □

A similar result holds as well for homogeneous local peaks $s \underset{L \to R}{\overset{p}{\longleftarrow}} u \xrightarrow[\mathcal{R}]{q} t$ where the above higher-order step is left-linear and the below step occurs below its fringe. This property, called (LAP$\mathcal{R}$a), is illustrated in Figure 3.3. As already said, (LAP$\mathcal{R}$a) requires the linearity assumption.

Lemma 3.4.3 (LAP$\mathcal{R}$a)**.** *Let (L, R) be a left-linear rule, u be a term and $p, q \in \mathcal{P}os(u)$ such that $q \geq_{\mathcal{P}} p \cdot F_L$ and $s \underset{(L,R)}{\overset{p}{\longleftarrow}} u \xrightarrow{q} t$. Then $s \overset{\geq_{\mathcal{P}} p}{\Longrightarrow} \underset{(L,R)}{\overset{p}{\longleftarrow}} t$.*

Proof. Assuming $p = \Lambda$, we have by definition, $u = L\gamma \longrightarrow R\gamma = s$. By Lemma 3.1.10, $\gamma \longrightarrow \tau$ and $t = L\tau$. By Lemma 2.2.26, $s = R\gamma \Longrightarrow R\tau$ and by stability, $L\tau \longrightarrow R\tau$.

If $p >_{\mathcal{P}} \Lambda$ then we use the lemma on $s|_p \underset{L \to R}{\overset{\Lambda}{\longleftarrow}} u|_p \overset{p^{-1}(q)}{\longrightarrow} t|_p$ and conclude by stability. □

We move on to the linear ancestor peak properties of orthogonal β-reductions. Unlike the "above case", the "below case" listed first follows easily from Lemma 3.4.2 (LAPβa).

Corollary 3.4.3.1 (LAPOb)**.** *If $\mathcal{R}$ is left-linear and $O >_{\mathcal{P}} q$, then* $\underset{\mathcal{R}}{\overset{q}{\longleftarrow}} \otimes \underset{\beta}{\overset{O}{\Longrightarrow}} \subseteq \otimes \underset{\beta}{\overset{\geq_{\mathcal{P}} q}{\Longrightarrow}} \underset{\mathcal{R}}{\overset{q}{\longleftarrow}}$.

Proof. Assuming $s \underset{L \to R}{\overset{q}{\longleftarrow}} u$ for some $L \to R \in \mathcal{R}$, Since L and F_β do not overlap, $O \geq_{\mathcal{P}} q \cdot F_L$ and we can apply Lemma 3.4.3 (LAP$\mathcal{R}$a). Parallel $\underset{\beta}{\otimes\Longrightarrow}$ steps merge into a single step. □

Definition 3.4.4. *A non-empty set of positions $O \subseteq \mathcal{P}os(u)$ is said to be* rigid in u *if there exists $q \leq_{\mathcal{P}} O$ such that $\mathcal{V}ar(u|_O) \subseteq \mathcal{V}ar(u|_q)$.*

Note that we can always choose for q, the greatest lower bound of O w.r.t. $\leq_{\mathcal{P}}$ and that a position $q \in \mathcal{P}os(u)$ is a singleton set of rigid positions in u.

Example 1: Consider the term $u := (\mathtt{f}\ x)\ (\lambda y.\, (\lambda z.\, (\mathtt{f}\ y)\ (\mathtt{f}\ z))\ (\mathtt{f}\ x))$ and the positions p_1, p_2, p_3 and p_4 corresponding to the successive occurrences of an applied $\mathtt{f}$

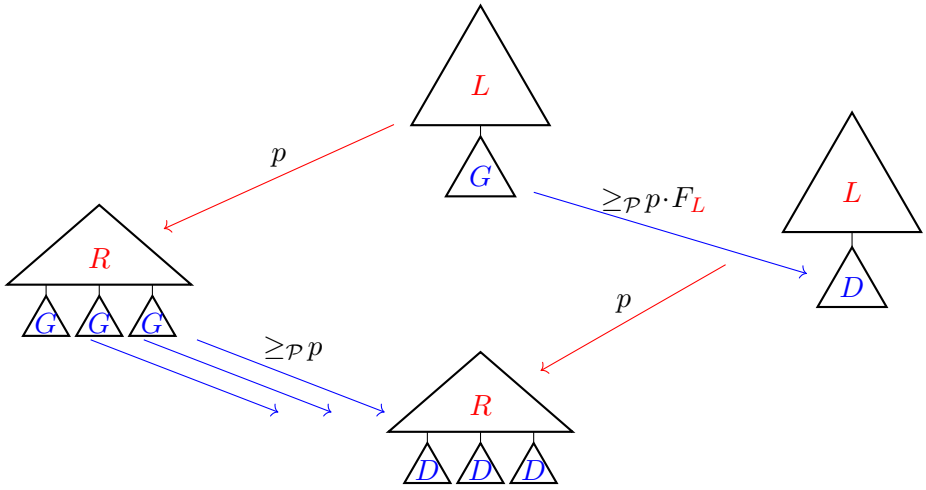


Figure 3.3: Non-overlapping peak of left-linear L →R above G →D: Lemma 3.4.3 (LAP$\mathcal{R}$a).

symbol. The set $\{p_1, p_4\}$ is rigid since the only free variable in $u|_{p_1}$ and $u|_{p_4}$ is x which is free in u. The set $\{p_2, p_4\}$ is rigid too since the variable y is bounded above both positions and so is $\{p_2, p_3\}$ for the same reason. However $\{p_1, p_2\}$ is not rigid since the only possible $q \leq_{\mathcal{P}} \{p_1, p_2\}$ is $q = \Lambda$ and y is not free in $u|_\Lambda = u$. Similarly, $\{p_3, p_4\}$ is not rigid because of the variable z which is free in $u|_{p_3}$ but not bound above p_4. Note that a singleton set of a position of u is always rigid in u.

As illustrated in Figure 3.4, the property for a (set of) meta-variable(s) to not be self-nested is preserved by $\otimes\!\!\Longrightarrow$ if they occur at a rigid set of positions. This property is particularly useful to ensure that rewrite steps at a set of rigid parallel positions remain parallel (but not rigid) when β reduced.

Lemma 3.4.5. *Assume a meta-variable Z not self-nested in u such that $\mathcal{P}os(Z, u)$ is rigid in u. If $u \overset{O}{\underset{\beta}{\otimes\!\!\Longrightarrow}} v$, then Z is not self-nested in v.*

Proof. Since $\mathcal{P}os(Z, u)$ is rigid in u, there exists a position $q \leq_{\mathcal{P}} \mathcal{P}os(Z, u)$ such that if $Z[\overline{x}] \lhd u$, then $\overline{x} \subseteq \mathcal{V}ar(u|_q)$.

We prove the result by induction on the set of positions O using the well-founded multiset extension $\succ_{mul}$ of the usual ordering $\leq_{\mathcal{P}}$ on positions (a set is of course a multiset).

If $O \# q$ then the rewrite steps leaves $u|_q$ and therefore all occurrences of Z in u untouched. If $O \geq_{\mathcal{P}} q$, then the variables $\overline{x}$ occurring in a subterm of $u|_q$ of the form $Z[\overline{x}]$ are all bound above q in u. These subterms may be moved around, deleted, duplicated or reduced by β-steps occurring below q but they are never "substituted into". In particular they cannot become self-nested, therefore Z is not self-nested in v. Otherwise $q >_{\mathcal{P}} O$. There are two cases:

If $O = O_1 \uplus O_2$ for some $O_1 \# O_2$ such that $q >_{\mathcal{P}} O_1$, then Z only occurs below O_1 and

we have $u \overset{O_1}{\underset{\beta}{\otimes\Longrightarrow}} u' \overset{O_2}{\underset{\beta}{\otimes\Longrightarrow}} v$. By induction hypothesis, Z is not self-nested in u' and since $O_1 \# O_2$, neither is it in v.

If $O = \{p\} \uplus O'$ for some $O' >_{\mathcal{P}} p \cdot F_\beta$, If $p >_{\mathcal{P}} \Lambda$, then $u|_p \overset{p^{-1}(O)}{\underset{\beta}{\otimes\Longrightarrow}} v'$ and by induction hypothesis, Z is not self-nested in v' and since $v = u[v']_p$ and Z only occurs below p in u neither is it in v. Assuming $p = \Lambda$, then $u = (\lambda x\!:\!A.\, M)\ N$ and, by Lemma 3.1.8, $O = \{\Lambda\} \uplus 11 \cdot O_A \uplus 12 \cdot O_M \uplus 2 \cdot O_N$, where $M \overset{O_M}{\underset{\beta}{\otimes\Longrightarrow}} v_M$, $M \overset{O_N}{\underset{\beta}{\otimes\Longrightarrow}} v_N$ and $v = v_M\{x \mapsto v_N\}$.

The meta-variable Z only occurs in $u|_q$ which is a subterm of either A, M or N.

- If $q \geq_{\mathcal{P}} 11$, then all occurrences of Z are in A and are deleted by the β step.
- If $q \geq_{\mathcal{P}} 12$, then by induction hypothesis, occurrences of Z in v_M are not self-nested. Since $Z \notin \mathcal{MV}ar(N)$, $Z \notin \mathcal{MV}ar(v_N)$ and Z is not self-nested in v either.
- If $q \geq_{\mathcal{P}} 2$, then by induction hypothesis, occurrences of Z in v_N are not self-nested. Since all occurrences of Z in v are duplicated versions of the ones in v_N they are not self-nested either. □

Lemma 3.4.6 (LAPOa)**.** *If $O \not\geq_{\mathcal{P}} q$ then* $\overset{O}{\underset{\beta}{\Longleftarrow\!\!\otimes}} \xrightarrow[L\to R]{q} \subseteq \overset{\geq_{\mathcal{P}}(O \cup \{q\})}{\underset{L\to R}{\Longrightarrow}} \overset{O}{\underset{\beta}{\Longleftarrow\!\!\otimes}}$.

Proof. Assume $s \overset{O}{\underset{\beta}{\Longleftarrow\!\!\otimes}} u \xrightarrow[L\to R]{q} t$. By Corollary 3.1.11.1, since $O \not\geq_{\mathcal{P}} q$, $s' \overset{O}{\underset{\beta}{\Longleftarrow\!\!\otimes}} \underline{u}_q$ and $s = s'\overline{u}^q$. By Lemma 3.1.10, $\overline{u}^q \xrightarrow[L\to R]{} \sigma$ such that $t = \underline{u}_q\sigma$. and by stability, $s'\sigma \overset{O}{\underset{\beta}{\Longleftarrow\!\!\otimes}} t$. Besides $\underline{u}_q = u[Z[\overline{z}]]_q$ so $\mathcal{P}os(Z, \underline{u}_q) = \{q\}$ is rigid in $\underline{u}_q$ and Z is obviously not self-nested in $\underline{u}_q$. Therefore, by Lemma 3.4.5, Z is not self-nested in s' and, by stability, $s = s'\overline{u}^q \overset{\mathcal{P}os(Z,s')}{\Longrightarrow} s'\sigma$. □

Lemma 3.4.7 (LAPO)**.** *If $L \to R$ is a left-linear rule, then* $\overset{}{\underset{\beta}{\Longleftarrow\!\!\otimes}} \xrightarrow[L\to R]{} \subseteq \underset{L\to R}{\Longrightarrow} \overset{}{\underset{\beta}{\Longleftarrow\!\!\otimes}}$.

Proof. The proof is illustrated in Figure 3.5. We assume $s \overset{O}{\Longleftarrow\!\!\otimes}\ u \xrightarrow[i]{q} t$. By Lemma 3.1.7 we can sequentialize the β-step into $s \overset{R}{\Longleftarrow\!\!\otimes} v \overset{Q}{\Longleftarrow\!\!\otimes} u' \overset{P}{\Longleftarrow\!\!\otimes} u$ with $O = P \uplus Q \uplus R$ such that $P \# q$, $Q \geq_{\mathcal{P}} q$ and $R <_{\mathcal{P}} q$. Since β and rules do not overlap, $q \notin O$ and $Q \geq_{\mathcal{P}} q \cdot F_L$. By commutation we easily get $u' \xrightarrow[L\to R]{q} t' \overset{P}{\underset{\beta}{\Longleftarrow\!\!\otimes}} t$. By Lemma 3.4.3.1 we have $v \xrightarrow[L\to R]{q} r \overset{\geq_{\mathcal{P}} q}{\underset{\beta}{\Longleftarrow\!\!\otimes}} t'$ and by Lemma 3.4.6 we have $s \underset{L\to R}{\Longrightarrow} w \overset{R}{\underset{\beta}{\Longleftarrow\!\!\otimes}} r$. Finally, by Lemma 3.1.7 again, all three β-steps can be merged into a single orthogonal $w \overset{}{\underset{\beta}{\Longleftarrow\!\!\otimes}} t$. □

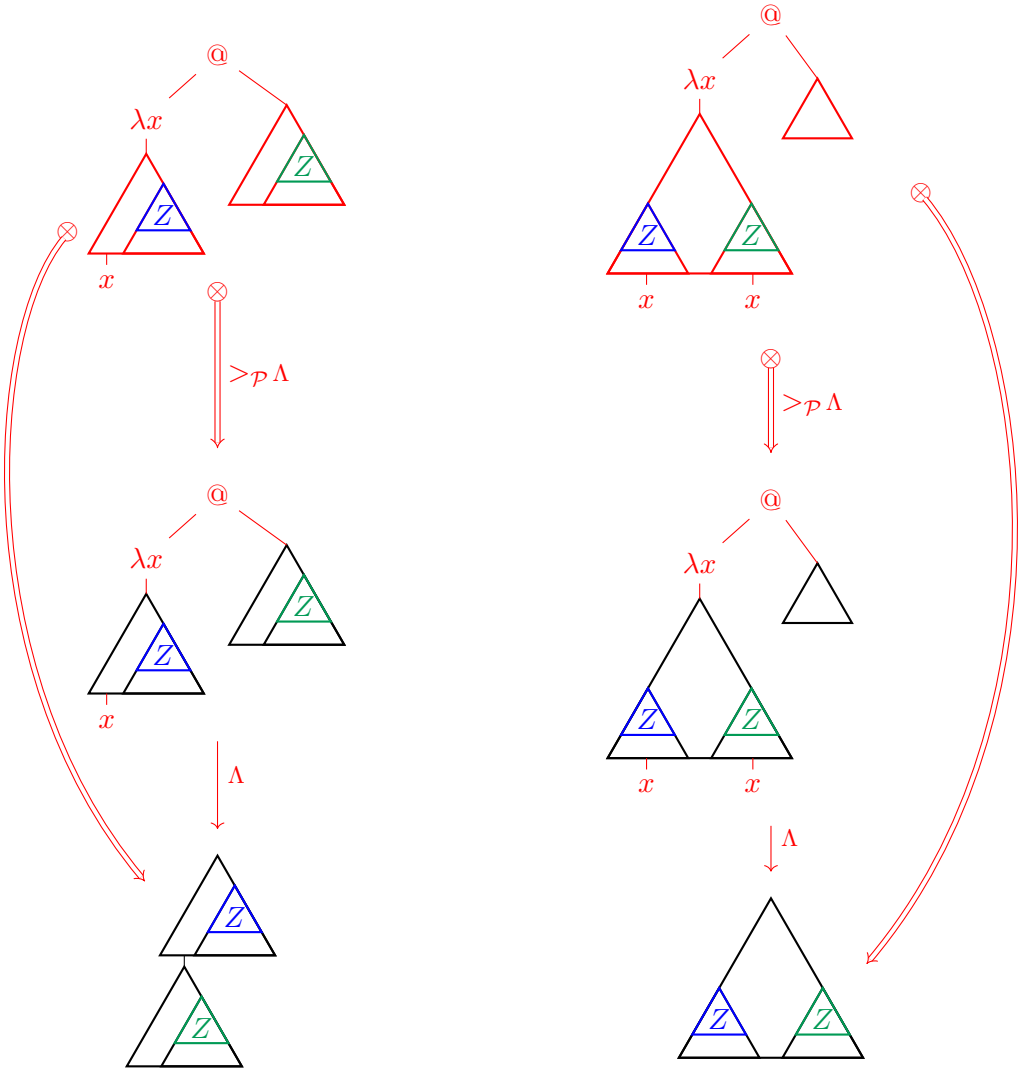


Figure 3.4: Preservation of unnestedness with β: Lemma 3.4.5

3.5 Confluence of rewriting

The commutation of simple $\mathcal{R}$-rewriting step and orthogonal β-reduction, as stated in Lemma 3.4.7, only provides a decreasing diagram if β steps are labeled above $\mathcal{R}$ since a multi-step is facing the $\mathcal{R}$ single step. If we try to extend it to parallel $\mathcal{R}$-rewriting, then we need to consider an orthogonal step to close : $\underset{\beta}{\Longleftarrow\!\!\otimes}\ \underset{L\to R}{\Longrightarrow} \subseteq \otimes\!\!\underset{L\to R}{\Longrightarrow}\ \underset{\beta}{\Longleftarrow\!\!\otimes}$. However, when considering orthogonal $\mathcal{R}$-rewriting, then we get a commutation diagram, see Lemma 3.5.1. In fact it generalizes to orthogonal reduction with any two non-overlapping sets of left-linear rules. This strong commutation property allows to deduce two crucial properties: modularity of confluence for non-overlapping systems [vOvR94] and confluence of systems

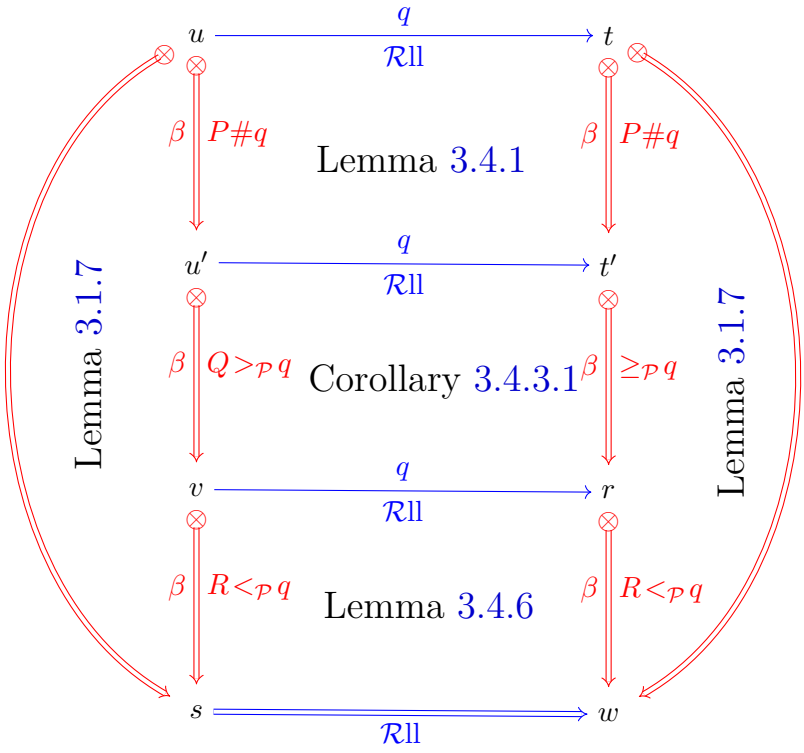


Figure 3.5: Construction of a decreasing diagram for heterogeneous local peaks. See Lemma 3.4.7

without critical pairs [Ros73].

In presence of overlaps between the rewrite rules of $\mathcal{R}$, confluence can only be obtained if critical pairs are joinable. However, this property suffices to guarantee local confluence but we need extra hypotheses in order to achieve confluence. In the case where $\mathcal{R}$ defines a strongly normalization relation, Newman's lemma allows to conclude, [New42].

In the case of non-terminating systems, we can still prove confluence provided the rules' right-hand sides are not self-nested, as defined in Definition 2.2.25 and therefore behave like first-order rewrite rules. In that case we also require the rules to be labeled such that the critical pairs are joined with a decreasing diagram and extra conditions on critical peaks akin to Toyama's Variable Condition.

While the above criteria already allow to prove the confluence of many rewrite systems, they fail to do so in the case where critical pairs of rewrite rules require β-steps to be joined. To build a decreasing diagram for such cases, it is necessary to label (orthogonal) β-steps with a lower label than rewriting which, in turn, forces to consider orthogonal rewrite steps and therefore orthogonal critical pairs.

3.5.1 Confluence of non-overlapping systems

Lemma 3.5.1. *Assume $\mathcal{R}_1$ and $\mathcal{R}_2$ two sets of left-linear rules such that rules of $\mathcal{R}_1$ do not overlap with rules of $\mathcal{R}_2$. Then $\underset{\mathcal{R}_1}{\otimes\!\!\Longrightarrow}$ and $\underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow}$ sub-commute: $\underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes}\,\underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow} \subseteq \underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow}\,\underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes}$.*
More precisely, if $s \overset{P}{\underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes}} u \overset{Q}{\underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow}} t$ then $s \overset{\geq_{\mathcal{P}} P\cup Q}{\underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow}} v \overset{\geq_{\mathcal{P}} P\cup Q}{\underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes}} t$ and $u \overset{P\cup Q}{\underset{\mathcal{R}_1\cup\mathcal{R}_2}{\otimes\!\!\Longrightarrow}} v$.
In particular, $\underset{\mathcal{R}_1}{\longrightarrow}$ and $\underset{\mathcal{R}_2}{\longrightarrow}$ commute.

Proof. Assuming $s \underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes} u \underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow} t$, we prove, as in Lemma 3.4.5, by induction on O using $\succ_{mul}$, that $s \underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow} v \underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes} t$ for some v.

If $\Lambda \notin P \cup Q$, then assuming, for instance, $u = X[\overline{u}]$ (the $u = u_1\ u_2$ and $u = \lambda x : u_1.\, u_2$ cases are similar), we have $P = \uplus\, i \cdot P_i$, $Q = \uplus\, i \cdot Q_i$ and $s_i \underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes} u_i \underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow} t_i$ such that $s = X[\overline{s}]$ and $t = X[\overline{t}]$. Induction hypothesis gives $s_i \underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow} v_i \underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes} t_i$ and $u_i \underset{\mathcal{R}_1\cup\mathcal{R}_2}{\otimes\!\!\Longrightarrow} v_i$ which can be merged into that $s \underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow} v \underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes} t$ and $u \underset{\mathcal{R}_1\cup\mathcal{R}_2}{\otimes\!\!\Longrightarrow} v$ for $v := X[\overline{v}]$.

Otherwise, w.l.o.g. $P = \{\Lambda\} \uplus P'$ and, by definition, $s = R\gamma \overset{\Lambda}{\underset{\mathcal{R}_1}{\longleftarrow}} L\gamma \overset{P'}{\underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes}} u$ for some $(L, R) \in \mathcal{R}_1$ and $P' \geq_{\mathcal{P}} F_L$. By assumption we also have $Q \geq_{\mathcal{P}} F_L$ otherwise there would be a $\mathcal{R}_2$-redex at position $q \in \mathcal{FP}os(L)$, an overlapping peak. By induction hypothesis, $L\gamma \overset{\geq_{\mathcal{P}} F_L}{\underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow}} v' \overset{\geq_{\mathcal{P}} F_L}{\underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes}} t$ and $u \overset{\geq_{\mathcal{P}} F_L}{\underset{\mathcal{R}_1\cup\mathcal{R}_2}{\otimes\!\!\Longrightarrow}} v'$. By Lemma 3.1.10, since L linear, $v' = L\tau$ for τ such that $\gamma \underset{\mathcal{R}_1}{\otimes\!\!\Longrightarrow} \tau$. We have $s = R\gamma \underset{\mathcal{R}_2}{\otimes\!\!\Longrightarrow} R\tau = v$ and we group together $u \overset{\geq_{\mathcal{P}} F_L}{\underset{\mathcal{R}_1\cup\mathcal{R}_2}{\otimes\!\!\Longrightarrow}} L\tau \underset{\mathcal{R}_1\cup\mathcal{R}_2}{\otimes\!\!\Longrightarrow} R\tau$ into $u \underset{\mathcal{R}_1\cup\mathcal{R}_2}{\otimes\!\!\Longrightarrow} v$ and $R\tau \underset{\mathcal{R}_1}{\longleftarrow} L\tau \overset{\geq_{\mathcal{P}} F_L}{\underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes}} t$ into $v \underset{\mathcal{R}_1}{\Longleftarrow\!\!\otimes} t$. □

A simple corollary is the modularity of confluence when two left-linear systems do not overlap, [vOvR94].

Corollary 3.5.1.1. *Assume $\mathcal{R}_1$ and $\mathcal{R}_2$ two sets of left-linear the rules such that rules of $\mathcal{R}_1$ do not overlap with the rules of $\mathcal{R}_2$ and such that $\xrightarrow[\mathcal{R}_1]{}$ and $\xrightarrow[\mathcal{R}_1]{}$ are both confluent. Then $\xrightarrow[\mathcal{R}_1\cup\mathcal{R}_2]{}$ is confluent.*

Left-linear system without critical pair are sometimes called *orthogonal* even though we will steer clear of this denomination to avoid confusion with orthogonal relations. Their confluence follows easily from Lemma 3.5.1.

Lemma 3.5.2. *If $\mathcal{R}$ is a left-linear system without critical pair (for instance $\mathcal{R} = \{\beta\}$). Then $\otimes\!\!\xrightarrow[\mathcal{R}]{}$ has the diamond property and therefore $\xrightarrow[\mathcal{R}]{}$ is confluent.*

Proof. By Lemma 3.5.1 with $\mathcal{R}_1 = \mathcal{R}_2 = \mathcal{R}$. □

Theorem 3.5.3. *Assume $\mathcal{R}$ a left-linear rewrite system such that $\xrightarrow[\mathcal{R}]{}$ is confluent. Then $\xrightarrow[\mathcal{R}]{} \cup \xrightarrow[\beta]{}$ (written $\xrightarrow[\mathcal{R}\cup\beta]{}$ or $\xrightarrow[\mathcal{R}\beta]{}$) is confluent.*

Proof. By Lemma 3.5.2, $\otimes\!\!\xrightarrow[\beta]{}$ is confluent, so is $\xrightarrow[\mathcal{R}]{}$ and by Lemma 3.4.7, they commute. Hence they are confluent together and since $\xrightarrow[\mathcal{R}\beta]{} \subseteq \xrightarrow[\mathcal{R}]{} \cup \otimes\!\!\xrightarrow[\beta]{} \subseteq \xrightarrow[\mathcal{R}\beta]{}\!\!\!\!\!\rightarrow$, so is $\xrightarrow[\mathcal{R}\beta]{}$. □

Corollary 3.5.3.1. *Left-linear rewrite systems without critical pair are confluent together with β.*

Example 1: For instance, the untyped lambda calculus can be encoded with two symbols $\texttt{lam} : (\texttt{T} \to \texttt{T}) \to \texttt{T}$ and $\texttt{app} : \texttt{T} \to \texttt{T} \to \texttt{T}$ for some declared type $\texttt{T} : *$ in the signature. The β rule is represented with the rule $\texttt{app}\ (\texttt{lam}\ \lambda x : \texttt{T}.\, F[x])\ T \to F[T]$ which is a single rule left-linear system without critical pair, therefore confluent with β.

3.5.2 Confluence of terminating rewrite systems

Lemma 3.5.4. *If $\mathcal{R}$ is a left-linear set of rules which critical pairs are joinable with $\xrightarrow[\mathcal{R}]{}$, then $\xrightarrow[\mathcal{R}]{}$ is locally confluent: $\xleftarrow[\mathcal{R}]{}\xrightarrow[\mathcal{R}]{} \subseteq \xrightarrow[\mathcal{R}]{}\!\!\!\!\!\rightarrow \xleftarrow[\mathcal{R}]{}\!\!\!\!\!\leftarrow$.*

Proof. Assume $s \xleftarrow[\mathcal{R}]{p} u \xrightarrow[\mathcal{R}]{q} t$.

If $p \# q$, then the steps commute by Lemma 3.4.1: $s = u[s']_p \xrightarrow[\mathcal{R}]{q} u[s']_p[t']_q \xleftarrow[\mathcal{R}]{q} u[t']_p = t$.

If $q \geq_{\mathcal{P}} p \cdot F_L$, then Lemma 3.4.3 (LAP$\mathcal{R}$a) gives the result.

If $q = p \cdot o$ and $o \in \mathcal{FP}os(F_L)$. By Lemma 3.3.5, there is a critical peak obtained by overlapping G onto L at position o. By assumption, this peak is joinable with rules of $\mathcal{R}$, hence the pair s, t is joinable by monotonicity and stability, Lemma 2.2.16. □

Theorem 3.5.5. *If $\mathcal{R}$ is a left-linear, terminating rewrite system which all critical pairs are joinable with $\mathcal{R}$, then $\underset{\mathcal{R}\beta}{\longrightarrow}$ is confluent.*

Proof. By Lemma 3.5.4, $\underset{\mathcal{R}}{\longrightarrow}$ is locally confluent and since it is also terminating it is confluent by Newman's lemma. We conclude by Theorem 3.5.3. The labeling allowing to have decreasing diagrams for all local peak relies on self-labeling for rewriting steps: $u \underset{\mathcal{R}}{\longrightarrow} v$ is labeled with u and these labels are ordered with $\underset{\mathcal{R}}{\longrightarrow}$ with is well-founded by assumption. The functional reduction $\underset{\beta}{\otimes\Longrightarrow}$ is labeled with an extra constant label which is bigger than all labels of $\mathcal{R}$. □

3.5.3 Confluence by decreasing diagram

Rewrite rules which right-hand sides are not self-nested, see Definition 2.2.25, allow to consider parallel rewriting, and therefore parallel critical pairs in the study of their critical peaks.

Definition 3.5.6. *A rewrite rule $L \to R$ is called* self-nested *if its right-hand side, R, is.*

Definition 3.5.7. *Given a parallel critical peak $s \underset{L\to R}{\overset{\Lambda,m}{\Longleftarrow}} u \overset{P,n}{\Longrightarrow} t$, its decreasing diagram is* strict *if the decreasing sequence issuing from s has the form $s \overset{\geq_{\mathcal{P}} Q}{\twoheadrightarrow} \overset{\geq_{\mathcal{P}} Q,n}{\Longrightarrow} \twoheadrightarrow v$ where Q is a set of positions such that $\mathcal{MV}ar(R|_Q) \cap (\mathcal{MV}ar(L) \setminus \mathcal{MV}ar(L|_P)) = \varnothing$.*

The above condition restrict the set of positions Q to those that do not interfere with the rewrites below meta-variables of R that are not in the scope of P. It is an adaptation of Toyama's so-called Variable Condition (TVC) [Toy81] introduced in a first-order setting.

Example 2: Assume the following rewrite systems:

$$\begin{array}{lcl} \mathcal{R}_1 & : & \{ \quad \mathtt{g}\ X \to X \quad , \quad \mathtt{f}\ (\mathtt{g}\ X)\ Y \to \mathtt{g}\ \mathtt{f}\ (\mathtt{g}\ X)\ Y \quad \} \\ \mathcal{R}_2 & : & \{ \quad \mathtt{g}\ X \to X \quad , \quad \mathtt{f}\ (\mathtt{g}\ X)\ Y \to \mathtt{f}\ X\ (\mathtt{g}\ Y) \quad \} \end{array}$$

Both have a single parallel critical peak, $\mathtt{g}\ \mathtt{f}\ (\mathtt{g}\ X)\ Y \longleftarrow \mathtt{f}\ (\mathtt{g}\ X)\ Y \overset{\{12\}}{\longrightarrow} \mathtt{f}\ X\ Y$ which is simple. In order to have a strict decreasing diagram for this peak, the variable Y must not interfere with the left-hand decreasing sequence. In the case of $\mathcal{R}_1$, it is not the case since $\mathtt{g}\ \mathtt{f}\ (\mathtt{g}\ X)\ Y \overset{Q}{\Longrightarrow} \mathtt{f}\ X\ Y$ with $Q = \{12, 11\}$ such that $Y \notin R|_Q$. However, in the case of $\mathcal{R}_2$, $\mathtt{f}\ X\ (\mathtt{g}\ Y) \overset{\{2\}}{\Longrightarrow} \mathtt{f}\ X\ Y$ is not strict. Note that in both cases it is also possible to label the first relation with a lower label than the second and consider the joining sequence as the middle step which is unconstrained. Other rules in the system may however forbid this labeling.

In systems satisfying this condition, it is possible to prove confluence using parallel reduction.

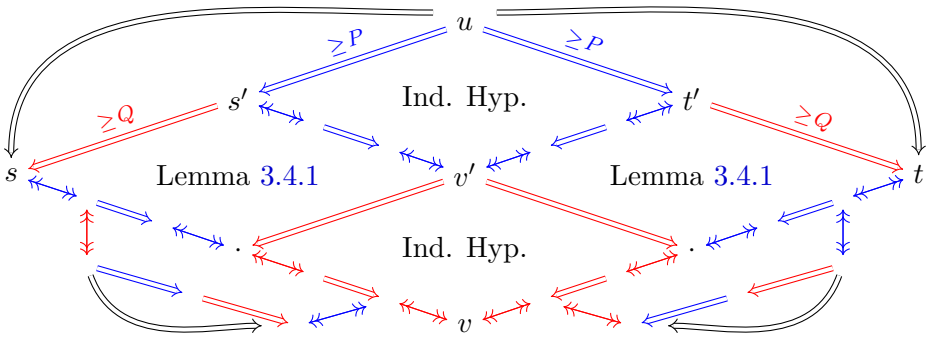


Figure 3.6: Parallel decreasing diagrams merging in Theorems 3.5.8 and 3.5.9

Theorem 3.5.8. *If $\mathcal{R}$ is a set of labeled left-linear rewrite rules which parallel critical pairs have strict decreasing diagrams with $\underset{\mathcal{R}}{\longrightarrow}$ and no rule in $\mathcal{R}$ is self-nested. Then $\underset{\mathcal{R}}{\longrightarrow}$ and $\underset{\mathcal{R}\beta}{\longrightarrow}$ are confluent.*

Proof. We extend the labels of simple rewriting $\underset{L\to R}{\overset{l}{\longrightarrow}}$ to parallel rewrite steps $\underset{L\to R}{\overset{l}{\Longrightarrow}}$. For all local peaks that can be split into steps at parallel subsets of positions, we conclude with a diamond diagram, exactly as before.

In the case $s \underset{L\to R}{\overset{p,m}{\Longleftarrow}} u \underset{G\to D}{\overset{p\cdot P,n}{\Longrightarrow}} t$, we use the assumptions to guarantee that the parallel facing step can be merged with parallel steps nested below the overlap. Assuming $p = \Lambda$, we have a critical peak $s' \underset{L\to R}{\overset{\Lambda,m}{\Longleftarrow}} u' \underset{G\to D}{\overset{Q,n}{\Longrightarrow}} t'$ and a substitution θ such that $u = u'\theta$ and $\theta \overset{n}{\Longrightarrow} \tau$. However, because the right-hand step is parallel $u = u'\theta \underset{G\to D}{\overset{Q,n}{\Longrightarrow}} t'\tau = t$, necessarily we have $\mathcal{D}om(\theta) \subseteq \mathcal{MV}ar(u') \setminus \mathcal{MV}ar(u'|_P) = \mathcal{MV}ar(L) \setminus \mathcal{MV}ar(L|_P)$. By assumption, the right-hand side closing decreasing sequence in $s' \overset{\geq_{\mathcal{P}} Q}{\Longrightarrow\!\!\!\!\!\!\rightarrow} v' \overset{Q,n}{\Longrightarrow} w' \overset{DS(m,n)}{\Longleftarrow\!\!\!\!\!\!\leftarrow} t'$ preserves the positions of the meta-variables $\mathcal{D}om(\theta)$ so that we have the following decreasing diagram for the local peak $s'\theta \overset{\geq_{\mathcal{P}} Q}{\Longrightarrow\!\!\!\!\!\!\rightarrow} v'\theta \overset{Q',n}{\Longrightarrow} v'\tau \overset{Q,n}{\Longrightarrow} w'\tau \overset{DS(m,n)}{\Longleftarrow\!\!\!\!\!\!\leftarrow} t'\tau = t$ using, stability and Lemma 2.2.26, since v' not self-nested. The parallel steps can be grouped. □

3.5.4 Higher-Order confluence by decreasing diagrams

Theorem 3.5.9. *Let $\mathcal{R}$ be a left-linear labeled rewrite system which orthogonal critical pairs have decreasing diagrams with $\underset{\mathcal{R}\beta}{\otimes\!\!\Longrightarrow}$, then $\underset{\mathcal{R}\beta}{\longrightarrow}$ is confluent.*

Proof. Assume $L \to R$ is labeled with l, we extend the labeling of simple rewriting $\underset{L\to R}{\overset{l}{\longrightarrow}}$ to orthogonal steps $\underset{L\to R}{\overset{l}{\otimes\!\!\Longrightarrow}}$, similarly to the proof of Theorem 3.5.8. Assume a local peak $s \underset{L\to R}{\overset{m,P}{\Longleftarrow\!\!\otimes}} u \underset{G\to D}{\overset{n,Q}{\otimes\!\!\Longrightarrow}} t$ for some labels m and n, we show, by induction on $|P|+|Q|$ that this peak

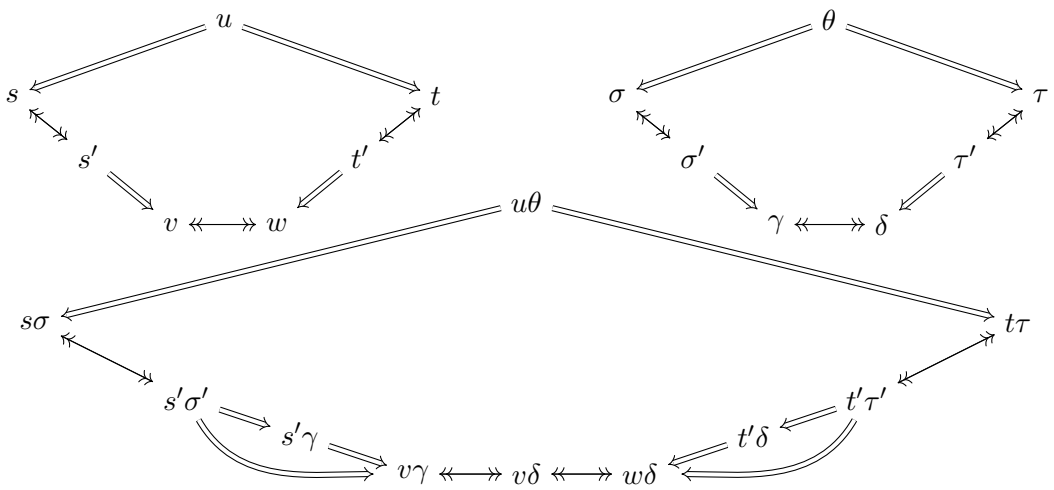


Figure 3.7: Nested decreasing diagrams merging in Theorem 3.5.9

has a decreasing diagram containing exclusively rewrite steps at positions below $P \cup Q$. By Lemma 3.3.8, $P = P_1 \uplus P_2$, $Q = Q_1 \uplus Q_2$ such that we are in one of the following cases:

- (Disjoint peak) $(P_1 \cup Q_1)\#(P_2 \cup Q_2)$, as illustrated in Figure 3.6. The peak can be split into $s \underset{L\to R}{\overset{m,\geq_{\mathcal{P}} P_2}{\Longleftarrow\!\!\otimes}} s' \underset{L\to R}{\overset{m,\geq_{\mathcal{P}} P_1}{\Longleftarrow\!\!\otimes}} u \underset{G\to D}{\overset{n,\geq_{\mathcal{P}} Q_1}{\otimes\!\!\Longrightarrow}} t' \underset{G\to D}{\overset{n,\geq_{\mathcal{P}} Q_2}{\otimes\!\!\Longrightarrow}} t$. By induction hypothesis, there is a decreasing diagram joining s' and t'. By commutation of steps at orthogonal positions (Lemma 3.4.1) we have $s \overset{DS(n,m),\geq_{\mathcal{P}} Q_1}{\longrightarrow\!\!\!\!\!\!\rightarrow} \overset{m,\geq_{\mathcal{P}} P_2}{\Longleftarrow\!\!\otimes} v' \overset{n,\geq_{\mathcal{P}} Q_2}{\otimes\!\!\Longrightarrow} \overset{DS(m,n),\geq_{\mathcal{P}} P_1}{\longrightarrow\!\!\!\!\!\!\rightarrow} t$. The middle local peak has a decreasing diagram by induction hypothesis. We conclude by permuting steps at parallel positions and merging, on each side, the facing steps from the two induction hypothesis together.
- (Ancestor or overlapping peak) $(P_2 \cup Q_2) \geq_{\mathcal{P}} (P_1 \cdot F_L \cup Q_1 \cdot F_G)$ and P_1 and Q_1 overlap as illustrated in Figure 3.7. By Lemma 3.3.10 and Definition 3.3.7, there is a critical peak $s' \overset{P_1}{\Longleftarrow\!\!\otimes} u' \overset{Q_1}{\otimes\!\!\Longrightarrow} t'$ and θ such that $u = u'\theta$. Since $P_2 \geq_{\mathcal{P}} F_{u'}$, by Lemma 3.1.9, the local peak can be decomposed as $s = s'\sigma \underset{L\to R}{\overset{m,P_1}{\Longleftarrow\!\!\otimes}} u'\sigma \underset{L\to R}{\overset{m}{\Longleftarrow\!\!\otimes}} u'\theta \underset{G\to D}{\overset{n,}{\otimes\!\!\Longrightarrow}} u'\tau \underset{G\to D}{\overset{n,Q_1}{\otimes\!\!\Longrightarrow}} t'\tau$. By induction hypothesis, there are sequences $s' \overset{DS(m,n)}{\longrightarrow\!\!\!\!\!\!\rightarrow} v' \overset{DS(n,m)}{\longleftarrow\!\!\!\!\!\!\leftarrow} t'$ and $\sigma \overset{DS(m,n)}{\longrightarrow\!\!\!\!\!\!\rightarrow} \gamma \overset{DS(n,m)}{\longleftarrow\!\!\!\!\!\!\leftarrow} \tau$. By stability and monotonicity, these steps can be rearranged, and facing steps merged, into a decreasing diagram for the local peak. □

Chapter 4

Confluence of Non-Left-Linear Systems

The usual techniques for showing confluence of untyped (higher-order) theories are restricted to systems of left-linear rules. In this chapter, we consider the case of rewrite rules which left-hand sides may be non-linear and study syntactical restrictions that allow to have confluence, even in presence of (restricted) functional rewriting and/or non-terminating systems. Rewriting with rules that are not left-linear may seem inoffensive at first sight. However, most standard properties simply fail.

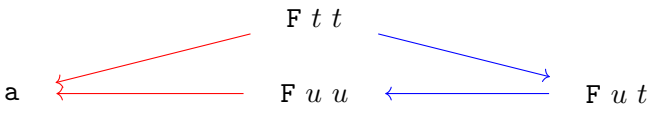
For instance a counter-example of termination of the simply typed λ-calculus together with non-linear rules is given in [Oka89]. Numerous counter-examples to confluence in the (non-terminating) pure λ-calculus, due to Klop, are given in [Klo80]. A striking one is provided by the simple rule $\mathtt{f}\ X\ X \to \mathtt{a}$, which interaction with a fixpoint combinator generates diverging reductions that cannot be joined: confluence of the pure λ-calculus can't be preserved when adding rewrite rules which left-hand sides are non-linear, even in cases where local confluence can be shown.

Example 1: We are, in particular, interested in the following, relatively simple, examples of non-left-linear systems which can prove useful in the context of the encoding of universes

$$\mathtt{F}\ X\ X \to \mathtt{a} \qquad\qquad \mathtt{u}\ X\ (\mathtt{c}\ X\ Y) \to Y$$
$$\mathtt{F}\ X\ X\ Y \to Y \qquad\qquad \mathtt{max}\ X\ X \to X$$

The main reason for non-left-linear rules to be so ill-behaved is that left-linearity guarantees that nested rewrite steps do not interfere with the above redexes: if $L\sigma \xrightarrow{\geq_{\mathcal{P}} F_L} u$ then u still matches L. For instance, if $\mathtt{f}\ X \longrightarrow \mathtt{g}$, then $\mathtt{f}\ (\mathtt{f}\ t) \longrightarrow \mathtt{f}\ \mathtt{g}$ and both left- and right-hand side match the rule at the head with $\{X \mapsto (\mathtt{f}\ t)\}$ and $\{X \mapsto \mathtt{g}\}$ respectively. Steps below the fringe of a redex are played in the matching substitution. In fact left-linearity even allows steps that do not overlap to commute, yielding some crucial confluence modularity properties. Without left-linearity the previous properties

no longer hold. Consider for instance a rewrite system containing a non-linear rewrite rule $\mathtt{f}\ X\ X \to \mathtt{a}$ and a term t such that $t \longrightarrow u$. Then the following non-overlapping ancestor peak no longer commutes.



Besides, its joining sequence cannot be labeled so as to be a decreasing diagram, assuming the labels for $\longrightarrow$ and $\longrightarrow$ depend on redexes. Indeed, while the red joining step $\mathtt{a} \longleftarrow \mathtt{F}\ u\ u$ has the same label as the facing $\mathtt{a} \longleftarrow \mathtt{F}\ t\ t$, the blue joining step must be of strictly lower label than the preceding $\mathtt{F}\ t\ t \longrightarrow \mathtt{F}\ u\ t$ which is not the case here.

A natural way to make this diagram decreasing is to merge together the two joining steps into a single $\mathtt{a} \longleftarrow \mathtt{F}\ u\ t$ facing step. For instance if we consider the extension, $\longmapsto$, of $\longrightarrow$ such that $\mathtt{F}\ u\ t \longmapsto \mathtt{a}$ for all $t \twoheadleftarrow\!\!\!\!\!\!\twoheadrightarrow u$, then $\longmapsto$ is still stable and monotonic but this time, if $\mathtt{a} \longmapsfrom \mathtt{F}\ u\ t \longrightarrow \mathtt{F}\ u'\ t'$, then $\mathtt{a} \longmapsfrom \mathtt{F}\ u'\ t'$ which is decreasing. In fact it can be shown that $\Longrightarrow$ and $\otimes\!\!\Longrightarrow$ commute if $\longrightarrow$ has no critical pair with *the linearized version of* $\longmapsto$. As noted in [Klo80], this means that, in particular, $\Longrightarrow \cup \underset{\beta}{\longrightarrow}$ is confluent. In this chapter we are going to consider a more conservative extension of $\longmapsto$ such that $\mathtt{F}\ u\ t \rhd\!\!\longrightarrow \mathtt{a}$ if $u \longrightarrow\!\!\!\!\!\!\twoheadrightarrow \twoheadleftarrow\!\!\!\!\!\!\longleftarrow t$. The joining sequence is called an *equalization* [JL12b], occurs exclusively below non-linear positions and its labels are strictly smaller than that of the above non-linear rewrite step.

Our main contribution is the description of conditions under which sets of higher-order rewrite rules preserve confluence of the λ-calculus, on some restricted subset of the set of pure λ-terms. These sets of terms, we call them *layered*, are characterized by a kind of typing system which types, called here *levels*, are integers. The idea is that if a higher-order redex contains a β-redex as a strict subterm, the latter is typed by a strictly smaller integer, hence defining layers that can be exploited in the confluence proof, in a way similar to [LJO15]. This technique generalizes *confinement* as described in [ADJL16], by having infinitely many layers instead of only two. It also generalizes the left-linear case, since the whole set of terms can then be considered as forming a single layer at level 0.

Our main result is Theorem 4.4.9 which generalizes Theorem 3.5.5. It states that terminating rewrite systems are confluent together with β restricted to the subset of layered terms.

4.1 Confinement and layering

There are two well-known counter examples to the confluence of non-left-linear systems.

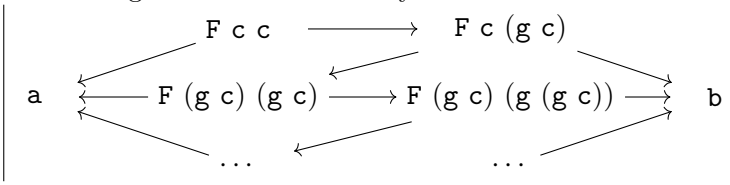
Example 1: Klop's counter example [Kd89] consists of a single non-left-linear rewrite rule $\mathtt{F}\ X\ X \to \mathtt{a}$ considered together with β. It is possible to define, in the untyped λ-calculus, a fixpoint term Y such that $Y\ t \underset{\beta}{\longrightarrow\!\!\!\!\!\!\twoheadrightarrow} t\ (Y\ t)$. We write $C := Y\ (\lambda cx.\mathtt{F}\ x\ (c\ x))$ and

$A := Y\ C$ for convenience. We have $A \underset{\beta}{\longrightarrow\!\!\!\!\rightarrow} C\ A \underset{\beta}{\longrightarrow\!\!\!\!\rightarrow} \mathtt{F}\ A\ (C\ A) \underset{\beta}{\longrightarrow\!\!\!\!\rightarrow} \mathtt{F}\ (C\ A)\ (C\ A) \underset{\mathcal{R}}{\longrightarrow} \mathtt{a}$. Besides we also have $A \underset{\beta}{\longrightarrow\!\!\!\!\rightarrow} C\ A \underset{\beta}{\longrightarrow\!\!\!\!\rightarrow} C\ \mathtt{a} \underset{\beta}{\longrightarrow\!\!\!\!\rightarrow} \mathtt{F}\ \mathtt{a}\ (C\ \mathtt{a})$ which does not reduce to $\mathtt{a}$.

The main idea to forbid this counter-example is to stratify terms into layers in order to limit the occurrences of non-linearly rewritten symbols and forbid a set of terms, deemed unsafe, that is as small as possible. In Klop's counter example, the functional term $\lambda x.\mathtt{F}\ x\ (c\ x)$ should be treated carefully since its argument is nested below the unsafe $\mathtt{F}$ symbol. In the example of Section 4.5 this term is simply forbidden since it is layer-increasing. As discussed in Section 4.6, this criteria could most likely be improved to allow this particular term. Instead the application of the fixpoint Y operator should be controlled so as to forbid the term C creating the counter example.

Example 2: Consider the following first-order rewrite system:

$$\begin{cases} \mathtt{F}\ X\ X & \longrightarrow & \mathtt{a} \\ \mathtt{F}\ X\ (\mathtt{g}\ X) & \longrightarrow & \mathtt{b} \\ \mathtt{c} & \longrightarrow & \mathtt{g}\ \mathtt{c} \end{cases}$$



Rules do not overlap and therefore define a locally confluent system. However, because some rules are non-left-linear, it is in fact necessary to consider their linearized versions, $\mathtt{F}\ X\ Y \to \mathtt{a}$ and $\mathtt{F}\ X\ (\mathtt{g}\ Y) \to \mathtt{b}$ which do overlap. A generalized critical peak must be considered $\mathtt{a} \longleftarrow \mathtt{F}\ X\ X \overset{X\ =\ \mathtt{g}\ X}{=\!=} \mathtt{F}\ X\ (\mathtt{g}\ X) \longrightarrow \mathtt{b}$ which can be obtained by the so-called cyclic unification [LJO15], a variant of unification of infinite rational terms [Hue72]. In this work, we decided to exclude such overlaps by requiring that in any overlap between two linearized left-hand side patterns, the non-linear variables in each pattern occur at parallel positions, see Definition 4.3.4. Taking them into account is probably possible but would require more complex technical developments that are left as future work.

4.1.1 Confinement

The notion of *confinement* was introduced by Assaf, Dowek, Jouannaud and Liu [ADJL16] in order to consider a non-left-linear first-order rewrite system together with higher-order rewrite rules. Non-left-linear rules are meant to operate exclusively on a controlled set of closed terms representing universes. This confinement was furthermore compatible with associativity and commutativity of some symbols, such as `max` and `plus`. This work allowed a faithful representation of universe level conversion in presence of abstract universe variables, either floating or locally bounded, a first step towards supporting universe polymorphism in Dedukti encodings.

[LJO15] introduces an extra set of *confined* first-order symbols $\mathcal{F}_c$ which arity is no longer assumed to be 0 as for symbols of $\mathcal{F}$. The full set of symbols is therefore $\{@, \boldsymbol{\lambda}\} \uplus \mathcal{F} \uplus \mathcal{F}_c$. We also assume a set of confined variable $\mathcal{X}_c$ disjoint from $\mathcal{X}$ and we denote by $\mathcal{T}_c \subseteq \mathcal{T}$ the set of *confined expressions*, which are the terms built over symbols of $\mathcal{F}_c$ and variables of $\mathcal{X}_c$. Terms of $\mathcal{T}_c$ are first-order as they may not contain any occurrence of $@$ or

λ. Variables $x \in \mathcal{X}_c$ in the confined level may not be bound with the usual λ-abstraction, $\lambda x.\, x$ is forbidden, if x is confined. However we allow other binding operators, such as products $\Pi(x : A).B$, to bind confined variables as long as it is guaranteed that such variables are never substituted in reduction steps. This is the case of binding operators that do not occur in rewrite rules, such as products.

We consider a rewrite system $\mathcal{R} = \mathcal{R}_{\text{fo}} \uplus \mathcal{R}_{\text{ho}}$ made of two sets of rules, $\mathcal{R}_{\text{fo}}$ and $\mathcal{R}_{\text{ho}}$. Rules in $\mathcal{R}_{\text{ho}}$ have a, possibly non-left-linear, pattern as their left-hand side. Rules in $\mathcal{R}_{\text{fo}}$ are *confined* and operate exclusively on the set $\mathcal{T}_c$ of *confined* first-order expressions. $\mathcal{R}_{\text{fo}}$ may indeed be an arbitrary normal first-order rewriting systems, with rules, simplifiers and equations [JL12a], as in [ADJL16]. In order to keep away the technicalities associated with normal rewriting systems, we will instead allow $\mathcal{R}_{\text{fo}}$ to be an infinite set of confluent and terminating first-order rewrite rules on the set of confined expressions without meta-variables.

Given a pattern L, we denote by L^{lin} its *linearized incarnation*, obtained by renaming each occurrence of a non-linear meta-variable X of L to X^p if it occurs at position p in L. Note that linearization is the identity for linear patterns. We use the notations $\mathcal{MV}ar^{\text{nl}}(L) \subseteq \mathcal{MV}ar(L)$ for the set of non-linear meta-variables of L and $\mathcal{MV}ar^{\text{l}}(L) := \mathcal{MV}ar(L) \setminus \mathcal{MV}ar^{\text{nl}}(L)$ for the set of its linear meta-variables. Finally, we also need the *non-linear fringe* $F_L^{\text{nl}} \subseteq F_L$ and *linear-fringe* $F_L^{\text{l}} := F_L \setminus F_L^{\text{nl}}$, both sets of positions in L corresponding to the pre-redexes in L formed with the non-linear and linear meta-variables, respectively.

Our purpose in this section is to show the confluence of our rewriting relation $\underset{\mathcal{R}\beta}{\longrightarrow}$, defined as the union of first-order rewriting with $\mathcal{R}_{\text{fo}}$, higher-order rewriting with $\mathcal{R}_{\text{ho}}$, and β-rewrites. Since confluence cannot be obtained for all terms, we first introduce a denumerable set of sets of terms satisfying some assumptions, on which union confluence will hold. We then define the rewriting relation which will be used to show the confluence property by using van Oostrom's technique. Finally, we show that all local peaks of this relation admit decreasing diagrams and conclude.

4.1.2 Term layering

We assume a subset $\mathcal{L} \subseteq \mathcal{T}$ of terms partitioned into pairwise disjoint sets $\mathcal{L}_n$ called *layers*, where $n \in \mathbb{N}$ is the *level* of the terms in $\mathcal{L}_n$. Let also $\mathcal{L}_{\leq n} = \bigcup_{k \leq n} \mathcal{L}_k$.

Definition 4.1.1. *A substitution σ is* well-layered *if $x \in \mathcal{L}_n$ implies that $\sigma(x) \in \mathcal{L}_{\leq n}$.*

Note that the well-layeredness condition only mentions the images of variables. Meta-variables, and meta-terms in general, are not assigned any level.

We assume that the partition into layers satisfies the following properties:

- **(H0)** $\mathcal{L}_0 = \mathcal{T}_c$
- **(H1)** If $t \in \mathcal{L}_n$ and $t \rhd u$, then $u \in \mathcal{L}_{\leq n}$
- **(H2)** If $t \in \mathcal{L}_n$ and $t \underset{\mathcal{R}\beta}{\longrightarrow} u$, then $u \in \mathcal{L}_{\leq n}$

- **(H3)** If $t \in \mathcal{L}_n$ and σ is a well-layered substitution, then $t\sigma \in \mathcal{L}_{\leq n}$
- **(H4)** If $L \to R \in \mathcal{R}$ and $L^{\text{lin}}\sigma \in \mathcal{L}_n$, then $\forall p \in F_L^{\text{nl}}\ (\sigma(X^p) \in \mathcal{L}_{\leq n-1})$

The stratification allows to consider rules such as $\texttt{lift}\ (\texttt{univ}\ X)\ (\texttt{univ}\ X)\ T \longrightarrow T$ provided the non-linear meta-variable X is in $\mathcal{L}_k$ while the whole left-hand side of the rule is at level $\mathcal{L}_n$ for some $n > k$. It also allows abstractions, including non-terminating λ-terms, to inhabit any layer starting with $\mathcal{L}_1$.

The confinement criterion, allows non-left-linear rules to operate at the same level as their non-linear meta-variables, provided this level is 0. At that level, terms are confined, that is first-order, no beta-rewrites are possible. For instance $\texttt{max}(I, I) \longrightarrow I$ is supported in the confined level, `max` being a confined function symbol, hence implying that all considered instances of $\texttt{max}(I, I)$ are confined. Rewriting at the confined level is however forbidden to interact with potentially non-terminating higher-order rules such as β.

4.2 Layered sub-rewriting

4.2.1 Layered rewriting

Rewrite steps may now be labeled with the level of their redex, writing the label below the arrow sign. A step which rewrites several redexes at the same time, such as parallel or orthogonal rewriting may also be labeled with n if all its redexes belong to the same level. If they do not, that steps can still be labeled by some condition on the level, such as $\leq n$.

Definition 4.2.1 (Abstract level rewriting)**.** *Let $u \xrightarrow{P} v$ for some set P of positions in u. We say that u rewrites to v* at level n*, written $u \underset{n}{\xrightarrow{P}} v$, if $\forall p \in P, u|_p \in \mathcal{L}_n$, and with level* at most n*, written $u \underset{\leq n}{\xrightarrow{P}} v$, if $\forall p \in P, u|_p \in \mathcal{L}_{\leq n}$).*

The arrow $\xrightarrow{P}$ may of course be $\overset{P}{\Longrightarrow}$ or $\overset{P}{\otimes\!\!\Longrightarrow}$. Naturally, because of **(H1)** and layer disjointness, rewriting at a level n does not operate on terms of $\mathcal{L}_{<n}$.

Lemma 4.2.2. *If $\mathcal{L}_k \ni u \underset{n}{\xrightarrow{p}} v$ then $k \geq n$. Besides, if $p = \Lambda$ then $k = n$.*

Proof. By definition and disjointness of levels if $p = \Lambda$. Follows from **(H1)** otherwise. □

Rewriting at level n is included in rewriting at level $\leq n$: $\underset{n}{\longrightarrow} \subseteq \underset{\leq n}{\longrightarrow} \subseteq \longrightarrow$. When mentioning a rule or relation name is needed, we will write $L \to R; n$, $\mathcal{R}; n$ or $\beta; n$ instead of only n. Note that $\underset{\mathcal{R}_{\text{fo}}}{\longrightarrow}$ and $\underset{\mathcal{R}_{\text{fo}};0}{\longrightarrow}$ coincide.

Lemma 4.2.3. *Assume $\longrightarrow$ is a monotonic relation. Then $\underset{n}{\longrightarrow}$ and $\underset{\leq n}{\longrightarrow}$ are monotonic.*

Proof. By definition. □

Lemma 4.2.4. *Assume $\longrightarrow$ is a stable relation. Then $\underset{\leq n}{\longrightarrow}$ is stable.*

Proof. Follows from **(H3)**. □

The latter property is not true of $\underset{n}{\longrightarrow}$ however, as evidenced by **(H3)**.

4.2.2 Sub-rewriting

The idea of sub-rewriting is introduced in [LDJ14] and developed further in [LJO15]. Assuming that terms are categorized into levels, and that an instance of a left-hand side of rule belongs to some level, its variables must be instantiated by terms belonging to a strictly smaller level. Here, we need to assume that property for the non-linear variables only, which suffices to rewrite recursively with a strictly lower level the different values of a non-linear variable until they become equal in which case rewriting takes place. This can be seen as a very controlled way of using matching modulo the equational theory defined by a set of confluent rewrite rules. Although levels introduced in [LJO15] are completely different from the levels defined here, and sub-rewriting is slightly different, similar techniques apply. We hide most of these differences away by using an axiomatic approach which allows us to replace the level values with the properties **(H0)** to **(H4)** that they satisfy.

Definition 4.2.5 (Sub-rewriting)**.** *A term u* sub-rewrites *to a term v at position $p \in \mathcal{P}os(u)$ for some rule $L \to R \in \mathcal{R}$, written $u \overset{p}{\underset{\mathcal{R}}{\rhd\!\!\longrightarrow}} v$, if there exists a substitution θ such that $u \xrightarrow[\mathcal{R}\beta]{\geq_{\mathcal{P}} p \cdot F_L^{\mathrm{nl}}\!\!\!\!\!\!\twoheadrightarrow} u[L\theta]_p \xrightarrow[L\to R]{p} u[R\theta]_p = v$.*

The term $u|_p$ is called sub-rewriting redex *of u at p, the substitution θ is an* equalizer *of L, and the rewrite steps from $u|_p$ to $L\theta$ constitute the corresponding* equalization*.*

Note that we could replace sub-rewriting below F_L^{nl} with sub-rewriting below the level of the redex, which would be more permissive. The present definition eases some technicalities, but this alternative might lead to sharper results.

This definition of sub-rewriting allows arbitrary rewriting below the left-hand side of the rule until a redex is obtained. It relates therefore to rewriting modulo, but differs from it by being directional and by matching modulo rewrite steps below the non-linear variables of the left-hand side of the rule exclusively, instead of inside the whole (sub-rewriting) redex.

We can of course categorize sub-rewriting by levels, with the expected notation. Sub-rewriting at level 0 should coincide with rewriting with $\mathcal{R}_{\mathrm{fo}}$, which actually follows with our view of an infinite closed rewriting system.

Sub-rewriting is already a multi-step reduction but it can safely be further extended to parallel sub-rewriting. It is even necessary to consider orthogonal sub-rewriting in order to get an equivalent of Theorem 3.5.9 in a non-left-linear setting. Because steps nested below the non-linear fringe can be considered as part of the equalization steps, it is sufficient to

consider orthogonal sub-rewriting at positions O such that nested positions are below the linear fringe of the positions in O which are above: $\forall p, q \in O, p >_{\mathcal{P}} q \Rightarrow p \geq_{\mathcal{P}} q \cdot F^{\mathrm{l}}_{L}$.

Level sub-rewriting has the following monotonicity and stability properties:

Lemma 4.2.6 (Monotonicity)**.** *Let $s[t]_p$ and $s[u]_p$ be layered terms such that $t \underset{\mathcal{R};n}{\overset{}{\rhd\!\!\longrightarrow}} u$. Then $s[t]_p \overset{p}{\underset{\mathcal{R};n}{\rhd\!\!\longrightarrow}} s[u]_p$.*

Lemma 4.2.7 (Stability)**.** *Assume $u \overset{p}{\underset{\mathcal{R};n}{\rhd\!\!\longrightarrow}} v$ and that σ is a well-layered substitution. Then $u\sigma \overset{p}{\underset{\mathcal{R};\leq n}{\rhd\!\!\longrightarrow}} v\sigma$.*

Proof. By induction on the level. Stability at level $< n$ allows to have $u\sigma \overset{\geq_{\mathcal{P}} pF^{\mathrm{nl}}_{L}}{\underset{\mathcal{R};<n}{\rhd\!\!\longrightarrow\!\!\!\!\!\rightarrow}} u[L\theta]_p\sigma$. We have by stability of (standard) rewriting $(u\sigma)[L\theta\sigma]_p \underset{L\to R}{\overset{p}{\longrightarrow}} u\sigma[R\theta\sigma]_p = v\sigma$. We conclude since, by **(H3)**, $(u\sigma)|_p = u|_p\sigma \in \mathcal{L}_{\leq n}$. □

A important property that follows directly from Definition 4.2.5 is that a sub-rewriting redex is an instance of a linearized left-hand side of rule:

Lemma 4.2.8. *Let $u \overset{p}{\underset{\mathcal{R}_{\mathrm{ho}};n}{\rhd\!\!\longrightarrow}} v$. Then there exist $L \to R \in \mathcal{R}_{\mathrm{ho}}$ and substitutions σ, τ, δ st:*

- $\mathcal{D}om(\sigma) = \mathcal{MV}ar^{\mathrm{l}}(L)$, $\mathcal{D}om(\delta) = \mathcal{MV}ar^{\mathrm{nl}}(L)$, *and* $\mathcal{D}om(\tau) = \mathcal{MV}ar(L^{\mathrm{lin}}) \backslash \mathcal{MV}ar^{\mathrm{l}}(L)$
- $\forall X \in \mathcal{MV}ar^{\mathrm{nl}}(L)\ \forall X^p \in \mathcal{MV}ar(L^{\mathrm{lin}})\ (\tau(X^p) \underset{<n}{\overset{}{\longrightarrow\!\!\!\!\!\rightarrow}} \delta(X))$
- $u|_p = L^{\mathrm{lin}}(\sigma \cup \tau)$ *and* $v|_p = R(\sigma \cup \delta)$.

Note that $\sigma \cup \tau = \sigma\tau = \tau\sigma$ since σ and τ have disjoint domains and ranges. The same applies to σ, δ.

Corollary 4.2.8.1. $\underset{\mathcal{R}\beta;n}{\overset{}{\longrightarrow}} \subseteq \underset{\beta;n}{\overset{}{\otimes\!\!\Longrightarrow}} \cup \underset{\mathcal{R},n}{\overset{}{\rhd\!\!\longrightarrow}} \subseteq \underset{\beta;n}{\overset{}{\longrightarrow\!\!\!\!\!\rightarrow}} \cup \underset{\mathcal{R}\beta;<n}{\overset{}{\longrightarrow\!\!\!\!\!\!\!\rightarrow}} \underset{\mathcal{R};n}{\overset{}{\longrightarrow}} \subseteq \underset{\mathcal{R}\beta,\leq n}{\overset{}{\longrightarrow\!\!\!\!\!\!\!\rightarrow}}$.

This corollary shows that the relation $\underset{\mathcal{R}\beta}{\overset{}{\longrightarrow}}$ is confluent iff the relation $\bigcup_{n\geq 0} \underset{\beta;n}{\overset{}{\otimes\!\!\Longrightarrow}} \cup \underset{\mathcal{R}}{\overset{n}{\rhd\!\!\longrightarrow}}$ is confluent. We will show that the latter is confluent, provided $\mathcal{R}$ satisfies some conditions, by using van Oostrom's technique within an induction over n. This however requires some preparation.

4.3 Overlapping sub-rewriting peaks

Overlapping non-linear patterns is already more complex than in the linear case but unification of non-linear patterns can still be done in linear time [DHKP98]. The algorithm is similar to the linear case with a simple check that X does not occur in t before substituting a solved equation $X[\overline{x}] = t$ in the remaining problems. Equations $X[\overline{x}] = t$ with X a strict subterm of t simply have no solution and so-called flexible-flexible equation $X[\overline{x}] = X[\overline{y}]$

are resolved, similarly to the linear case, by introducing a fresh meta-variable Z, local variables $\overline{z}$ such that $x_i = y_i \Rightarrow x_i \in \overline{z}$ and the equation $X[\overline{x}] = Z[\overline{z}]$.

Overlaps between sub-rewriting steps are way more complex as nested equations such as $X = \mathtt{S}\ X$ may now have a solution modulo sub-rewriting if X is a non-linear meta-variable. In our case, sub-rewriting is done at a smaller level, but this constraint does not offer much guarantee since each level, starting from $\mathcal{L}_1$, contains the usual β-reduction which is non-terminating and allows the definition of fixpoint combinators.

Adapting the unification algorithm can lead to non-terminating calculations if one is not careful. Consider for example the following unification problem $\mathtt{F}\ X\ (\mathtt{S}\ X)\ u = \mathtt{F}\ Y\ Y\ v$. The unification algorithm yields $\{X = Y \wedge Y = \mathtt{S}\ X \wedge u = v\}$ and the first two equations may be substituted in the remaining unification problem $u = v$. In the linear case, they could simply be substituted once in all remaining problems, however, here, substituting X with Y in u and v introduces the variables Y which may in turn be substituted with $\mathtt{S}\ X$, therefore reintroducing the variable X and possibly requiring further substitutions of equations in solved form. This non-termination is usually avoided in unification algorithms by performing an *occurs-check* and failing in case of cyclic unification problems. With sub-rewriting, this becomes unsound.

We need a condition allowing us to reduce the unification of two higher-order patterns modulo $\mathcal{R} \cup \beta$ sub-steps to the higher-order unification of the linearized versions of these patterns.

4.3.1 Linear independence

Our first condition aims at forbidding the potential cyclicity of non-linear unification problems. We rely on a property of the linearized unifications very much alike Toyama's variable condition [AYT09].

Definition 4.3.1 (Linear independence)**.** *Two closed pre-patterns L and G are* linearly independent *iff $\forall p \in \mathcal{P}os(L)$ such that L^{lin} and G^{lin} are renamed apart and unify at p, we have:*

- $\forall q \in \mathcal{FP}os(G)$ *such that* $p \cdot q \in F_L^{\text{nl}}$, $\mathcal{MV}ar^{\text{nl}}(G) \cap \mathcal{MV}ar(G|_q) = \varnothing$;
- $\forall q \in \mathcal{FP}os(L)$ *such that* $q \in p \cdot F_G^{\text{nl}}$, $\mathcal{MV}ar^{\text{nl}}(L) \cap \mathcal{MV}ar(L|_q) = \varnothing$;

In particular, if either L or G is linear or if L^{lin} and G^{lin} are not unifiable at any position, then L is linearly independent from G. Any pattern L unifies with a renaming of itself at the root, but this never prevents a closed pre-pattern to be linearly independent from itself (however unification at other positions might).

This criteria is a bit stronger than Toyama's variable condition and could probably be relaxed as long as it keeps forbiding cyclicity and therefore removes the need for occur-checks in the usual unification algorithm of linear pre-patterns.

Lemma 4.3.2. *Computing a most general unifier of two linearly independent closed pre-patterns is done in polynomial time.*

Proof. A unifier θ of $L|_p$ and $G^{\overline{x}}$ (lifting of G where variables bound above p in L are added to the arguments of the meta-variables of G) is an instance of the most general unifier σ_0 of $L^{\mathrm{lin}}|_p$ and $(G^{\mathrm{lin}})^{\overline{x}}$ (which is computed in linear time): $\theta = \sigma_0\tau_0$, for some τ_0. By linear independence, for all linearized version $X^q \in \mathcal{D}om(\sigma_0)$ of some non-linear meta-variable $X \in \mathcal{MV}ar^{\mathrm{nl}}(L)$, there is an occurrence $X^q[\overline{y}] \lhd L^{\mathrm{lin}}|_p$ and $q \in \mathcal{FP}os(G)$ such that $X^q[\overline{y}]\sigma_0 = G|_q\sigma_0$ where $G|_q$ (and therefore $G|_q\sigma_0$) is linear. If we consider two such occurrences at positions q and q', then we have $X^q[\overline{y}]\sigma_0\tau_0 = X^{q'}[\overline{y}]\sigma_0\tau_0 = X[\overline{y}]\theta$, therefore τ_0 is a unifier of $G|_q\sigma_0$ and $G|_{q'}\sigma_0$. This unification problem is now linear, hence has a linear m.g.u. σ_1 computable in linear time such that $\tau_0 = \sigma_1\tau_1$ and therefore $\theta = (\sigma_0\sigma_1)\tau_1$. We proceed to successively unify this way all meta-variables in $\mathcal{D}om(\sigma)$ corresponding to the same non-linear meta-variable in $L|_p$ or in G. An easy induction on the number of occurrences of X in $L|_p$ or in G shows therefore that there is a most general linear substitution $\sigma := \sigma_0 \cdots \sigma_n$ and τ_n such that $\theta = \sigma\tau_n$ and all $\sigma(X^q)$ are equal. We can therefore uniquely define an unifier σ' which domain ranges over the meta-variables of $L|_p$ and G by $\sigma'(X) := \sigma(X^q)$ if $X^q \in \mathcal{D}om(\sigma)$, otherwise $\sigma'(Y) := \sigma(Y)\{X^q \mapsto \sigma(X^q)\}$. We have $L|_p\sigma' = (L|_p)^{\mathrm{lin}}\{X^q \mapsto X\}\sigma' = (L|_p)^{\mathrm{lin}}\sigma = G^{\mathrm{lin}}\sigma = G^{\mathrm{lin}}\{X^q \mapsto X\}\sigma' = G\sigma'$. □

Example 1: Consider the problem $\mathtt{F}\ T\ (\mathtt{S}\ T)\ (\mathtt{S}\ (\mathtt{S}\ T)) = \mathtt{F}\ (\mathtt{S}\ X)\ (\mathtt{S}\ Y)\ (\mathtt{S}\ Z)$. The usual unification algorithm applied to the linearized patterns yields the most general unifier $\sigma_0 := \{T_1 \mapsto \mathtt{S}\ X, T_2 \mapsto Y, Z \mapsto \mathtt{S}\ T_2\}$. We unify $(\mathtt{S}\ X)\sigma_0 = \mathtt{S}\ X$ and $Y\sigma_0 = Y$ and get $\sigma_1 = \{Y \mapsto \mathtt{S}\ X\}$ which allows to define $\sigma := \sigma_0\sigma_1 = \{T_1 \mapsto \mathtt{S}\ X, T_2 \mapsto \mathtt{S}\ X, Z \mapsto \mathtt{S}\ T_2, Y \mapsto \mathtt{S}\ X\}$. This substitution is then turned into a unifier for the non-linearized problem by collapsing meta-variables: $\sigma' = \{T \mapsto \mathtt{S}\ X, Z \mapsto \mathtt{S}\ (\mathtt{S}\ X), Y \mapsto \mathtt{S}\ X\}$.

Definition 4.3.3. *A set of rules is* linearly independent *if any two of its rules have linearly independent left-hand sides.*

Note that, since patterns are β-normal and the β-rule is left-linear, $\mathcal{R} \cup \beta$ is linearly independent if so is $\mathcal{R}$.

4.3.2 Linear compatibility

We also need the equalization steps from both side of a local peak to preserve the functional positions of the other redex so as to be replayed.

Definition 4.3.4 (Linear compatibility)**.** *Two closed pre-patterns L and G are* linearly compatible *in a set $\mathcal{R}$ of rewrite rules iff $\forall p \in \mathcal{P}os(L)$ such that L^{lin} and G^{lin} are renamed apart and unify at p, we have:*

- *$\forall q \in \mathcal{FP}os(G)$ such that $p \cdot q \geq_{\mathcal{P}} F^{\mathrm{nl}}_L$, $G|_q$ is unifiable with no linearized left-hand side of a rule in $\mathcal{R}$;*
- *$\forall p \cdot q \in \mathcal{FP}os(L)$ such that $q \geq_{\mathcal{P}} F^{\mathrm{nl}}_G$, $L|_{p\cdot q}$ is unifiable with no linearized left-hand side of a rule in $\mathcal{R}$.*

In particular, if L and G are linear or if L^{lin} and G^{lin} are not unifiable at any position, then L is linearly compatible with G in any set $\mathcal{R}$. Any pattern L unifies with a renaming of itself at the root, but this never prevents a closed pre-pattern to be linearly compatible with itself (however unification at other positions might).

Lemma 4.3.5. *If $F_L^{\text{nl}} \cap p \cdot \mathcal{FP}os(G) = p \cdot F_G^{\text{nl}} \cap \mathcal{FP}os(L) = \varnothing$ for all position p at which L^{lin} and G^{lin} unify, then L and G are linearly compatible in all set $\mathcal{R}$ of rewrite rules.*

Proof. The criteria ensures that no unification check with rules of $\mathcal{R}$ is needed. □

Example 2: We assume $\mathtt{u}$ is a symbol rewritten by some rule in $\mathcal{R}$. The pre-pattern $L := \mathtt{f}\ X\ (\mathtt{u}\ X)$ is linearly compatible with $\mathtt{f}\ X\ Y$ and $\mathtt{f}\ X\ Y\ (\mathtt{u}\ X)$ but neither from $\mathtt{f}\ (\mathtt{u}\ X)\ Y$ nor from $\mathtt{u}\ (\mathtt{u}\ X)$ nor even from $G := \mathtt{f}\ X\ X$ even though G is not unifiable with L. Indeed, the linearized versions of L and G unify at position Λ and we have both $2 \geq_{\mathcal{P}} F_G^{\text{nl}} = \{12, 2\}$ and $\Lambda \cdot 2 \in \mathcal{FP}os(L)$ since $L|_2 = \mathtt{u}\ X$ which is unifies with some rule of $\mathcal{R}$ by assumption.

Definition 4.3.6. *A set $\mathcal{R}$ of rules is* linearly compatible *if the left-hand sides of any two rules in $\mathcal{R}$ are linearly compatible in $\mathcal{R}$.*

Note that, since patterns are β-normal and the β-rule is left-linear, $\mathcal{R} \cup \beta$ is linearly compatible if so is $\mathcal{R}$.

Example 3: Let $\mathcal{R} := \{\mathtt{f}\ X\ (\mathtt{f}\ X\ Y) \to R\ ,\ \mathtt{f}\ Z\ Z \to D\}$. Checking linear compatibility requires considering all three possibilities of unifying one (linearized) left-hand side with a subterm of the (linearized) other:

- $\mathtt{f}\ X\ (\mathtt{f}\ X\ Y)$ with itself at position 2, which gives the linearized solvable equation $\mathtt{f}\ X^1\ (\mathtt{f}\ X^{2\cdot1}\ Y) = \mathtt{f}\ X'^{2\cdot1}\ Y'$. The condition is satisfied without the need for a check;
- $\mathtt{f}\ Z\ Z$ with $\mathtt{f}\ X\ (\mathtt{f}\ X\ Y)$ at position 2, which gives the linearized solvable equation $\mathtt{f}\ Z^1\ Z^2 = \mathtt{f}\ X^{2\cdot1}\ Y$ for which the condition is, again, satisfied.
- $\mathtt{f}\ Z\ Z$ with $\mathtt{f}\ X\ (\mathtt{f}\ X\ Y)$ at position Λ, which gives the linearized solvable equation $\mathtt{f}\ Z^1\ Z^2 = \mathtt{f}\ X^1\ (\mathtt{f}\ X^{2\cdot1}\ Y)$. In that case the condition is not satisfied since Z^2 must be unified with a functional subterm $\mathtt{f}\ X^{2\cdot1}\ Y$.

$\mathcal{R}$ is thus not linearly compatible, but replacing $\mathtt{f}\ X\ (\mathtt{f}\ X\ Y)$ by $\mathtt{f}\ X\ (\mathtt{g}\ X\ Y)$ yields a linearly compatible set of rewrite rules.

4.3.3 Critical peaks

In order to ease the presentation of our critical peak lemma, we will require rules to be linearly independent as an extra assumption.

- **(H5)** $\mathcal{R}$ is linearly independent and linearly compatible.

Lemma 4.3.7. ***(H5)*** *is checked in polynomial time.*

Proof. The quadratic number of overlaps of linearized left-hand sides can be generated in quadratic time. Both properties are then check linearly in the total size of these overlaps. □

Critical pairs between rules in $\mathcal{R}_{\text{fo}}$ do not matter since $\mathcal{R}_{\text{fo}}$ is assumed confluent. However, we still have the case where L is a left-hand side of rule in $\mathcal{R}_{\text{ho}}$ and G a left-hand side of rule in $\mathcal{R}_{\text{fo}}$. **(H5)** must therefore apply in that case.

If a pair of rules in $\mathcal{R}_{\text{ho}}$ satisfies **(H5)** (or the above one in $\mathcal{R}_{\text{ho}}$ and the other in $\mathcal{R}_{\text{fo}}$), then a non-linear variable of one left-hand side cannot sit above a non-linear variable of the left-hand side of the other in case they overlap. This is not very restrictive in practice, but has two important consequences: firstly, unification of two such patterns can be performed using the linear unification algorithm given in [FJ19a], the result being a substitution of domain included in $\mathcal{MV}ar^{\text{l}}(L) \cup \mathcal{MV}ar^{\text{l}}(G)$; secondly, equalization steps from an overlapping peak $u \underset{L\to R}{\overset{\Lambda}{\leftarrow\!\!\triangleleft}} s \underset{G\to D}{\overset{p}{\triangleright\!\!\longrightarrow}} v$ do not take place above F_L or $p \cdot F_G$. This simplifies the proof of the coming critical peak lemma drastically, although we suspect that this second property is not really needed.

The critical pair lemma comes in two parts: a critical peak lemma that characterizes the situation under which an overlapping peak may exist; and a lifting lemma that shows a joinability property of all overlapping peaks provided it is true of all critical peaks.

Lemma 4.3.8 (Critical peak lemma)**.** *Assume* $s \underset{L\to R;n}{\overset{q}{\leftarrow\!\!\triangleleft}} u \underset{G\to D;n}{\overset{q\cdot p}{\triangleright\!\!\longrightarrow}} t$ *is an overlapping peak. Then,* $L|_p$ *and* $G^{\overline{x}}$ *have a most general unifier* ρ *and there is a substitution* δ *and a critical peak* $s' \underset{L\to R;n}{\overset{\Lambda}{\leftarrow\!\!\triangleleft}} L\rho = L\rho[D\rho]_p \underset{G\to D;n}{\overset{p}{\triangleright\!\!\longrightarrow}} t'$ *such that* $s|_q \underset{<n}{\longrightarrow\!\!\!\!\!\twoheadrightarrow} s'\delta$ *and* $t|_q \underset{<n}{\longrightarrow\!\!\!\!\!\twoheadrightarrow} t'\delta$.

Proof. By monotonicity of sub-rewriting, we can assume without loss of generality that $q = \Lambda$. The proof is illustrated in detail in Figure 9.3, Figure 4.1 oversimplified but conveys the idea.

By Lemma 4.2.8, $u = L^{\text{lin}}\sigma\tau$ with $\mathcal{D}om(\sigma) \subseteq \mathcal{MV}ar^{\text{nl}}(L)$, $\mathcal{D}om(\tau) \subseteq \mathcal{MV}ar^{\text{l}}(L)$, $\sigma \underset{<n}{\longrightarrow\!\!\!\!\!\twoheadrightarrow} \sigma'$ (using **(H4)**,**(H1)** and **(H2)**) and $u \longrightarrow\!\!\!\!\!\twoheadrightarrow L\sigma'\tau \underset{L\to R}{\overset{\Lambda}{\longrightarrow}} R\sigma'\tau = s$. We write $v := L\sigma'\tau$.

By the same token, $u|_p = G^{\text{lin}}\gamma\theta$ with $\mathcal{D}om(\gamma) \subseteq \mathcal{MV}ar^{\text{nl}}(G)$, $\mathcal{D}om(\theta) \subseteq \mathcal{MV}ar^{\text{l}}(G)$, $\gamma \underset{<k}{\longrightarrow\!\!\!\!\!\twoheadrightarrow} \gamma'$ and $u|_p \longrightarrow\!\!\!\!\!\twoheadrightarrow G\gamma'\theta \underset{G\to D}{\overset{\Lambda}{\longrightarrow}} D\gamma'\theta = t|_p$. We write $w := L^{\text{lin}}\sigma\tau[G\gamma'\theta]_p$ and by monotonicity, $u \longrightarrow\!\!\!\!\!\twoheadrightarrow L^{\text{lin}}\sigma\tau[G\gamma'\theta]_p = w \underset{G\to D}{\overset{\Lambda}{\longrightarrow}} L^{\text{lin}}\sigma\tau[D\gamma'\theta]_p = t$.

We define $F_H := \{o \in F_L \mid o \geq_{\mathcal{P}} p \cdot F_G^{\text{nl}}\} \cup \{o \in p \cdot F_G \mid o \geq_{\mathcal{P}} F_L^{\text{nl}}\}$ the positions from the fringe of either left-hand side which are also below the non-linear fringe of the other. We define $H := u_{F_H}$, which is, following our notations, the term u in which the subterms at positions F_H are replaced with fresh meta-variables applied to all locally bounded variables. Since u is ground, F_H is the fringe of H.

All steps in the derivation from u to v must be at positions below F_H, since otherwise, the first one falsifying this property would falsify linear independence of $\mathcal{R} \cup \beta$, which is assumed in **(H5)**. This is true as well of the derivation from u to w. Therefore, v and w

coincide with H at all positions above F_H, and since H is linear, $v = H\delta$ and $w = H\varphi$ for some substitutions δ and φ respectively.

We construct now, using our confluence assumption, derivations from v and from w to a common reduct $v \overset{\geq_{\mathcal{P}} F_H}{\underset{<n}{\twoheadrightarrow}} u' \overset{\geq_{\mathcal{P}} F_H}{\underset{<n}{\twoheadleftarrow}} w$. Since v is an instance of L and $F_H \geq_{\mathcal{P}} F_L$, so is u'. Since w is an instance of $L[G^{\overline{x}}]_p$ and $F_H \geq_{\mathcal{P}} p \cdot F_G$, so is u', hence unifying $L|_p$ and $G^{\overline{x}}$. Note that both derivations operate at levels strictly below n. Finally, since both v and w are instance of H, so is u'. Therefore, L and G are unifiable at position p and we have $u' = H\delta$ for some unifying substitution δ.

We are left with routine calculations for commuting the two derivations originating from v (resp. w), and conclude the proof. □

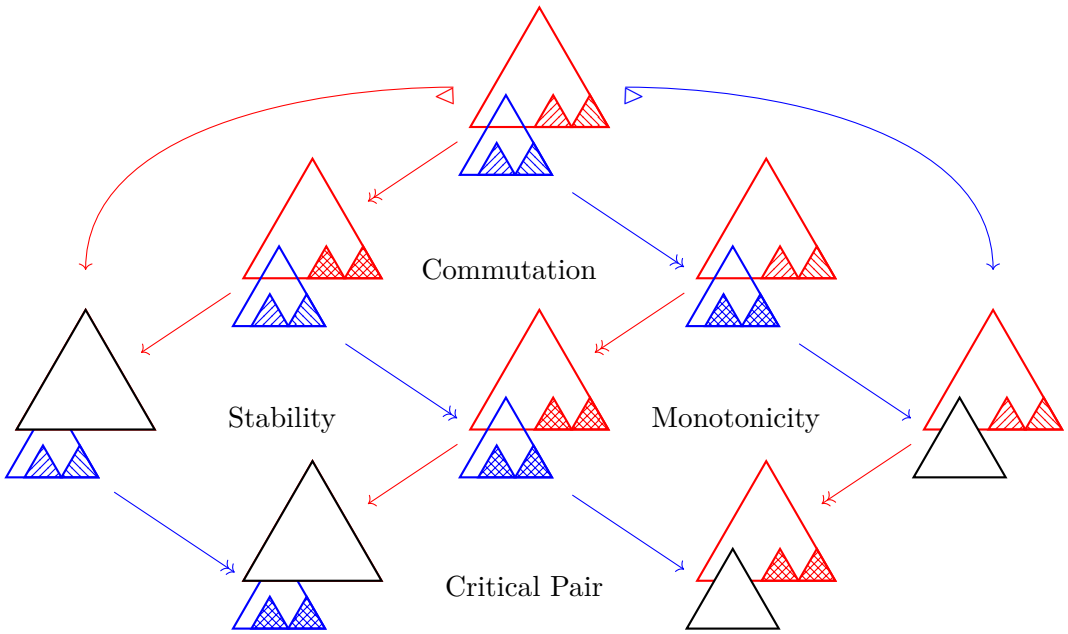


Figure 4.1: Critical pair lemma: Lemma 4.3.8

The critical pair lemma shows that overlapping peaks are joinable when critical pairs are joinable.

4.4 Decreasing Diagrams

We assume from now on a set $\mathcal{R}$ of rewrite rules satisfying **(H5)**. We will prove that Theorem 3.5.5 extends to sub-rewriting with non-linear rules. The strong normalization assumption must be replaced by a last assumption:

- **(H6)** There is a measure function $[\![u]\!]_n$ mapping a term u and a level n to a set $\mathcal{E}$ equipped with a well-founded partial order $\preceq$ such that

– If $u \underset{\mathcal{R}\beta;<n}{\longrightarrow} v$, then $[\![v]\!]_n \preceq [\![u]\!]_n$
– If $u \underset{\mathcal{R};n}{\longrightarrow} v$, then $[\![v]\!]_n \prec [\![u]\!]_n$

Note that if $\mathcal{R}$ is terminating, we can choose an empty set of confined terms, $\mathcal{L}_0 = \varnothing$, a unique layer containing all terms, $\mathcal{L}_1 = \mathcal{T}$ and define $[\![u]\!]_n := u \in \mathcal{E} := \mathcal{T}$ equipped with the well-founded $\underset{\mathcal{R}}{\longrightarrow}$. When considering more than one non-empty layer, the difficulty is to define $[\![\cdot]\!]_n$ decreasing by rewriting at level $k < n$. One way to achieve this is to consider $[\![u]\!]_n$ to be the term u where all subterms at level strictly lower than n are replaced by a special constant $\Box$. This erasure mechanism requires however to prove the termination of rewriting at level n without relying on the shape of subterms at level $< n$, see Section 4.5 for an example of use of this idea.

We consider the relations $\underset{\beta;n}{\otimes\!\!\Longrightarrow}$ labeled with $(n, \bot)$ and $u \underset{\mathcal{R};n}{\triangleright\!\!\longrightarrow} v$ labeled with $(n, [\![u]\!]_n)$. The order on labels is the lexicographic ordering $(\leq, \preceq)$ where $\preceq$ itself is extended to have $[\![u]\!]_n \preceq \bot$.

The relation $\underset{\mathcal{R}_{\text{fo}}}{\longrightarrow}$ only operates on $\mathcal{L}_0$ and is the only relation operating on this set of terms. We assume it is confluent, a property that can be proved in practice using standard first-order techniques since $\mathcal{L}_0$ contains first-order terms only.

4.4.1 Joinability of non-overlapping local peaks

We now proceed proving our confluence theorem, Theorem 4.4.9, stated at the end of the section: assuming critical pairs of $\mathcal{R}_{\text{ho}}$ and critical pairs of $\mathcal{R}_{\text{fo}}$ onto $\mathcal{R}_{\text{ho}}$ are joinable, $\underset{\mathcal{R}\beta}{\longrightarrow}$ is confluent. Its proof is by a course-of-values induction and therefore assume in the rest of this section a level $n > 0$ such that $\underset{\beta\mathcal{R};<n}{\longrightarrow}$ is already proven confluent and we prove that $\underset{\beta\mathcal{R};n}{\longrightarrow}$ is confluent.

There are seven cases of local peaks for which we exhibit decreasing diagrams in the coming lemmas. Eventually we conclude that $\underset{\mathcal{R}\beta}{\longrightarrow}$ is confluent on well-layered terms. In the proof the confluence assumption of $\underset{\beta\mathcal{R};<n}{\longrightarrow}$ made in those lemmas is replaced by the induction hypothesis.

We start with local peaks of $\triangleright\!\!\longrightarrow_{\mathcal{R}}$, then move to local peaks of $\underset{\beta}{\otimes\!\!\Longrightarrow}$, and end up with mixed local peaks involving both $\triangleright\!\!\longrightarrow_{\mathcal{R}}$ and $\underset{\beta}{\otimes\!\!\Longrightarrow}$.

Disjoint higher-order rewrite steps commute, as usual, because of monotonicity:

Lemma 4.4.1. *If* $s \overset{p}{\underset{\mathcal{R};m}{\longleftarrow\!\!\triangleleft}} u \overset{q}{\underset{\mathcal{R};n}{\triangleright\!\!\longrightarrow}} t$ *with* $q \# p$, *then* $s \overset{q}{\underset{\mathcal{R};n}{\triangleright\!\!\longrightarrow}} v \overset{p}{\underset{\mathcal{R};m}{\longleftarrow\!\!\triangleleft}} t$ *for some* v.

We check that this is a decreasing diagram, by **(H6)**. If $n = m$, then $[\![s]\!]_n \prec [\![u]\!]_n$ and $[\![t]\!]_m \prec [\![u]\!]_m$, otherwise, w.l.o.g., $n < m$ and $(n, [\![s]\!]_n) \prec (m, [\![t]\!]_m) \preceq (m, [\![u]\!]_m)$.

We now consider $\mathcal{R}$ ancestor peaks. The decreasing diagrams for both cases are illustrated in Figure 4.2.

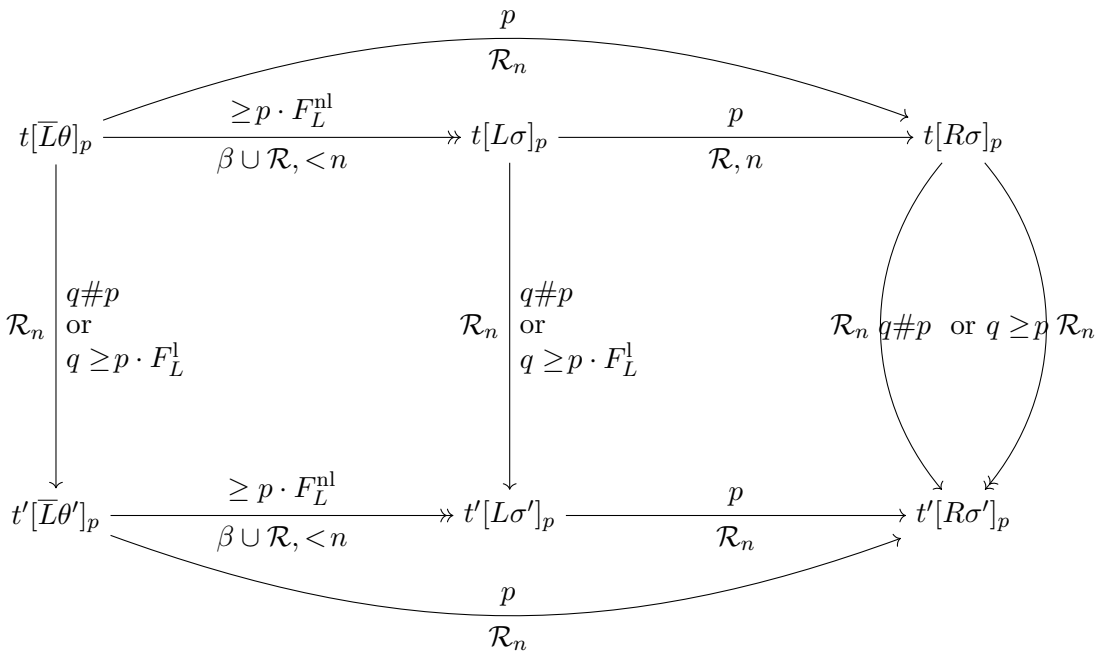


Figure 4.2: DD for non-critical peaks: $\underset{\mathcal{R},n}{\triangleleft\!\longleftarrow} \underset{\mathcal{R},n}{\longrightarrow\!\triangleright}$. See Lemma 4.4.2 and Lemma 4.4.1.

Lemma 4.4.2. *If* $s \underset{L\to R;n}{\overset{p}{\triangleleft\!\longleftarrow}} u \underset{\mathcal{R};k}{\overset{q}{\longrightarrow\!\triangleright}} t$ *with* $q \geq_{\mathcal{P}} p{\cdot}F^{\mathrm{l}}_L$*, then* $n \geq k$ *and* $s \underset{\mathcal{R};\leq k}{\overset{\geq_{\mathcal{P}} p}{\triangleright\!\twoheadrightarrow}} v \underset{L\to R;\leq n}{\overset{p}{\triangleleft\!\longleftarrow}} t$ *for some* v*.*

Proof. Assuming $p = \Lambda$, by Lemma 4.2.8, $u = L^{\mathrm{lin}}(\sigma \cup \tau)$ and $s = R(\sigma \cup \delta)$, for some σ, τ, δ such that $\mathcal{D}om(\sigma) \subseteq \mathcal{MV}ar^{\mathrm{l}}(L)$, $\mathcal{D}om(\tau) \subseteq \mathcal{MV}ar^{\mathrm{nl}}(L)$, $\mathcal{D}om(\delta) \subseteq \mathcal{MV}ar^{\mathrm{nl}}(L)$, and we have $u = L^{\mathrm{lin}}(\sigma \cup \tau) \underset{<n}{\overset{\geq_{\mathcal{P}} p \cdot F^{\mathrm{nl}}_L}{\twoheadrightarrow}} L(\sigma \cup \delta) \underset{\mathcal{R};n}{\longrightarrow} R(\sigma \cup \delta) = s$. Since $q \geq_{\mathcal{P}} F^{\mathrm{l}}_L$, then $q = o{\cdot}q'$, where $o \in F^{\mathrm{l}}_L$ and $L(o) = X$. By **(H1)**, $n \geq k$. Then, $\sigma(X) \underset{\mathcal{R},k}{\overset{q'}{\longrightarrow\!\triangleright}} \theta(X)$, for some meta-substitution θ such that $t = L^{\mathrm{lin}}(\theta \cup \tau)$. Since $q\#F^{\mathrm{nl}}_L$, the rewrite at q commutes with the equalization steps, and therefore: $s = R(\sigma \cup \delta) \underset{k}{\triangleright\!\twoheadrightarrow} R(\theta \cup \delta) \underset{n}{\twoheadleftarrow} L(\theta \cup \delta) \underset{<n}{\overset{\geq_{\mathcal{P}} F^{\mathrm{l}}_L}{\twoheadleftarrow}} L^{\mathrm{lin}}(\theta \cup \tau) = t$. □

Lemma 4.4.3. *Let* $u \underset{L\to R;n}{\overset{p}{\triangleleft\!\longleftarrow}} s \underset{\mathcal{R};k}{\overset{q}{\longrightarrow\!\triangleright}} v$ *with* $q \geq_{\mathcal{P}} p{\cdot}F^{\mathrm{nl}}_L$*. Then* $n > k$ *and* $u \underset{\mathcal{R};<n}{\longrightarrow\!\!\!\!\!\twoheadrightarrow} t \underset{L\to R;\leq n}{\overset{p}{\triangleleft\!\longleftarrow}} v$ *for some* t*.*

Proof. We start as above by using Lemma 4.2.8. This time, $q \geq_{\mathcal{P}} p \cdot F^{\mathrm{nl}}_L$, hence $q \geq_{\mathcal{P}} p \cdot o$, where $o \in F^{\mathrm{nl}}_L$ and $L|_o = X^o$. This case will require using the induction hypothesis.

Let $\gamma(X) = v|_{p \cdot o}$, hence $\gamma(X) \underset{k}{\longleftarrow} \tau(X^o) \underset{<n}{\twoheadrightarrow} \delta(X)$. By **(H4)**, $\tau(X^o) \in \mathcal{L}_{<n}$, hence, by **(H1)**, k<n. By induction hypothesis, $\delta(X) \underset{<n}{\twoheadrightarrow} \theta(X) \underset{<n}{\twoheadleftarrow} \gamma(X)$, hence defining $\theta(X)$. Call

now θ the substitution equal to δ except for the variable X, for which it was just defined. Then, $u = s[L(\sigma \cup \delta)]_p \underset{<n}{\longrightarrow\!\!\!\!\!\rightarrow} s[L(\sigma \cup \theta)]_p \underset{<n}{\longleftarrow\!\!\!\!\!\leftarrow} s[L(\sigma \cup \gamma)]_p = v$. The result follows. □

We now move to the local peaks of orthogonal β-reductions:

Lemma 4.4.4. $\overset{O}{\underset{\beta;\leq n}{\otimes\!\!\Longrightarrow}} \subseteq \overset{P}{\underset{\beta;n}{\otimes\!\!\Longrightarrow}} \underset{\beta;<n}{\longrightarrow\!\!\!\!\!\rightarrow}$ *with* $P \subseteq O$.

Proof. Let $t \overset{O}{\underset{\beta;\leq n}{\otimes\!\!\Longrightarrow}} u$. The proof is by induction on the size of O. There are two cases:

1. $\Lambda \notin O$. By induction hypothesis, commutation of β-steps occuring at parallel positions, and finally grouping orthogonal steps occuring at parallel positions.
2. $\Lambda \in O$, hence $t = (\lambda x : t_0.\, t_1)\ t_2$. If $t \in \mathcal{L}_k$ for some $k < n$, the result holds by **(H1)** and **(H2)**. Otherwise $t_1 \overset{O_1}{\underset{\beta;\leq n}{\otimes\!\!\Longrightarrow}} u_1$ and $t_2 \overset{O_2}{\underset{\beta;\leq n}{\otimes\!\!\Longrightarrow}} u_2$ by **(H1)** and $u_1\{x \mapsto u_2\} = u$. By induction hypothesis, $t_1 \overset{P_1}{\underset{\beta;n}{\otimes\!\!\Longrightarrow}} v_1 \underset{\beta;<n}{\longrightarrow\!\!\!\!\!\rightarrow} u_1$ and $t_2 \overset{P_2}{\underset{\beta;n}{\otimes\!\!\Longrightarrow}} v_2 \underset{\beta;<n}{\longrightarrow\!\!\!\!\!\rightarrow} u_2$ with $P_1 \subseteq O_1$ and $P_2 \subseteq O_2$. Finally, using **(H3)** with $\sigma = \{x \mapsto v_2\}$, there is some k such that:

$$t = (\lambda x : t_0.\, t_1)\ t_2 \overset{12\cdot P_1 \cup 2\cdot P_2}{\underset{\beta;n}{\otimes\!\!\Longrightarrow}} (\lambda x : t_0.\, v_1)\ v_2 \underset{\beta;k}{\overset{\Lambda}{\longrightarrow}} v_1\{x \mapsto v_2\} \underset{\beta;<n}{\longrightarrow\!\!\!\!\!\rightarrow} u_1\{x \mapsto u_2\} = u$$

If $k < n$ then the middle step is part of the $\underset{\beta;<n}{\longrightarrow\!\!\!\!\!\rightarrow}$ sequence, otherwise $k = n$ and it can be merged with the orthogonal step. □

Lemma 4.4.5. $\underset{\beta;n}{\Longleftarrow\!\!\otimes}\ \underset{\beta;n}{\otimes\!\!\Longrightarrow} \subseteq \underset{\beta;n}{\otimes\!\!\Longrightarrow}\ \underset{<n}{\overset{\beta}{\longrightarrow\!\!\!\!\!\rightarrow}}\ \underset{<n}{\overset{\beta}{\longleftarrow\!\!\!\!\!\leftarrow}}\ \underset{\beta;n}{\Longleftarrow\!\!\otimes}$

Proof. Orthogonal β-reductions are known to be strongly confluent: $\underset{\beta}{\Longleftarrow\!\!\otimes}\ \underset{\beta}{\otimes\!\!\Longrightarrow} \subseteq \underset{\beta}{\otimes\!\!\Longrightarrow}\ \underset{\beta}{\Longleftarrow\!\!\otimes}$. Hence $\underset{\beta;n}{\Longleftarrow\!\!\otimes}\ \underset{\beta;n}{\otimes\!\!\Longrightarrow} \subseteq \underset{\beta;\leq n}{\otimes\!\!\Longrightarrow}\ \underset{\beta;\leq n}{\Longleftarrow\!\!\otimes}$ by **(H2)**. The result follows then from Lemma 4.4.4. □

We are left with mixed local peaks.

Lemma 4.4.6. *If* $s \overset{O}{\underset{\beta;<n}{\Longleftarrow\!\!\otimes}} u \underset{\mathcal{R};n}{\overset{p}{\triangleright\!\!\longrightarrow}} t$, *then* $s \underset{\mathcal{R};n}{\overset{p}{\triangleright\!\!\longrightarrow}}\ \underset{\mathcal{R}\beta;<n}{\longleftarrow\!\!\!\!\!\triangleleft} t$.

Proof. This proof is illustrated in Figure 4.3. By **(H1)**, $O \not\geq_{\mathcal{P}} p$ and therefore we can write $O = O_1 \uplus O_2 \uplus O_3$ such that $O_1 \# p$, $O_2 \geq_{\mathcal{P}} p \cdot F^{\mathrm{l}}_L$ and $O_3 \geq_{\mathcal{P}} p \cdot F^{\mathrm{nl}}_L$. The peak can then be decomposed into $\overset{O_3}{\Longleftarrow\!\!\otimes}\ \overset{O_2}{\Longleftarrow\!\!\otimes}\ \overset{O_1}{\Longleftarrow\!\!\otimes}\ \overset{p}{\triangleright\!\!\longrightarrow}$ and the inner peak commutes $\overset{O_3}{\Longleftarrow\!\!\otimes}\ \overset{O_2}{\Longleftarrow\!\!\otimes}\ \overset{p}{\triangleright\!\!\longrightarrow}\ \overset{O_1}{\Longleftarrow\!\!\otimes}$. We study the remaining peak assuming w.l.o.g. that $p = \Lambda$. By Lemma 4.2.8, we decompose the sub-rewriting step and commute steps occuring at parallel positions:

$$s = L^{\mathrm{lin}}\tau'\sigma' \overset{O_3}{\Longleftarrow\!\!\otimes} L^{\mathrm{lin}}\tau\sigma' \overset{O_2}{\Longleftarrow\!\!\otimes} u = L^{\mathrm{lin}}\tau\sigma \underset{<n}{\overset{\geq_{\mathcal{P}} F^{\mathrm{nl}}_L}{\longrightarrow\!\!\!\!\!\rightarrow}} L\delta\sigma \underset{\mathcal{R};n}{\overset{\Lambda}{\longrightarrow}} R\delta\sigma = t$$

$$s = L^{\mathrm{lin}}\tau'\sigma' \overset{O_3}{\Longleftarrow\!\!\otimes} L^{\mathrm{lin}}\tau\sigma' \underset{<n}{\overset{\geq_{\mathcal{P}} p\cdot F^{\mathrm{nl}}_L}{\longrightarrow\!\!\!\!\!\rightarrow}} L\delta\sigma' \overset{O_2}{\Longleftarrow\!\!\otimes} L\delta\sigma \underset{\mathcal{R};n}{\longrightarrow} R\delta\sigma = t$$

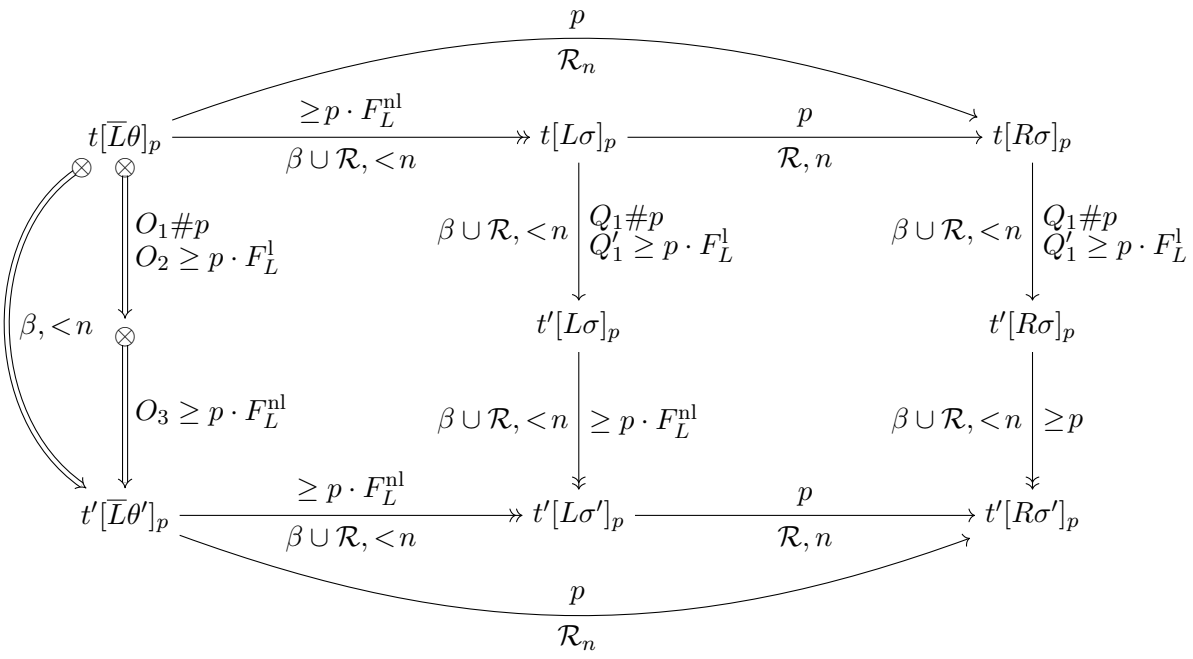


Figure 4.3: DD for the mixed peak: $\underset{\beta;k}{\Longleftarrow\!\!\otimes} \underset{\mathcal{R},n}{\Longrightarrow}$ with $k < n$. See Lemma 4.4.6.

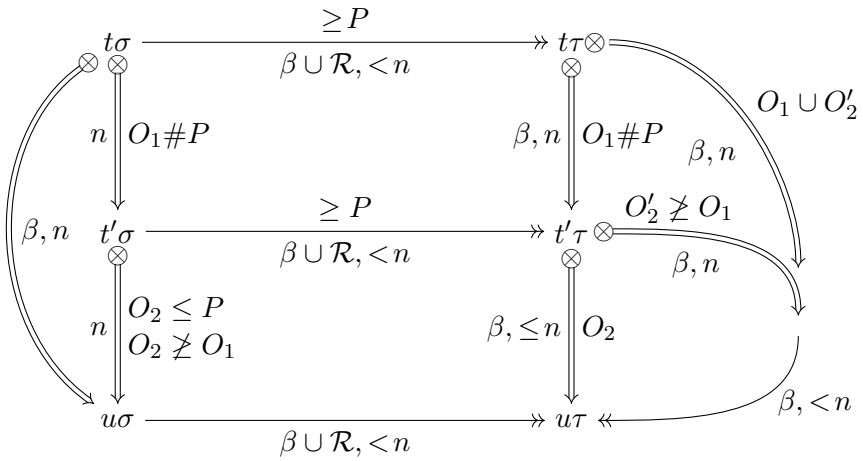


Figure 4.4: DD for the mixed peak: $\overset{O}{\underset{\beta,n}{\Longleftarrow\!\!\otimes}}\ \overset{\geq P}{\underset{\beta\cup\mathcal{R};<n}{\twoheadrightarrow}}$ with $O \not\geq_{\mathcal{P}} P$. See Lemma 4.4.7.

The assumed confluence of rewriting with label $< n$ allows to collapse τ' into δ' such that $\delta' \underset{\beta\mathcal{R};<n}{\twoheadleftarrow} \delta$ and stability then gives $s \overset{\geq_{\mathcal{P}} F^{\mathrm{nl}}_L}{\underset{\mathcal{R}\beta;<n}{\twoheadrightarrow}} L\delta'\sigma' \overset{\geq_{\mathcal{P}} F^{\mathrm{nl}}_L}{\underset{\mathcal{R}\beta;<n}{\twoheadleftarrow}} L\delta\sigma' \underset{\mathcal{R};n}{\longrightarrow} R\delta\sigma' \overset{O_2}{\underset{\beta;n}{\Longleftarrow\!\!\otimes}} R\delta\sigma = t$. Finally, stability and **(H2)** allow the sub-rewriting steps to be grouped into our joining sequence $s \underset{\mathcal{R};n}{\longrightarrow\!\!\triangleright} R\delta'\sigma' \overset{\geq_{\mathcal{P}} F^{\mathrm{nl}}_L}{\underset{\mathcal{R}\beta;<n}{\twoheadleftarrow}} R\delta\sigma = t$ which is a decreasing diagram. □

Lemma 4.4.7. *If* $u \overset{O}{\underset{\beta;n}{\Longleftarrow\!\!\otimes}} s \overset{q}{\underset{\mathcal{R};<n}{\triangleright\!\!\longrightarrow}} v$, *then* $u \overset{Q}{\underset{\mathcal{R};<n}{\triangleright\!\!\twoheadrightarrow}} w \underset{\beta;<n}{\twoheadleftarrow\!\!\triangleleft} v' \overset{}{\underset{\beta;n}{\Longleftarrow\!\!\otimes}} v$.

Proof. This proof is illustrated in Figure 4.4 and follows the same structure as that of Lemma 4.4.6. □

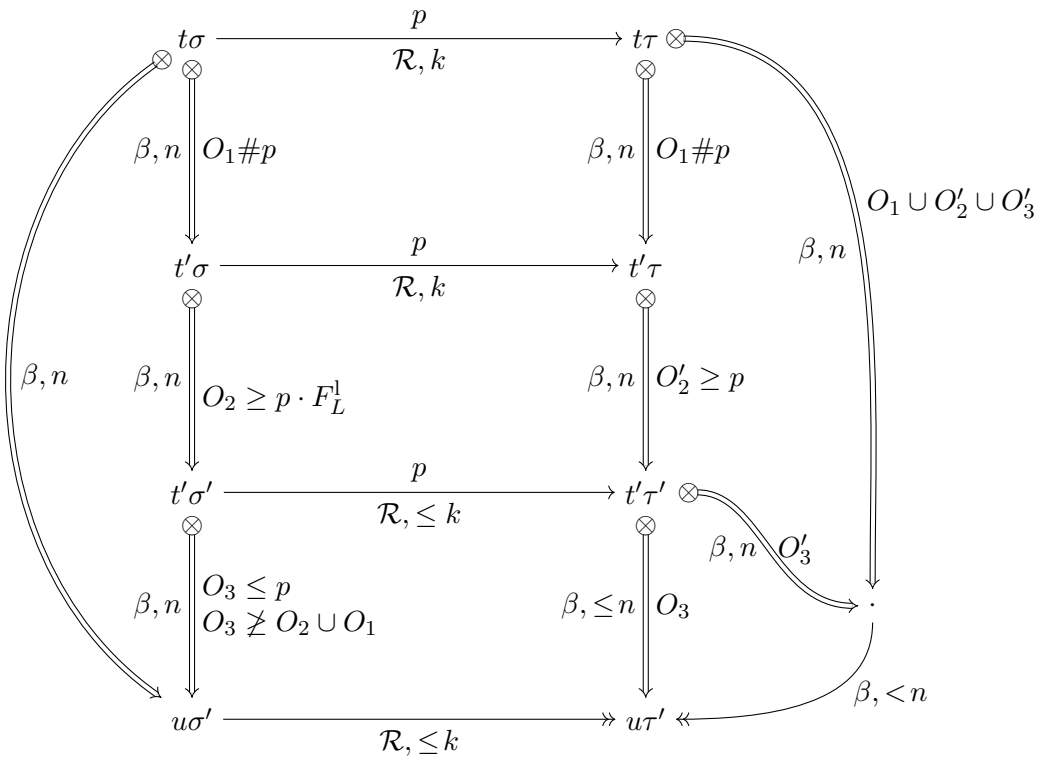


Figure 4.5: DD for the mixed peak: $\underset{\beta;n}{\Longleftarrow\!\!\otimes}\ \underset{\mathcal{R};k}{\longrightarrow}$ with $k \leq n$. See Lemma 4.4.8.

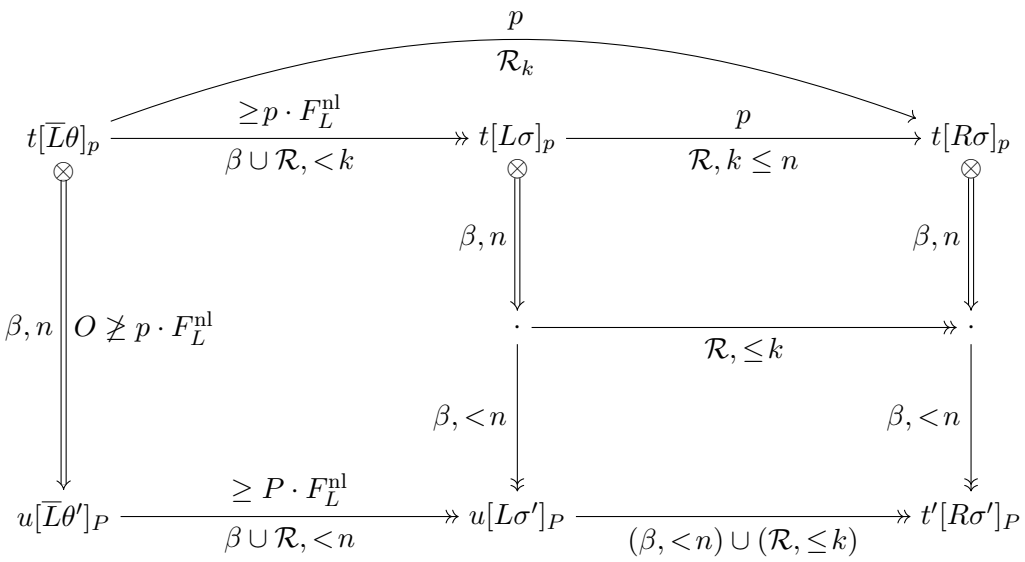


Figure 4.6: DD for the mixed peak: $\underset{\beta,n}{\Longleftarrow\!\!\otimes}\ \underset{\mathcal{R},k}{\Rightarrow}$ with $k \leq n$. See Lemma 4.4.8.

Lemma 4.4.8. *If* $u \underset{\beta;n}{\overset{O}{\Longleftarrow\!\!\otimes}} s \underset{\mathcal{R};n}{\overset{p}{\rhd\!\!\longrightarrow}} v$, *then* $u \underset{\mathcal{R};\leq n\cup\beta;<n}{\overset{p}{\longrightarrow\!\!\!\!\!\!\rightarrow}} w \underset{\beta;<n}{\overset{}{\longleftarrow\!\!\!\!\lhd}} v' \underset{\beta;n}{\overset{}{\Longleftarrow\!\!\otimes}} v$.

Proof. This proof is illustrated in Figure 4.5 and Figure 4.6 and follows the same structure as that of Lemma 4.4.6. □

We are now ready for our main result:

Theorem 4.4.9. *Assume a rewrite system* $\mathcal{R} = \mathcal{R}_{\text{fo}} \cup \mathcal{R}_{\text{ho}}$ *and layered sets* $(\mathcal{L}_n)_n$ *of terms such that:*

- ***(H1,2,3,4,5,6)*** *are satisfied,*
- $\mathcal{R}_{\text{fo}}$ *is confluent on confined closed terms,*
- *critical pairs of* $\mathcal{R}_{\text{ho}}$ *and critical pairs of* $\mathcal{R}_{\text{fo}}$ *onto* $\mathcal{R}_{\text{ho}}$ *are joinable.*

Then $\underset{\mathcal{R}\beta}{\longrightarrow}$ *is confluent.*

Proof. As explained at the beginning of the section, we prove the confluence property of the relation $\underset{\beta}{\otimes\!\!\Longrightarrow} \cup \underset{\mathcal{R}}{\rhd\!\!\longrightarrow}$ which satisfies $\underset{\mathcal{R}\beta}{\longrightarrow} \subseteq \underset{\beta}{\otimes\!\!\Longrightarrow} \cup \underset{\mathcal{R}}{\rhd\!\!\longrightarrow} \subseteq \underset{\mathcal{R}\beta}{\longrightarrow\!\!\!\!\!\!\rightarrow}$. The orthogonal β-reduction at level n is labeled with $(n, \bot)$ and $u \underset{\mathcal{R};n}{\rhd\!\!\longrightarrow} v$ is labeled with $(n, [\![u]\!]_n)$ and the order on labels is the lexicographic ordering $(\leq, \preceq)$.

The proof of confluence is done by course-of-values induction on the maximum level l of the considered terms. For the base case, the relation $\underset{\mathcal{R}_{\text{fo}}}{\longrightarrow}$ in the only one operating on the set of confined terms $\mathcal{L}_0 = \mathcal{T}_c$. By assumption it is confluent.

We move to the induction case $l+1$ and we show that all local peaks $s \underset{m}{\longleftarrow} u \underset{n}{\longrightarrow} t$ enjoy a decreasing diagram with respect to our labeling.

If both $m \leq l$ and $n \leq l$, we simply conclude by induction hypothesis. Otherwise, there are 7 cases depending on the labels, positions and kind of rewrite steps in both sides. Note that we will be able to use our joinability lemmas, since their confluence assumption is now provided by the induction hypothesis.

1. $s \underset{\mathcal{R};m}{\overset{p}{\longleftarrow\!\!\!\!\lhd}} u \underset{\mathcal{R};n}{\overset{q}{\rhd\!\!\longrightarrow}} t$, with $p \# q$. Then, we conclude by Lemma 4.4.1 which yields a DD since both part of the label are non-increasing in the facing steps, using **(H6)** .
2. $s \underset{L\to R;l+1}{\overset{p}{\longleftarrow\!\!\!\!\lhd}} u \underset{\mathcal{R};n}{\overset{q}{\rhd\!\!\longrightarrow}} t$ and $q \geq_{\mathcal{P}} p \cdot F^{\text{l}}_L$. Then, we conclude by Lemma 4.4.2 which yields a DD using **(H6)** again.
3. $s \underset{L\to R;l+1}{\overset{p}{\longleftarrow\!\!\!\!\lhd}} u \underset{\mathcal{R};n}{\overset{q}{\rhd\!\!\longrightarrow}} t$ with $q \geq_{\mathcal{P}} p \cdot F^{\text{nl}}_L$. Then, we conclude by Lemma 4.4.3 which yields a decreasing diagram since, by **(H4)**, $n \leq l$.
4. $s \underset{L\to R;l+1}{\overset{p}{\longleftarrow\!\!\!\!\lhd}} u \underset{\mathcal{R};n}{\overset{q}{\rhd\!\!\longrightarrow}} t$ with $q \in p \cdot \mathcal{FP}os(L)$. Then, by Lemma 4.3.8 and the assumption that higher order critical pairs are joinable, $s \underset{\mathcal{R};\leq l}{\longrightarrow\!\!\!\!\!\!\rightarrow} \overset{\geq_{\mathcal{P}} p}{\underset{\mathcal{R}}{\longrightarrow\!\!\!\!\!\!\rightarrow}} \overset{\geq_{\mathcal{P}} p}{\underset{\mathcal{R}}{\longleftarrow\!\!\!\!\!\!\leftarrow}} \underset{\mathcal{R};<n}{\longleftarrow\!\!\!\!\!\!\leftarrow} t$ and **(H6)** ensures that the unlabeled steps are in fact a DD.

5. $s \underset{\beta;l+1}{\Longleftarrow\!\!\otimes} u \underset{\beta;n}{\otimes\!\!\Longrightarrow} t$. Then, we conclude by Lemma 4.4.5.
6. $s \overset{O}{\underset{\beta;l+1}{\Longleftarrow\!\!\otimes}} \underset{\mathcal{R};n}{\overset{q}{\triangleright\!\!\longrightarrow}}$. The case where $p \# Q$ has already been considered, hence we can assume that there is some $o \in O$ such that $o <_{\mathcal{P}} q$, hence $n \leq l$ by **(H5)**. We then conclude by Lemma 4.4.7.
7. $s \underset{\mathcal{R};l+1}{\overset{p}{\longleftarrow\!\!\triangleleft}} u \underset{\beta;n}{\overset{Q}{\otimes\!\!\Longrightarrow}} t$. We conclude here either by Lemma 4.4.8 if $n = l + 1$, or by Lemma 4.4.6 if $n \leq l$. The diagram is decreasing since in both cases there is a single facing sub-rewrite step in the joining sequence. □

The $\mathcal{R}_{\text{fo}}$ relation is merely assumed to be stable, monotonic and confluent on first-order terms. The proof of its confluence can be done using better-known first-order techniques since β-reduction is disabled in the confined layer.

In [ADJL16], the first-order relation $\mathcal{R}_{\text{fo}}$, called normalized rewriting, is generated by a set of rules R, a set of simplifiers S, and a set of equations E. Rewriting operates on terms in normal form with respect to S modulo E, and uses pattern matching modulo $S \cup E$. Confluence can be classically reduced to critical pairs joinability (modulo $S \cup E$). Details can be found in [JL12b]. In practice, E is often taken to be associativity and commutativity, while simplifiers can be identity elimination or idempotency. In [ADJL16], confined expressions were natural numbers generated by `0` and `succ`, equipped with several operations, in particular $+$ (addition) and `max` (maximum), both being AC, with simplifiers being identity elimination for both and idempotency for max.

4.5 Example

We carry out an example illustrating how the previous theorem can be applied in practice. We focus on the particular case of the system $\mathcal{R} := \{\mathtt{F}\ X\ X\ Y \to Y\}$ of a single non-left-linear rewrite rule for some $\mathtt{F} \in \mathcal{F}$. This variant of Klop's famous example bears similarities with the rules required for the universe "lifting" operators that we introduce in Chapter 5. Notice that $\mathcal{R}$ satisfies **(H5)** and that $\underset{\mathcal{R}}{\longrightarrow}$ is terminating.

Our goal here is to syntactically restrict the set of terms and stratify it with levels so that the hypotheses **(H0)** to **(H6)** are all satisfied. The set of variables $\mathcal{X}$ is supposed to be split into layers $\mathcal{X}^n$ and we annotate them with their level for readability: $x^n \in \mathcal{X}^n$.

Some terms, but not all of them, can be assigned a level using the following rules:

$$\begin{array}{rcl@{\qquad}rcl} \forall x^n \in \mathcal{X}^n & , & x^n \in \mathcal{L}_n & \forall \mathtt{c} \in \mathcal{F}_c & , & \mathtt{c} \in \mathcal{L}_0 \\ \forall t, u \in \mathcal{L}_n & , & t\ u \in \mathcal{L}_n & \forall \mathtt{c} \in \mathcal{F} \setminus \{\mathtt{F}\} & , & \mathtt{c} \in \mathcal{L}_1 \\ \forall t, u \in \mathcal{L}_{\leq n} & , & \lambda x^n : t.\, u \in \mathcal{L}_n & \forall u \in \mathcal{L}_k, v \in \mathcal{L}_n & , & \mathtt{F}\ u\ v \in \mathcal{L}_{\max(k,n)+1} \end{array}$$

Note that any (unconfined) layer $\mathcal{L}_n$ contains all terms of the pure λ-calculus without symbols. In particular β-reduction remains non-terminating (on leveled terms) and, for instance, if $x, y \in \mathcal{X}^n$ then $(\lambda x : y.\, x\ x)\ (\lambda x : y.\, x\ x) \in \mathcal{L}_n$. We have chosen here to classify all symbols, except `F` in the first non-confined layer $\mathcal{L}_1$ but we could also have split them

among distinct layers. Note that the symbol $\mathtt{F}$, however, does not belong to any layer and every occurrence of it must be applied to at least two arguments to obtain a leveled term. In a way $\mathtt{F}$ is forced to have arity 2. Its two first arguments correspond to non-linear positions and must belong to a layer strictly lower than that of the whole term.

Lemma 4.5.1. *Layers $\mathcal{L}_n$ are pairwise disjoint and satisfy **(H1)**, **(H2)**, **(H3)** and **(H4)**.*

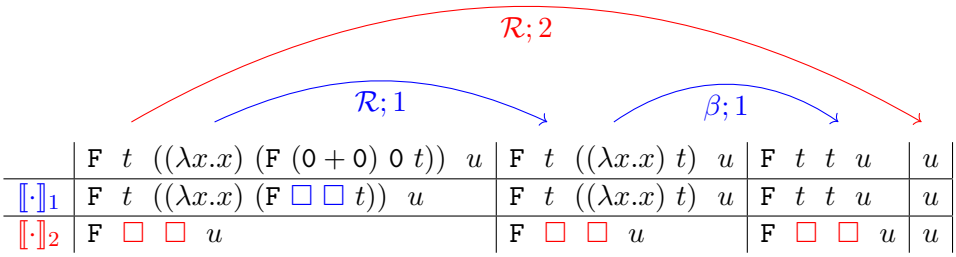
Proof. The level of a term is syntactically uniquely defined as only a single level assigning rule applies to it. All hypothesis are checked by induction on t. □

A strong property of this system is that it does not allow unsafe λ-terms such as $\lambda x\!:\!t.\,\mathtt{F}\ x\ x$ to belong to any layer. This term is a level-increasing function. If it were at level n then it could be applied to an other term $t \in \mathcal{L}_n$ and the application would β-reduce to $\mathtt{F}\ t\ t \in \mathcal{L}_{n+1}$ contradicting **(H2)**.

Finally, we need to provide a measure of terms at each level such that the measure at level n is strictly decreasing with $\mathcal{R}$-rewriting at level n and non-increasing with $\beta \cup \mathcal{R}$ at level strictly lower than n. As mentioned in Section 4.4, this can be done by means of an erasing function.

Definition 4.5.2. *We define the measure function $[\![u]\!]_n$ from terms to terms such that $[\![u]\!]_n := \Box \in \mathcal{F}$ if $u \in \mathcal{L}_{<n}$ and otherwise $[\![u\ v]\!]_n := [\![u]\!]_n\ [\![v]\!]_n$, $[\![\lambda x\!:\!t.\,u]\!]_n := \lambda x\!:\![\![t]\!]_n.\,[\![u]\!]_n$, $[\![x]\!]_n := x$ and $[\![\mathtt{c}]\!]_n := \mathtt{c}$.*

Example 1: For instance, assuming $t \in \mathcal{L}_1$, $u \in \mathcal{L}_2$ and $\mathtt{0}, + \in \mathcal{L}_0$:



The $[\![\cdot]\!]_1$ measure is invariant by rewriting at the confined level $\mathcal{L}_0$ since terms at this level are erased into a $\Box$. Similarly $[\![\cdot]\!]_2$ is invariant by both β and $\mathcal{R}$ rewriting at level 1 and 0.

Lemma 4.5.3. *The measure function $[\![\cdot]\!]_n$ satisfies **(H6)** with $\mathcal{E} := \mathcal{T}$ equipped with the $\xrightarrow[\mathcal{R}]{}^{+}$ order which is well-founded.*

Proof. By induction on the reduced term, using **(H1)** for the first case and **(H4)** for the second. □

4.6 Future work

We conclude with a few remarks on how the work presented in this chapter and the previous one could be extended. Our confluence result for non-left-linear systems with β restricted to a layered subset of terms is still too restrictive to show the confluence of practical rewrite systems. We believe it can be extended in several ways.

4.6.1 Simple type layering

Our layering conditions is still quite constraining and could probably be relaxed while still guaranteeing confluence of non-left-linear rewriting with β.

In Klop's counter example, the term $f := \lambda x.\, \mathtt{F}\ x\ (c\ x)$ should be treated carefully as it is a level-increasing function: if $x \in \mathcal{L}_n$ then occurrences of x in the body of the function forces it to be $\mathcal{L}_{n+1}$. Level-decreasing functions break the guarantee that subterms belong to a lower level and must therefore be forbidden . The function f, however, is not necessarily harmful as long as it is considered as a "special" term of $\mathcal{L}_{n+1}$. In particular it must only be applied to terms in $\mathcal{L}_n$ in order to guarantee that β-redexes $(\lambda x.\, t)\ u$ are reduced with a level-preserving substitution.

In all generality, it should be possible to extend our criteria so that level-increasingness is more precisely labeled but allowed. This can be achieved, for instance, using simple types over natural numbers $f\ :\ \mathcal{L}_n \to \mathcal{L}_{n+1}$. All the above development can be adapted to ensure level preservation, a work that we have not written down yet.

This technique should of course still allow to type non-terminating λ-terms within a layer, including fixpoint operators. For this reason we cannot require that all terms are simply typed, otherwise β would be terminating. Instead applications within a layer must be allowed, if $t, u\ :\ \mathcal{L}_n$, then $t\ u\ :\ \mathcal{L}_n$. Only level increasing functions need to be typed with a product. However unsafe applications, such as the fixpoint operator in Klop's example, can be forbidden using these types. Indeed a fixpoint operator can only be typed with an atomic level $\mathcal{L}_n$ and therefore cannot be applied to f which type is a sort-increasing product.

Example 1: We consider the example of a signature containing two symbols, a "coding" symbol, $\mathtt{c}$, and an "uncoding" symbol, $\mathtt{u}$, see 5.5.2 for context. We consider an empty confined layer, a first layer $\mathcal{L}_1$ containing "sort-representing" terms and an other layer $\mathcal{L}_2$ on top. The layering can be defined such that if $t \in \mathcal{L}_1$ and $u \in \mathcal{L}_2$ then $\mathtt{c}(t, u) \in \mathcal{L}_2$ and $\mathtt{u}(t, u) \in \mathcal{L}_2$. We consider a single terminating, level-preserving non-left-linear rewrite rule:

$$\mathtt{c}(S, \mathtt{u}(S, T)) \longrightarrow T$$

It is easy to see that this rule satisfies the conditions for Theorem 4.4.9 using our layering and syntactical restrictions on $\mathtt{c}$ and $\mathtt{u}$. However, our integer-leveled layering is a bit restrictive and forbids terms such as $\lambda s.\, \mathtt{u}(s, t)$. Indeed, the body of the λ-abstraction is at level 2 while the variable s and therefore the abstraction itself are both at level 1, breaking **(H1)**.

Relying on product layers would allow to consider function building a term in the top layer $\mathcal{L}_2$ from an argument in the lower layer $\mathcal{L}_1$, such as $\lambda x.\, \lambda s.\, \mathtt{u}(s, \mathtt{f}\ x)$, assuming $x \in \mathcal{L}_2$ and $s \in \mathcal{L}_1$. Note, however, that the encoding of the prenex universe polymorphism of definitions, introduced in Chapter 8, does not rely on abstractions over sorts.

4.6.2 Relaxed linear compatibility

The linear compatibility condition **(H5)** could be relaxed to allow the non-linear fringes of overlap to intersect. This would make the computation of critical peak more difficult but should not compromise confluence. Indeed, the commutation step in the Critical peak lemma 3.3.5, see Figure 9.3, can be replaced with an induction hypothesis using the confluence of the layer beneath to recreate equalization steps in both overlapping redexes so that they are instances of a critical pair.

Example 2: The rewrite rule in Example 1 could be considered together with its counterpart $\mathtt{u}(S, \mathtt{c}(S,T)) \longrightarrow T$. The two rules overlap in the following two critical peaks.

$$\begin{array}{ccccc} \mathtt{u}(S,T) & \xleftarrow{\Lambda} & \mathtt{u}(S, \mathtt{c}(S, \mathtt{u}(S,T))) & \xrightarrow{2} & \mathtt{u}(S,T) \\ \mathtt{c}(S,T) & \xleftarrow{\Lambda} & \mathtt{c}(S, \mathtt{u}(S, \mathtt{c}(S,T))) & \xrightarrow{2} & \mathtt{c}(S,T) \end{array}$$

Both critical pairs are trivial and the system is still terminating yet our criteria does not apply since the non-linear fringes in the overlap are not parallel.

4.6.3 Orthogonal sub-rewriting

We *conjecture* that Theorem 3.5.9 also extends to sub-rewriting with non-left-linear rewrite rules.

Conjecture 4.6.1. *Let $\mathcal{R} = \mathcal{R}_{\mathrm{fo}} \cup \mathcal{R}_{\mathrm{ho}}$ be a rewriting system and layered sets $(\mathcal{L}_n)_n$ of terms such that*

- ***(H1,2,3,4,5)** are satisfied,*
- *$\mathcal{R}_{\mathrm{fo}}$ is confined and confluent on closed terms,*
- *orthogonal critical pairs of $\mathcal{R}_{\mathrm{ho}}$ and critical pairs of $\mathcal{R}_{\mathrm{fo}}$ onto $\mathcal{R}_{\mathrm{ho}}$ admit decreasing diagrams compatible with level labeling, assuming that β-steps have smaller weights than $\mathcal{R}$-steps when in the same layer.*

Then $\underset{\mathcal{R}\beta}{\longrightarrow}$ is confluent on well-layered terms.

We believe that the proof of Theorem 4.4.9 can be adapted the same way it has been for Theorem 3.5.5. Considering *orthogonal $\mathcal{R}$-rewriting* does not change anything in the cases of peaks at different levels since the lower level multi-steps always occur at disjoint positions or nested below the higher levels steps. Mixed β-$\mathcal{R}$ peaks should not be a problem either since they do not overlap. Therefore, it only remains to check that the decreasing diagrams for orthogonal critical pairs extends to any local peak. The technical hypothesis **(H6)** is however no longer required to close $\mathcal{R}$-$\mathcal{R}$ peaks but we still need to prove the equivalent of Lemma 4.4.4 for $\mathcal{R}$ rewriting.

4.6.4 Linearizing systems

In a typed setting such as the $\lambda\Pi_{\equiv}$, the non-linearity of a rewrite rule may be enforced by its well-typedness. It is safe to consider the linearized version of a rewrite rule if the following conditions are met:

1. its linearized version is type-preserving;
2. it defines a confluent system together with β;
3. all well-typed instances of the left-hand side are such that all the images of the linearized versions of a same non-linear variables are convertible.

Example 3: Consider the rewrite rule in Example 1 together with the signature:

$$\begin{array}{llllll} \mathbb{C} & : & * & \mathtt{U}_{\square} & : & \boldsymbol{\mathcal{S}} \to * \\ \boldsymbol{\mathcal{S}} & : & * & \mathtt{s}_{\perp} & : & \boldsymbol{\mathcal{S}} \end{array} \qquad \begin{array}{lll} \mathtt{c} & : & \Pi s : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_s \to \mathbb{C} \\ \mathtt{u} & : & \Pi s : \boldsymbol{\mathcal{S}}.\, \mathbb{C} \to \mathtt{U}_s \end{array}$$

Both the non-left-linear and left-linear versions of the rewrite rule are type-preserving (with type $\mathbb{C}$), the linear version defines a rewrite system confluent with β. Besides, any instance $\mathtt{c}(s, \mathtt{u}(s', t))$ of the left-hand side must satisfy, by inversion, $s \equiv_{\beta\mathcal{R}} s'$ to be well-typed which ensures the rule is exclusively used with "almost" instances of the non-left-linear rule.

Note that this technique cannot be used systematically. The extra rule in Example 2, for instance, is only well-typed in its non-left-linear version and no information can be deduced from the well-typedness of left-hand sides instances, contradicting the first and third items. Considering the extra rules $\mathtt{U}_{\mathtt{s}_\perp} \longrightarrow \mathbb{C}$, $\mathtt{c}(\mathtt{s}_\perp, T) \longrightarrow T$ and $\mathtt{u}(\mathtt{s}_\perp, T) \longrightarrow T$ defines a signature in which all rules are still well-typed, however, the linearized rule now has two non-joinable critical pairs, contradicting the second item:

$$T \xleftarrow{\Lambda} \mathtt{c}(\mathtt{s}_\perp, \mathtt{u}(S, T)) \xrightarrow{\Lambda} \mathtt{u}(S, T) \qquad \text{and} \qquad T \xleftarrow{\Lambda} \mathtt{c}(S, \mathtt{u}(\mathtt{s}_\perp, T)) \xrightarrow{2} \mathtt{c}(S, T)$$

It is not clear, however, that the linearized version of the rule, even if the non-linearity is ensured with typing, has the same behavior as its non-left-linear version. Indeed, the well-typedness of an instance ensures the *joinability* of subterms corresponding to the same non-linear variable *in the linearized system*. This technique is closer to the confluent $\longmapsto$ extension of Klop's example mentioned before:

$$\mathtt{F}\ X\ Y \longmapsto \mathtt{a} \qquad \text{iff} \quad X \twoheadleftarrow\!\!\twoheadrightarrow Y$$

This relation is proven confluent, however it is defined using itself which complicates its study and makes it impossible to compute in practice.

4.6.5 Rewriting modulo equivalence

It is often useful to consider equivalence relations on terms that are not conveniently represented as rewrite rules. If this relation behaves well, term rewriting, as defined in Chapter 2, can be extended to operate on equivalence classes of terms rather than on syntactical terms.

Definition 4.6.2 (Rewriting modulo $\equiv_E$)**.** *Assume a stable and monotonic equivalence relation $\equiv_E$, a meta-term u, a position $p \in \mathcal{P}os(u)$, and a rule (L, R). We define rewriting* modulo $\equiv_E$*, such that $u \xrightarrow[(L,R)_E]{p} v$ iff $u|_p \equiv_E L\gamma$ for some meta-substitution γ and $v = u[R\gamma]_p$.*

Lemma 4.6.3. *We have* $\xrightarrow[(L,R)]{p} \subseteq \xrightarrow[(L,R)_E]{p} \subseteq \equiv_E^{\geq_{\mathcal{P}} p} \xrightarrow[(L,R)]{p}$.

In the context of encoding common algebras, it is often useful to have some symbols enjoy the property of being both associative and commutative. This property cannot however be represented using a terminating rewrite system as defined previously. For this reason it is convenient to define AC conversion of terms built from a considered subset $\mathcal{F}_{\text{AC}} \subseteq \mathcal{F}$ of symbols.

Definition 4.6.4 (AC conversion)**.** *We define* associative-commutative conversion *as the smallest reflexive, symmetric, transitive and monotonic relation on meta-terms such that for all $\mathtt{c} \in \mathcal{F}_{\text{AC}}$, and terms t, u, v, we have $\mathtt{c}\ t\ (\mathtt{c}\ u\ v) \equiv_{\text{AC}} \mathtt{c}\ (\mathtt{c}\ t\ u)\ v$ and $\mathtt{c}\ u\ v \equiv_{\text{AC}} \mathtt{c}\ v\ u$.*

If $\mathtt{c}$ is $+$, $\vee$ or $\wedge$, we use the infix notation $t + u$ and allow to omit parenthesis whenever terms are considered "modulo AC": $t + u + v := t + (u + v) \equiv_{\text{AC}} (t + u) + v$. If $\mathtt{c}$ is $\mathtt{max}$, we use the multi-ary notation $\mathtt{max}(t, u, v) := \mathtt{max}\ t\ (\mathtt{max}\ u\ v) \equiv_{\text{AC}} \mathtt{max}(v, u, t)$.

The study of rewriting modulo equivalence is made harder by the fact that equivalences have no reason to preserve positions in a term, or the general shape of a term. In practice, matching and unification modulo equivalences is much more complicated. In the case of AC, for instance, matching is decidable but already an NP-complete problem and unification problems have a computably enumerable but infinite complete set of unifiers that can be finitely described.

Despite these difficulties, results both from Chapter 3 and Chapter 4 could be adapted to rewriting modulo well-behaved equivalence classes such as $\equiv_{\text{AC}}$.

The rewrite engine of the Dedukti tool has been extended by the author to allow rewriting modulo the associativity and commutativity of a declared subset of symbols. For details on how to efficiently implement matching modulo AC, we refer to the work of Contejean [Con04] adapted to work with Maranget's compiled decision trees [Mar08].

Part II

Embedding Higher-order Logics with Universes

Chapter 5

Embedding Cumulativity

We move on from confluence criteria to study, in the rest of this manuscript, applications of higher-order term rewriting to embeddings of logical systems in $\lambda\Pi_{\equiv}$. Our main target, described in Chapter 9, is the description and practical implementation of a translator from Coq to Dedukti. To achieve this goal, we provide a representation of universe polymorphism, defined, studied and proven correct in Chapter 8 which consists in one of the main contributions of our work. Prior to this, we introduce in Chapter 6 universe variables in the Calculus of Constructions, and universe polymorphism in Chapter 7. In Chapter 5, we review the different existing ways that allow us to encode cumulative systems in $\lambda\Pi_{\equiv}$, and develop new techniques of our own that are used in the subsequent chapters.

There are two main challenges of encoding discussed in the following two preliminary chapters. Rewriting techniques embedding an infinite hierarchy of sorts containing algebraic universe expressions are introduced in Chapter 6. In this chapter, we begin our journey by looking into the more general context of Cumulative Type Systems which already present some difficulties, the sets of sorts can be potentially infinite or surprisingly structured, the inference rules are not type-directed and uniqueness of type does not hold in presence of subtyping.

We begin with a description of the latest developments regarding the embedding of type systems in $\lambda\Pi_{\equiv}$. We introduce in this chapter three families of type systems.

- PTS is the simplest, even if its encoding already requires to reflect the β conversion of the original system which is best done directly using the β conversion of the encoding system, $\lambda\Pi_{\equiv}$, therefore defining a *shallow* encoding.
- $\text{PTS}^{\preceq}$ is more evolved and features sort subtyping therefore breaking the uniqueness of type, which must hold in $\lambda\Pi_{\equiv}$. As done in Assaf, it can still be encoded by recovering uniqueness of type by explicitly annotating subtyping in terms.
- CTS extends subtyping covariantly on the product types' codomain. This system is the core logic of our long-term target, Coq. As pointed out by Assaf, the encoding of $\text{PTS}^{\preceq}$ can be adapted to CTS provided the η conversion is considered both in the original and the encoding systems.

We then discuss the challenges of encoding CTS, without η, which were not covered by previous work:

- the more general subtyping relation can be explicitly annotated using a *cast* operator extending Assaf's *lift* on sorts. Equivalently, the link between a term's translation and its (non unique) typing derivations can be recovered by directly considering a translation of its typing derivations instead.
- the infinite and complex structure of universes can be represented using predicates rather than functional sort operators defined with rewrite rules. Relying on provable predicates in the encoding provides some flexibility but requires to ensure their *proof irrelevance* which is achieved here by means of a *private* encoding.

Many type systems for the "à la Church" λ-calculi, from simple types to System F, the lambda-Pi-calculus (also known as LF or $\lambda\Pi$) and the Calculus of Constructions, all share a common syntax and similar typing inference rules. These systems are all well-behaved and the properties they share made them, or rather extensions of them, particularly fit to be used as the core logic of several proof systems such as Agda, Matita or Coq. *Pure type systems*, PTS, are a parameterized family of type theories encompassing all of them. Cousineau and Dowek defined [CD07] an embedding of pure type systems in $\lambda\Pi_{\equiv}$ and a translation function from terms of PTS to $\lambda\Pi_{\equiv}$ terms. They showed that this translation function preserves typing for all PTS satisfying the *uniqueness of type* property, which most useful PTS do.

Just like the set of all sets cannot be a set in set theory, Russell's paradox has shown that it is inconsistent, in type theories, to type a sort with itself, $\texttt{Type} : \texttt{Type}$. A natural way to still reason about types of types, called *sorts*, is to consider a –potentially infinite– hierarchy of them $\texttt{Type}_0 : \texttt{Type}_1 : \texttt{Type}_2 : \ldots$. Each sort $\texttt{Type}_n$ allows to reason about the one lying directly below. Infinitely sorted type systems are already more of a hassle to properly translate. The main difficulty is that they require encoding constructions to be *parameterized by a sort* rather than (infinitely) duplicated. Dependent types allow to properly type the expected parameterized operators but it requires to provide a faithful term representation of sorts.

In Martin-Löf intuitionistic type theory [ML75, ML84] it is suggested that a sort $\texttt{Type}_n$ should not only "contain" (to borrow set theoretic vocabulary) the sort directly below, this sort should also be a "subset" of it. This means that all types inhabiting $\texttt{Type}_n$ also inhabit any sort above, breaking the uniqueness of type property. Encoding these extensions in $\lambda\Pi_{\equiv}$, that does satisfy uniqueness of type, has been extensively studied by Assaf [Ass15a] which adapted the encoding PTS to support some form of sort subtyping by means of explicit casting operators as in the "à la Tarski" term representation.

This property of sorts is called *cumulativity* and can even be extended covariantly on the codomains of product types to define an even more general family of type systems, the *cumulative type systems*, CTS, introduced by Barras [Bar99] and later extensively studied by Lasson [Las12]. There are many challenges in further extending Assaf's encoding to CTS, in particular in the case of specifications featuring an *impredicative* sort such as in the Extended Calculus of Constructions (see Chapter 6).

In this chapter, we first define and recall basic properties of pure type systems. We then consider several extensions featuring various forms of subtyping all stemming from a cumulativity relation on sorts. We introduce previous work related to the embedding of these different systems into $\lambda\Pi_{\equiv}$ and discuss ways around the difficulties and limitations associated with these encodings. Finally we introduce new encoding paradigms relying on *private* symbols, used to extend the conversion but unavailable to the translation mechanism. We argue that they better allow to faithfully represent more complicated universe structures such as non-functionality, floating universes (see Chapter 6) and even universe polymorphism (see Chapter 7).

5.1 Pure Type Systems

Pure Type Systems are parameterized with a *specification* representing for each system the set of its *sorts* and its associated structure defined by two relations on sorts: the axiom and the product relations. Properties of these systems can often be reduced to properties of their specification. Although we quickly introduce it and recall some of its essential properties, a more extensive presentation of pure type systems can be found in the work of Geuvers and Nederhof [GN91] and Barendregt [Bar93].

The first challenge when encoding a PTS into $\lambda\Pi_{\equiv}$ is that types in PTS are also objects. There is no stratification of terms like the one we must have in $\lambda\Pi_{\equiv}$. The next challenge is that PTS may have a lot more sorts than the only-two-sorted $\lambda\Pi_{\equiv}$. It can even feature infinitely many sorts although we only consider finite sets of sorts for now. The idea to overcome both issues is to have two distinct representations of a term's translation, one *as a type* and the other one *as a well-typed term*. This idea is illustrated in Figure 5.1 where arrows are used to represent functions on term, such as the translation or the application of a symbol of the encoding and 

means that A has type B.

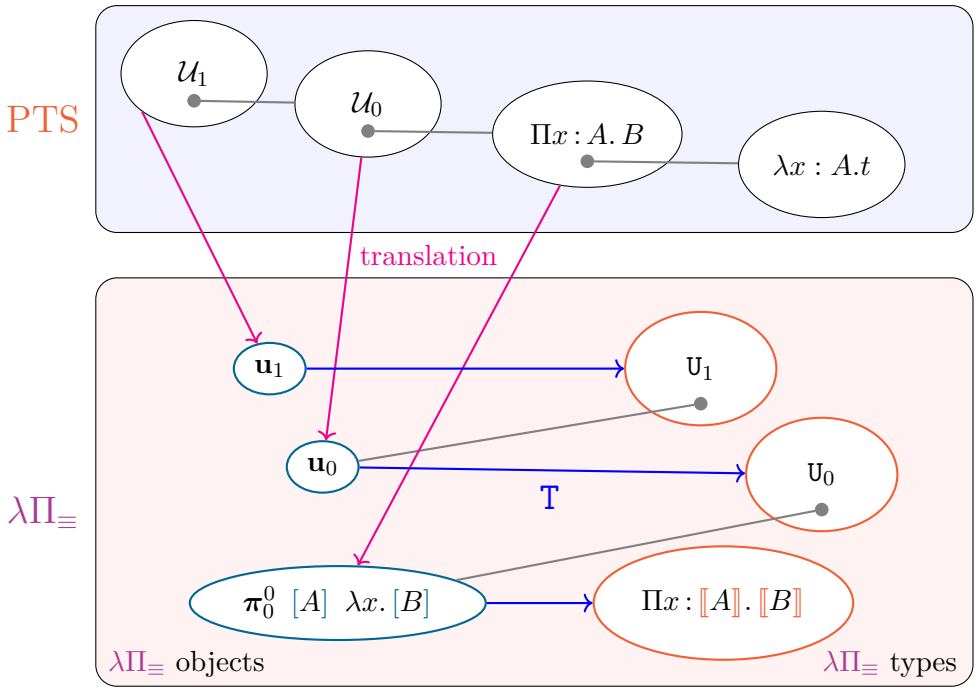


Figure 5.1: Cousineau & Dowek's paradigm

5.1.1 Definition

Definition 5.1.1 (Pure Type Systems Specification and Syntax)**.** *A* PTS *specification is a triple* $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ *where* $\mathcal{S}$ *is a set of* sorts*,* $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ *is the* axioms *relation and* $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ *is the* rules *relation.*

The syntax of $\mathrm{PTS}(\mathcal{S}, \mathcal{A}, \mathcal{R})$ *is as follows:*

$$\begin{array}{rrcl}
\textit{(Variable)} & x, y & \in & \mathcal{X} \\
\textit{(Sorts)} & s & \in & \mathcal{S} \\
\textit{(Terms)} & t, u, A, B & := & x \mid s \mid t\ u \mid \lambda x : A.t \mid \Pi x : A.t \\
\textit{(Context)} & \Gamma & := & \varnothing \mid \Gamma, x : t \\
\textit{(Context WF Judgment)} & & := & \Gamma\ \mathbf{WF}_{\mathcal{P}} \\
\textit{(Typing Judgment)} & & := & \Gamma \vdash_{\mathcal{P}} t : A
\end{array}$$

The typing system *of* $\mathrm{PTS}(\mathcal{S}, \mathcal{A}, \mathcal{R})$ *is defined from the inference rules of Figure 5.2.*

- *A term* t has type A *in the context* Γ *if the judgment* $\Gamma \vdash_{\mathcal{P}} t : A$ *is derivable. We also say that* t *is* well-typed *and that* A *is* inhabited with t *in the context* Γ*.*
- *A context* Γ *is* well-formed *if the judgment* $\Gamma\ \mathbf{WF}_{\mathcal{P}}$ *is derivable.*
- *A term* A *is a* well-formed type *in* Γ *if either* $A = s \in \mathcal{S}$ *or* $\Gamma \vdash_{\mathcal{P}} A : s$*.*

Lemma 5.1.2. *All* PTS*s enjoy the following properties*

- *confluence of the functional reduction* β*;*

$$\frac{}{\varnothing\ \mathbf{WF}_{\mathcal{P}}}\ \mathcal{P}^{\mathbf{WF}}_{\varnothing} \qquad \frac{\Gamma \vdash_{\mathcal{P}} A : s_1 \qquad \Gamma, x : A \vdash_{\mathcal{P}} B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash_{\mathcal{P}} \Pi x : A.\, B : s_3}\ \mathcal{P}_{\Pi}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} A : s \qquad x \notin \Gamma}{\Gamma, x : A\ \mathbf{WF}_{\mathcal{P}}}\ \mathcal{P}^{\mathbf{WF}}_{var} \qquad \frac{\Gamma, x : A \vdash_{\mathcal{P}} M : B \qquad \Gamma \vdash_{\mathcal{P}} \Pi x : A.\, B : s}{\Gamma \vdash_{\mathcal{P}} \lambda x : A.\, M : \Pi x : A.\, B}\ \mathcal{P}_{\lambda}$$

$$\frac{\Gamma\ \mathbf{WF}_{\mathcal{P}} \qquad (x : A) \in \Gamma}{\Gamma \vdash_{\mathcal{P}} x : A}\ \mathcal{P}_{\mathcal{X}} \qquad \frac{\Gamma \vdash_{\mathcal{P}} M : \Pi x : A.\, B \qquad \Gamma \vdash_{\mathcal{P}} N : A}{\Gamma \vdash_{\mathcal{P}} M\ N : B\{x \mapsto N\}}\ \mathcal{P}_{@}$$

$$\frac{\Gamma\ \mathbf{WF}_{\mathcal{P}} \qquad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash_{\mathcal{P}} s_1 : s_2}\ \mathcal{P}_{\mathcal{S}} \qquad \frac{\Gamma \vdash_{\mathcal{P}} M : A \qquad \Gamma \vdash_{\mathcal{P}} B : s \qquad A \equiv_{\beta} B}{\Gamma \vdash_{\mathcal{P}} M : B}\ \mathcal{P}_{\equiv}$$

Figure 5.2: Typing rules for $\mathrm{PTS}(\mathcal{S}, \mathcal{A}, \mathcal{R})$

- *weakening: if* $\Gamma \subseteq \Gamma'$ *then* $\Gamma \vdash_{\mathcal{P}} t : A \Rightarrow \Gamma' \vdash_{\mathcal{P}} t : A$;
- *inversion properties linking typable terms to the shape of their type;*
- *correctness of typing: if* $\Gamma \vdash_{\mathcal{P}} t : A$, *then* A *is a well-formed type;*
- *product compatibility and subject reduction.*

Definition 5.1.3. *The set* $\mathcal{S}^{\top} \subseteq \mathcal{S}$ *of* top-sorts *is such that* $s \in \mathcal{S}^{\top} \Leftrightarrow \forall s' \in \mathcal{S}, (s, s') \notin \mathcal{A}$.

A PTS *specification is* functional *if* $\mathcal{A}$ *and* $\mathcal{R}$ *are functional relations. If that is the case, we write* $\mathcal{A}(s)$ *(resp.* $\mathcal{R}(s_1, s_2)$*) the unique sort* s' *such that* $\mathcal{A}(s, s')$ *(resp.* $\mathcal{R}(s_1, s_2, s')$*).*

A PTS *specification is* full *if* $\mathcal{R}$ *is a total relation.*

A PTS *specification is* complete *if* $\mathcal{A}$ *is a total relation (*$\mathcal{S}^{\top} = \varnothing$*).*

Lemma 5.1.4 (Uniqueness of type)**.** *In a functional specification, if* $\Gamma \vdash_{\mathcal{P}} t : A$ *and* $\Gamma \vdash_{\mathcal{P}} t : B$, *then* $A \equiv_{\beta} B$.

5.1.2 Embedding in the lambda-Pi-calculus modulo theory

Definition 5.1.5. *Assume a finite and functional specification* $(\mathcal{S}, \mathcal{A}, \mathcal{R})$. *We define* $\mathcal{D}[\mathrm{PTS}(\mathcal{S}, \mathcal{A}, \mathcal{R})]$ *as the following signature which is permanently well-typed (see Definition 2.3.20) in* $\lambda\Pi_{\equiv}$.

$$\begin{array}{rcll}
\mathtt{U}_s & : & * & \forall s \in \mathcal{S} \\
\mathtt{T}_s & : & \mathtt{U}_s \to * & \forall s \in \mathcal{S} \\
\mathbf{u}_{s_1} & : & \mathtt{U}_{s_2} & \forall (s_1, s_2) \in \mathcal{A} \\
\boldsymbol{\pi}^{s_2}_{s_1} & : & \Pi a : \mathtt{U}_{s_1}.\, (\mathtt{T}_{s_1}\ a \to \mathtt{U}_{s_2}) \to \mathtt{U}_{s_3} & \forall (s_1, s_2, s_3) \in \mathcal{R} \\
\mathtt{T}_{s_1}\ \mathbf{u}_{s_2} & \longrightarrow & \mathtt{U}_{s_2} & \forall (s_1, s_2) \in \mathcal{A} \\
\mathtt{T}_{s_3}\ (\boldsymbol{\pi}^{s_2}_{s_1}\ A\ \lambda x.\, B[x]) & \longrightarrow & \Pi x : \mathtt{T}_{s_1}\ A.\, \mathtt{T}_{s_2}\ B[x] & \forall (s_1, s_2, s_3) \in \mathcal{R}
\end{array}$$

Note that the encoding symbols are duplicated for each possible sort level that they can be used with. For instance, if $\mathcal{S} := \{*, \Box\}$ and $\mathcal{R} := \{(*,*,*),(*,\Box,\Box)\}$ then the encoding features two distinct product encoding symbols, $\boldsymbol{\pi}^{*}_{*}$ and $\boldsymbol{\pi}^{\Box}_{*}$.

Just like in Martin-Löf's intuitionistic type theory [ML84] this encoding relies on a "Tarski-style" representation of terms. Whenever a term t is used as a well-typed object in context Γ, its term representation $[t]_\Gamma$, a $\lambda\Pi_{\equiv}$ object, is used. Its type representation, $[\![t]\!]_\Gamma$, a $\lambda\Pi_{\equiv}$ type, is used to represent t as a type. Well-typed terms that do not correspond to a type only have a term representation while, for instance top sorts only have a type representation.

Definition 5.1.6. *Assume a well-formed context Γ. We define the translation $[t]_\Gamma$ for all term t well-typed in Γ as a $\lambda\Pi_{\equiv}$ object term by induction on t:*

$$
\begin{array}{rcll}
[x]_\Gamma & := & x & \\
[s]_\Gamma & := & \mathbf{u}_s & \text{where } s \in \mathcal{S} \\
[t\ u]_\Gamma & := & [t]_\Gamma\ [u]_\Gamma & \\
[\lambda x : A.\, t]_\Gamma & := & \lambda x : \mathtt{T}_s\ [A]_\Gamma.\, [t]_{\Gamma, x:A} & \text{where } \Gamma \vdash_{\mathcal{P}} A : s \\
[\Pi x : A.\, B]_\Gamma & := & \boldsymbol{\pi}^{s_2}_{s_1}\ [A]_\Gamma\ (\lambda x.\, [B]_{\Gamma, x:A}) & \text{where } \Gamma \vdash_{\mathcal{P}} A : s_1 \text{ and } \Gamma, x : A \vdash_{\mathcal{P}} B : s_2
\end{array}
$$

We define $[\![A]\!]_\Gamma$ for all type A well-formed in Γ as a $\lambda\Pi_{\equiv}$ type term by induction on A and $[\![\Gamma]\!]$ as $\lambda\Pi_{\equiv}$ context by induction on Γ:

$$
\begin{array}{rcll}
[\![A]\!]_\Gamma & := & \mathtt{T}_s\ [A]_\Gamma & \text{where } \Gamma \vdash_{\mathcal{P}} A : s \\
[\![s]\!]_\Gamma & := & \mathtt{U}_s & \text{where } s \in \mathcal{S}
\end{array}
\qquad
\begin{array}{rcl}
[\![\varnothing]\!] & := & \varnothing \\
[\![\Gamma, x : A]\!] & := & [\![\Gamma]\!], x : [\![A]\!]_\Gamma
\end{array}
$$

The translation is not defined on terms but on terms *together with a context* in which they are well-typed. Note that the encoding signature could easily be adapted to non-functional specifications by having several copies of $\mathbf{u}_{s_1}$ (and $\boldsymbol{\pi}^{s_2}_{s_1}$) for all $(s_1, s_2) \in \mathcal{A}$. However the translation function is only well-defined on PTS with a functional specification since it requires the sort of a type to be uniquely determined. The first rewrite rule from Definition 5.1.5 ensures that if $s \in \mathcal{S}$ and, at the same time, $\Gamma \vdash_{\mathcal{P}} s : s'$, then both possible translations of s yield convertible terms: $\mathtt{T}_{s'}\ \mathbf{u}_s \equiv_{\beta\mathcal{R}} \mathtt{U}_s$. The second rewrite rule ensures that $[\![\Pi x : A.\, B]\!]_\Gamma \equiv_{\beta\mathcal{R}} \Pi x : [\![A]\!]_\Gamma.\, [\![B]\!]_{\Gamma, x:A}$ which allows for a shallow embedding: PTS functions are translated to $\lambda\Pi_{\equiv}$ functions.

The proof of correctness of this encoding is done in three steps, usually in that order.

Lemma 5.1.7 (Preservation of substitution)**.** *If $\Gamma, x : A, \Gamma' \vdash_{\mathcal{P}} t : B$ and $\Gamma \vdash_{\mathcal{P}} u : A$, then $[t]_{\Gamma, x:A, \Gamma'}\{x \mapsto [u]_\Gamma\} \equiv_{\beta\mathcal{R}} [t\{x \mapsto u\}]_{\Gamma, \Gamma'\{x \mapsto u\}}$.*

Note that the right-hand side is well-defined by the substitution lemma in PTS. This lemma is mostly required to prove the following.

Lemma 5.1.8 (Preservation of conversion)**.** *If $\Gamma \vdash_{\mathcal{P}} t : A$, $\Gamma \vdash_{\mathcal{P}} u : B$ and $t \equiv_\beta u$ then $[t]_\Gamma \equiv_{\beta\mathcal{R}} [u]_\Gamma$.*

Which is in turn required to prove the compatibility of the translation with the conversion typing rule $\mathcal{P}_{\equiv}$.

Lemma 5.1.9 (Preservation of typing). *If* $\Gamma \vdash_{\mathcal{P}} t : A$ *then* $\mathcal{D}[\text{PTS}]; [\![\Gamma]\!] \vdash_{\mathcal{D}} [t]_\Gamma : [\![A]\!]_\Gamma$.

The proofs of these three lemmas are relatively straightforward and can be found in [CD07].

5.2 Introducing cumulativity

In some cases, functionality of the axiom and rule relations is not strictly required to define a well-behaved system. Consider for instance the following artificial extension of $\lambda\Pi$: $\mathcal{S} = \{*, \Box, \triangle\}$, $\mathcal{A} = \{(*, \Box), (*, \triangle)\}$ and $\mathcal{R} = \{(*, *, *), (*, \Box, \Box), (*, \triangle, \triangle), (*, \Box, \triangle)\}$. In this non-functional specification, all sorts and products of type $\Box$ are systematically of type $\triangle$ too. In a sense, the sort $\Box$ is a *subtype* of the sort $\triangle$, which we write $\Box \preceq \triangle$.

The $\text{PTS}^{\preceq}$ extends the PTS to allow such a safe form of non-functionality thanks to a extra *cumulativity* relation, $\mathcal{C} \subseteq \mathcal{S} \times \mathcal{S}$. This relation represents a "subset" relation between sorts, just like the axiom relation represents an inclusion relation. In our example, this means that the subtyping property is extended from sorts and products to any other terms typable in $\Box$, such as applied variables.

Definition 5.2.1 ($\text{PTS}^{\preceq}$ Specification). *A* $\text{PTS}^{\preceq}$ *specification is a quadruple* $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$ *where* $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ *is a* PTS *specification and* $\mathcal{C} \subseteq \mathcal{S} \times \mathcal{S}$ *is the* cumulativity *relation.*

Definition 5.2.2. $\text{PTS}^{\preceq}(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$ *is the type system defined from the typing rules of* $\text{PTS}(\mathcal{S}, \mathcal{A}, \mathcal{R})$ *(see Figure 5.2) together with the extra typing rule*

$$\frac{\Gamma \vdash_{\mathcal{P}} A : s_1 \qquad (s_1, s_2) \in \mathcal{C}}{\Gamma \vdash_{\mathcal{P}} A : s_2}\ \mathcal{P}_{\preceq}$$

The full set of typing rules can be found in Figure 9.4.

If $\mathcal{C}$ is empty, the $\text{PTS}^{\preceq}$ is equivalent to a PTS. If it is not, then the new inference rule naturally breaks the uniqueness of type property even if $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{C}$ are functional relations.

However, Assaf showed [Ass14] that in the particular case of the Extended Calculus of Constructions, ECC, [Luo90, Luo89], introduced in Chapter 6, since the $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{C}$ relations are total functions, subtyping can safely be made explicit by means of a lifting operator $\uparrow_s$ that can be interpreted as a special term constructor such that if $\Gamma \vdash_{\mathcal{P}} t : s$ then $\Gamma \vdash_{\mathcal{P}} \uparrow_s t : \mathcal{C}(s)$.

Our presentation of $\text{PTS}^{\preceq}$ is "Russell-style" and terms have the same syntactical representation whether they play the role of a type or of an inhabitant. This presentation is considered more practical since it keeps the role of a term implicit. In the alternative "à la Tarski" presentation, which we write $\text{PTS}^{\uparrow}$, these two roles are syntactically kept apart. Terms of $\text{PTS}^{\preceq}$ that are both a type and well-typed must therefore have two syntactically different representations in $\text{PTS}^{\uparrow}$. For instance, a sort s can be represented either as the type $\mathtt{U}_s$ or the object $\mathtt{u}_s$ inhabiting $\mathtt{U}_{\mathcal{A}(s)}$. Similarly a product $\Pi x : A.\, B$ is either the type $\Pi x : A.\, B$ or the object $\boldsymbol{\pi}_{s_1}^{s_2}\ x : A.B$ inhabiting $\mathtt{U}_{\mathcal{R}(s_1,s_2)}$ if $\Gamma \vdash_{\mathcal{P}} A : \mathtt{U}_{s_1}$

and $\Gamma, x : A \vdash_{\mathcal{P}} B : \mathtt{U}_{s_2}$. The other term constructions, λ-abstraction, application and variables keep the same representation. Any object term A typed in a sort $\mathtt{U}_s$ has a type version written $\mathtt{T}_s\ A$. The typing rules of PTS$^{\uparrow}$ are essentially the same as PTS$^{\preceq}$ with object representations to the left of the $:$ typing operator and type representations to the right. The usual β conversion is extended so as to ensure type representation is compatible with their corresponding "à la Russell" representation. In particular it must ensure that two Russell-equivalent types are convertible.

Making subtyping explicit allows to retrieve the uniqueness of type property as two terms equivalent in the "à la Russell" style but typed differently correspond to different object representations in the "à la Tarski" style. Terms typed with explicit subtyping can be back-translated to terms of PTS$^{\preceq}$ by simply erasing explicit lifts and collapsing type and object representations. This allows to prove the soundness of explicit subtyping as it relies on a set of typing rules identical (up to back-translation) to that of PTS$^{\preceq}$. Assaf showed that this representation is also complete for the Extended Calculus of Constructions provided the β-conversion $\equiv$ relation is extended to satisfy the following *reflection* equations, called as such since they are required to *reflect* Russell-style equivalence.

$$
\begin{aligned}
\mathtt{T}_{s'}\ (\mathtt{u}_s) &\equiv \mathtt{U}_s && \text{if } (s,s') \in \mathcal{A}\\
\mathtt{T}_{\mathcal{R}(s_1,s_2)}\ (\boldsymbol{\pi}^{s_2}_{s_1}\ x{:}A.B) &\equiv \Pi x : \mathtt{T}_{s_1}\ A.\, \mathtt{T}_{s_2}\ B &&\\
\mathtt{T}_{s'}\ (\uparrow_s A) &\equiv \mathtt{T}_s\ A && \text{if } (s,s') \in \mathcal{C}\\
\boldsymbol{\pi}^{s_2}_{s'_1}\ x{:}\uparrow_{s_1} A.B &\equiv \uparrow^{\mathcal{R}(s'_1,s_2)}_{\mathcal{R}(s_1,s_2)} (\boldsymbol{\pi}^{s_2}_{s_1}\ x{:}A.B) && \text{if } (s_1,s'_1) \in \mathcal{C}\\
\boldsymbol{\pi}^{s'_2}_{s_1}\ x{:}A.\uparrow_{s_2} B &\equiv \uparrow^{\mathcal{R}(s_1,s'_2)}_{\mathcal{R}(s_1,s_2)} (\boldsymbol{\pi}^{s_2}_{s_1}\ x{:}A.B) && \text{if } (s_2,s'_2) \in \mathcal{C}
\end{aligned}
$$

where $\uparrow^{\mathcal{C}^n(s)}_s t := \uparrow_{\mathcal{C}^{n-1}(s)} \ldots \uparrow_{\mathcal{C}(s)} \uparrow_s t$.

Note that this way of encoding only works because the particular specification chosen here has some useful properties, such as functionality of both $\mathcal{A}$ and $\mathcal{R}$ as well as *compatibility* of the rules relation with the cumulativity relation: $(s_1,s'_1) \in \mathcal{C}$, $(s_2,s'_2) \in \mathcal{C} \Rightarrow (\mathcal{R}(s_1,s_2), \mathcal{R}(s'_1,s'_2)) \in \mathcal{C}^*$.

Assuming a finite and functional specification, then the "à la Tarski" PTS$^{\uparrow}$ can be correctly encoded in $\lambda\Pi_{\equiv}$ using the same embedding as for PTS extended with the following extra constructors and rewrite rules

$$
\begin{aligned}
\uparrow_s\ &:\ \mathtt{U}_s \to \mathtt{U}_{s'} && \forall (s,s') \in \mathcal{C}\\
\mathtt{T}_{s'}\ (\uparrow_s t) &\longrightarrow \mathtt{T}_s\ t && \forall (s,s') \in \mathcal{C}\\
\boldsymbol{\pi}^{s_2}_{s'_1}\ (\uparrow_{s_1} a)\ b &\longrightarrow \uparrow^{\mathcal{R}(s'_1,s_2)}_{\mathcal{R}(s_1,s_2)} (\boldsymbol{\pi}^{s_2}_{s_1}\ a\ b) && \forall s_2 \in \mathcal{S}, \forall (s_1,s'_1) \in \mathcal{C}\\
\boldsymbol{\pi}^{s'_2}_{s_1}\ a\ (\lambda x.\uparrow_{s_2} b[x]) &\longrightarrow \uparrow^{\mathcal{R}(s_1,s'_2)}_{\mathcal{R}(s_1,s_2)} (\boldsymbol{\pi}^{s_2}_{s_1}\ a\ (\lambda x : \mathtt{T}_{s_1}\ a.\, b[x])) && \forall s_1 \in \mathcal{S}, \forall (s_2,s'_2) \in \mathcal{C}
\end{aligned}
$$

The translation is then extended to lift-headed terms in a natural way, $[\uparrow_s t]_\Gamma := \uparrow_s [t]_\Gamma$. Since it is directly reflected with the rewrite rules of the system, preservation of conversion is easily extended to equivalence reflection. This encoding of PTS$^{\uparrow}$ can of course also be seen as a correct encoding of PTS$^{\preceq}$ for any specifications such that explicit subtyping is complete.

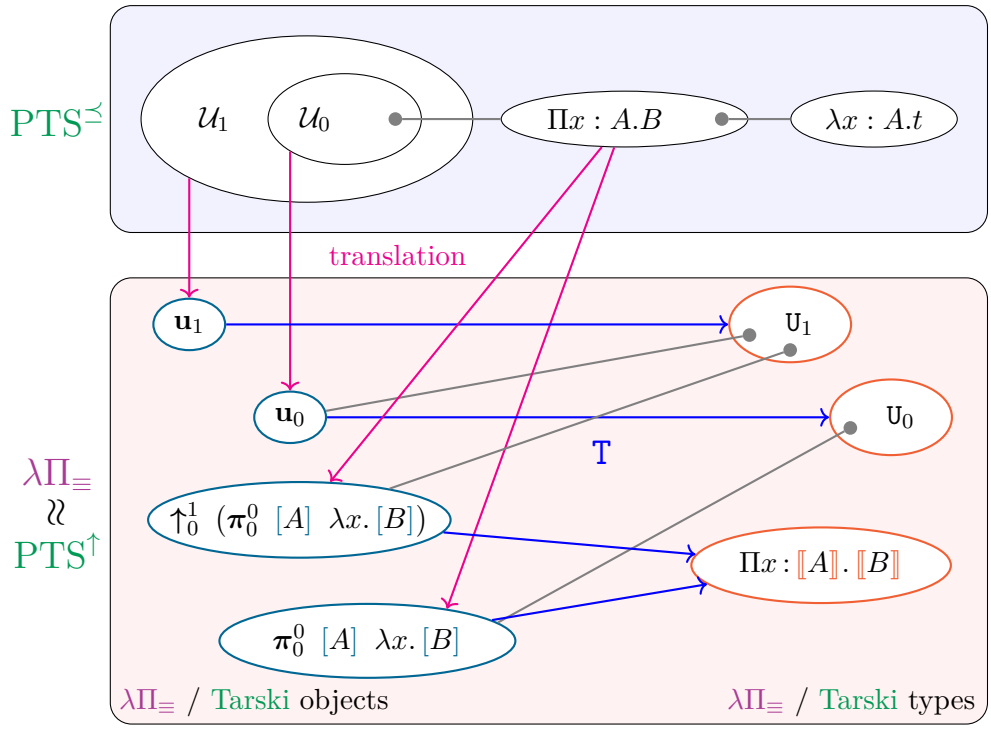


Figure 5.3: Assaf's paradigm

The use of the "à la Tarski" representation and of explicit lifting both come quite naturally in the context of type system translation. A similar approach was, for instance, used by Boulier, Pédrot and Tabareau [BPT17] to provide a syntactical model of the stratified Calculus of Construction in which types are represented with inductive-recursively defined *codes* like was previously done in [Dyb98] which encompasses cumulativity as well. These codes are a representation of (small) types which, they showed, can be decoded to an actual type by means of an, again, inductively defined operator. Just like in our presentation, the type of codes, their constructors and the decoding operator all have to be duplicated and annotated with the sort at which they operate. The use of general rewrite rules, rather than well-founded inductive-recursive definitions, allowed quite some flexibility in the definition of encoding operators. Inductive-like definitions are however more natural and provide desirable properties which often make it worth the effort and probably would have in our setting too.

5.3 Cumulative Type Systems

The primitive form of cumulativity in PTS$^{\preceq}$ is not so well-behaved. Consider, for instance, a simple cumulativity relation $(*, \Box) \in \mathcal{C}$. In this system, if $\vdash_{\mathcal{C}} \lambda x : A.\, t : A \to *$, then necessarily $x : A \vdash_{\mathcal{C}} t : *$ and $\vdash_{\mathcal{C}} \lambda x : A.\, t : A \to \Box$. The first product seem to be a

subtype of the second but exclusively for λ abstractions. In some systems, where the β conversion is extended with the η rule, reflecting some form of extensionality, these two types can in fact be seen as subtypes but only modulo η conversion. For instance we have $f : A \to * \vdash_{\mathcal{C}} \lambda x.\, f\ x : A \to \Box$ yet $f : A \to * \not\vdash f : A \to \Box$ even though f and $\lambda x.\, f\ x$ are η-equivalent terms. In order to fix these odd properties of subtyping, the so-called *cumulative type systems* were introduced by Barras [Bar99] and later extensively studied by Lasson [Las12]. They extend sort subtyping to product types *covariantly on their codomains.*

Definition 5.3.1 (Cumulative type systems)**.** *The syntax of* cumulative type systems, CTS, *is the same as that of* PTS *(Definition 5.1.1) with* subtyping judgment*:* $A \preceq_{\mathcal{C}} B$.

A CTS *specification is the same as a* $\text{PTS}^{\preceq}$ *specification.*

Assume a CTS *specification* $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$. CTS$(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$ *is the type system defined from the syntax and typing rules of* PTS$(\mathcal{S}, \mathcal{A}, \mathcal{R})$ *(see Figure 5.2) where the conversion rule* $\mathcal{P}_{\equiv}$ *is replaced with a subtyping rule:*

$$\frac{\Gamma \vdash_{\mathcal{C}} t : A \qquad \Gamma \vdash_{\mathcal{C}} B : s \qquad A \preceq_{\mathcal{C}} B}{\Gamma \vdash_{\mathcal{C}} t : B}\ \mathcal{P}_{\preceq_{\mathcal{C}}} \qquad \frac{}{A \preceq_{\mathcal{C}} A}\ \preceq_{\mathcal{C}} i \qquad \frac{(s, s') \in \mathcal{C}^*}{s \preceq_{\mathcal{C}} s'}\ \preceq_{\mathcal{C}} \mathcal{C}$$

$$\frac{A \equiv_\beta A' \qquad A' \preceq_{\mathcal{C}} B' \qquad B' \equiv_\beta B}{A \preceq_{\mathcal{C}} B}\ \preceq_{\mathcal{C}} \equiv \qquad \frac{B \preceq_{\mathcal{C}} B'}{\Pi x : A.\, B \preceq_{\mathcal{C}} \Pi x : A.\, B'}\ \preceq_{\mathcal{C}} \pi$$

The full set of typing rules can be found in Figure 9.5.

We use the same terminology as for PTS*. Besides, the type A is said to be a* subtype *of the type B if $A \preceq_{\mathcal{C}} B$. Just like conversion, this property is independent from context.*

This definition is slightly different, but equivalent to, the usual reflexive, symmetric, transitive, and covariantly on product codomains, extension of $\mathcal{C}$.

Lemma 5.3.2 ($\preceq_{\mathcal{C}}$ *characterization*)**.** *$A \preceq_{\mathcal{C}} B$ iff $A \equiv_\beta B$ or there exists $(s, s') \in \mathcal{C}^*$ such that $A \equiv_\beta \Pi x_1 : C_1 \ldots \Pi x_n : C_n.\, s$ and $B \equiv_\beta \Pi x_1 : C_1 \ldots \Pi x_n : C_n.\, s'$.*

Proof. By induction on the derivation of $A \preceq_{\mathcal{C}} B$. □

Corollary 5.3.2.1. *We have*

- *$\preceq_{\mathcal{C}}$ is reflexive and transitive and therefore a preorder on terms.*
- *$\preceq_{\mathcal{C}}$ is stable by conversion and substitution.*

Lemma 5.3.3. *All* CTS *enjoy several properties such as*

- *confluence of the functional reduction β*
- *weakening: if $\Gamma \subseteq \Gamma'$ then $\Gamma \vdash_{\mathcal{C}} t : A \Rightarrow \Gamma' \vdash_{\mathcal{C}} t : A$;*
- *inversion properties linking typable terms to the shape of their type;*
- *correctness of typing: if $\Gamma \vdash_{\mathcal{C}} t : A$, then A is a well-formed type;*
- *product compatibility and subject reduction.*

The functionality property of PTS is replaced in CTS with the local minimum property and the uniqueness of type property with the existence of *principal types.*

Definition 5.3.4 (Principal type)**.** *We write* $\Gamma \vDash_{\mathcal{C}} t \Rightarrow A$ *if* $\Gamma \vdash_{\mathcal{C}} t : A$ *and for all* B *such that* $\Gamma \vdash_{\mathcal{C}} t : B$ *we have* $A \preceq_{\mathcal{C}} B$*.*

Definition 5.3.5. *A specification* $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$ *has the* local minimum property *if*

$$\left.\begin{array}{r} r_1 \preceq_{\mathcal{C}} s_1 \wedge (s_1, s_2) \in \mathcal{A} \\ r_1 \preceq_{\mathcal{C}} s_1' \wedge (s_1', s_2') \in \mathcal{A} \end{array}\right\} \implies \exists r_2, \quad (r_1, r_2) \in \mathcal{A} \wedge \left\{\begin{array}{l} r_2 \preceq_{\mathcal{C}} s_2 \\ r_2 \preceq_{\mathcal{C}} s_2' \end{array}\right.$$

and

$$\left.\begin{array}{r} r_1 \preceq_{\mathcal{C}} s_1 \wedge r_2 \preceq_{\mathcal{C}} s_2 \wedge (s_1, s_2, s_3) \in \mathcal{R} \\ r_1 \preceq_{\mathcal{C}} s_1' \wedge r_2 \preceq_{\mathcal{C}} s_2' \wedge (s_1', s_2', s_3') \in \mathcal{R} \end{array}\right\} \implies \exists r_3, (r_1, r_2, r_3) \in \mathcal{R} \wedge \left\{\begin{array}{l} r_3 \preceq_{\mathcal{C}} s_3 \\ r_3 \preceq_{\mathcal{C}} s_3' \end{array}\right.$$

Lemma 5.3.6 (Existence of principal type)**.** *Assume a specification satisfying the local minimum property and such that* $\mathcal{C}$ *is well-founded. Then* $\Gamma \vdash_{\mathcal{C}} t : A \Rightarrow \exists B, \Gamma \vDash_{\mathcal{C}} t \Rightarrow B$*.*

Definition 5.3.7 (CTS Morphism)**.** *Let* $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$ *and* $(\mathcal{S}', \mathcal{A}', \mathcal{R}', \mathcal{C}')$ *be two* CTS *specifications. A function* $\phi : \mathcal{S} \longrightarrow \mathcal{S}'$ *is a* CTS morphism *if and only if it preserves all relations:* $\phi(\mathcal{A}) \subseteq \mathcal{A}'$*,* $\phi(\mathcal{R}) \subseteq \mathcal{R}'$ *and* $\phi(\mathcal{C}) \subseteq \mathcal{C}'$*.*

In particular, if $\mathcal{S} \subseteq \mathcal{S}'$, $\mathcal{A} \subseteq \mathcal{A}'$, $\mathcal{R} \subseteq \mathcal{R}'$ and $\mathcal{C} \subseteq \mathcal{C}'$, then the identity define a morphism from $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$ to $(\mathcal{S}', \mathcal{A}', \mathcal{R}', \mathcal{C}')$ which is called a CTS extension.

Lemma 5.3.8. CTS *morphisms extend to terms and contexts by substituting all sort occurrences. They preserve* β*-equivalence, subtyping and typing.*

Corollary 5.3.8.1. *Assume there is a* CTS *morphism from* $\mathcal{P}$ *to a strongly (resp. weakly) normalizing* CTS*. Then* $\mathcal{P}$ *is also strongly (resp. weakly) normalizing.*

Corollary 5.3.8.2. CTS *extensions are conservative.*

For instance, $\mathcal{C}' := \mathcal{C}^*$ defines a convenient extension such that subtyping coincides with cumulativity on sorts. A further extension is the *cumulative closure* of a CTS, $\mathcal{A}' := \mathcal{A}\mathcal{C}^*$ and $\mathcal{R}' := \{(s_1, s_2, s_3) \mid (s_1', s_2', s_3') \in \mathcal{R} \wedge (s_1, s_1') \in \mathcal{C}^* \wedge (s_2, s_2') \in \mathcal{C}^* \wedge (s_3', s_3) \in \mathcal{C}^*\}$ which allows not to use subtyping on sorts and product types. Both extensions are shown to be actually equivalent to the original CTS.

A more extensive presentation can be found in [Las12] together with the omitted proof of the previous results.

5.4 Embedding CTS's in the lambda-Pi-calculus modulo

Having presented the already existing encoding of PTS and $\text{PTS}^{\preceq}$ in $\lambda\Pi_{\equiv}$, we study now the several challenges in extending these techniques to CTS's. We consider infinitely sorted systems and discuss the necessity to properly represent its universe structure. The simple sort-subtyping of $\text{PTS}^{\preceq}$ is also extended to product types in CTS, requiring explicit lifting of sorts to be extended to subtypes. We informally discuss here several encoding paradigms and techniques that have been studied to extend Assaf's encoding of the simple sort-subtyping of $\text{PTS}^{\preceq}$ to the subtyping in infinitely sorted CTS.

These techniques will be used and studied in the following chapters in the particular cases of several extensions of the Calculus of Constructions. This section aims at justifying the encoding design introduced later on through more accessible examples and illustrations. We take particular care in making the reasons behind our successive choices explicit. Getting the encoding of a complex type system just right is often tedious. Our hope is that parts of this section can be applied in similar settings.

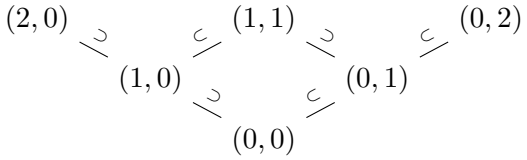
5.4.1 Infinite set of universes

The embedding of $\mathrm{PTS}^{\preceq}$ introduced in the previous section requires a dedicated type, $\vdash_{\mathcal{D}} \mathtt{U}_s : *$ for all sort s, a type decoder, $\vdash_{\mathcal{D}} \mathtt{T}_s : \mathtt{U}_s \to *$ for all sort s, a particular constructor for sorts, $\vdash_{\mathcal{D}} \mathbf{u}_{s_1} : \mathtt{U}_{s_2}$ if $(s_1, s_2) \in \mathcal{A}$, for products, $\pi_{s_1}^{s_2}$ if $(s_1, s_2, s_3) \in \mathcal{R}$, and the explicit lift symbol, $\vdash_{\mathcal{D}} \uparrow_{s_1}^{s_2} : \mathtt{U}_{s_1} \to \mathtt{U}_{s_2}$ if $(s_1, s_2) \in \mathcal{C}$. This no longer works with an infinite set of sorts as it would require an infinite signature.

First of all, encoding an infinitely sorted system requires to define a representation for its sorts using a finite signature of constructor symbols. We will assume from now on that sort representation are all typed with the same type, $\boldsymbol{\mathcal{S}}$. Correctly encoding sorts may sometimes require to use rewrite rules to properly reflect conversion properties between equivalent syntactical representations of sorts. In most cases the universe structure is simple enough so that this can be done using terminating first-order rewrite rules only.

Consider, for instance, the infinite double hierarchy of universes:

$$\mathrm{CTS}\left[\begin{array}{c} \mathcal{S} := \mathbb{N} \times \mathbb{N} \\ \mathcal{A}((i,j)) := (i+1, j+1) \\ \mathcal{R}((i,j),(k,l)) := (\max(i,k), \max(j,l)) \\ \mathcal{C}((i,j),(k,l)) \Leftrightarrow i \leq k \wedge j \leq l \end{array}\right]$$



Its set of sorts can be represented with the following convergent and well-typed signature.

$$\left\{\begin{array}{lcl} \boldsymbol{\mathcal{S}} & : & * \\ \mathtt{R} & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \\ \mathtt{L} & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \end{array}\right. \qquad \begin{array}{lcl} \mathtt{0} & : & \boldsymbol{\mathcal{S}} \\ & & \\ \mathtt{R}\ (\mathtt{L}\ X) & \longrightarrow & \mathtt{L}\ (\mathtt{R}\ X) \end{array}\left.\right\}$$

This encoding of sorts supports multiple syntactically different representations for a single sort such as $\mathtt{R}\ (\mathtt{R}\ (\mathtt{L}\ \mathtt{0}))$ and $\mathtt{L}\ (\mathtt{R}\ (\mathtt{R}\ \mathtt{0}))$ for $(1,2)$. However, in the empty context, all closed terms of type $\boldsymbol{\mathcal{S}}$ reduce to a unique normal form on the shape $\mathtt{L}^n\ (\mathtt{R}^k\ \mathtt{0})$ which represents (n,k). The rewrite rule collapses the representations, allowing to encode quotient sets.

When representing universe polymorphism, the set of sorts can be quantified over and the sort representation should support universe variables and be compatible with the substitution of these variables. Allowing universe variables usually makes it considerably more difficult to have a well-behaved representation even for the simplest structures. In the case of the simple linear hierarchy, it becomes necessary to consider sort expression such as $\max(i,j)$ representing the least upper bound of i and j and encoded with an extra `max` operator. The expressions $\max(x,y)$ and $\max(y,x)$ represent equal sorts so their

representations, $\mathtt{max}\ x\ y$ and $\mathtt{max}\ y\ x$, should reflect this and be convertible, hinting at associative-commutative properties for the $\mathtt{max}$ symbol. For similar reasons, $\mathtt{max}\ i\ i$ and i should be convertible, requiring a non-linear rule and $\mathtt{max}\ i\ (\mathtt{S}^n\ i)$ should be convertible with $\mathtt{S}^n\ i$ requiring an infinite set of rewrite rules.

Assuming a correct sort representation, the $\mathtt{U}$, $\mathtt{T}$, $\mathbf{u}$, $\boldsymbol{\pi}$ and $\uparrow$ operators should be finitely duplicated into parameterized versions. While this is straightforward for $\mathtt{U} : \boldsymbol{\mathcal{S}} \to *$ and $\mathtt{T} : \Pi s : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_s \to *$, the universe structure, defined by the $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{R}$ relations, needs to be reflected to only allow instances of the parameterized $\mathbf{u}$, $\boldsymbol{\pi}$ and $\uparrow$ that correspond to terms typable in the original system.

In the particular case where $\mathcal{A}$ and $\mathcal{R}$ are total relations, like in our example, a treatment similar to that of $\mathtt{U}$ and $\mathtt{T}$ can be applied to $\mathbf{u}$ and $\boldsymbol{\pi}$:

$$\left\{\begin{array}{lcl@{\qquad}lcl} \boldsymbol{\mathcal{A}} & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} & \mathbf{u}_\square & : & \Pi s : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_s \to \mathtt{U}_{(\boldsymbol{\mathcal{A}}\ s)} \\ \boldsymbol{\mathcal{R}} & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} & \boldsymbol{\pi}^\square_\square & : & \Pi s\ s' : \boldsymbol{\mathcal{S}}.\, \Pi a : \mathtt{U}_s.\, (\mathtt{T}_s\ a \to \mathtt{U}_{s'}) \to \mathtt{U}_{(\boldsymbol{\mathcal{R}}\ s\ s')} \end{array}\right\}$$

Note here that the $\mathbf{u}$ and $\boldsymbol{\pi}$ symbols are no longer duplicated. We however stick to an index notation for conciseness: for a term s, $\mathbf{u}_s\ :=\ (\mathbf{u}_\square\ s)\ :\ \mathtt{U}_s \to \mathtt{U}_{(\boldsymbol{\mathcal{A}}\ s)}$.

Naturally $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{R}}$ must be fully defined on closed terms, for instance with the rewrite rules $\boldsymbol{\mathcal{A}}\ X \longrightarrow \mathtt{L}\ (\mathtt{R}\ X)$ and the following rules defining $\boldsymbol{\mathcal{R}}$:

$$\left\{\begin{array}{rcl@{\qquad}rcl} \boldsymbol{\mathcal{R}}\ \mathtt{0}\ X & \longrightarrow & X & \boldsymbol{\mathcal{R}}\ (\mathtt{L}\ X)\ (\mathtt{R}\ Y) & \longrightarrow & \mathtt{L}\ (\boldsymbol{\mathcal{R}}\ X\ (\mathtt{R}\ Y)) \\ \boldsymbol{\mathcal{R}}\ X\ \mathtt{0} & \longrightarrow & X & \boldsymbol{\mathcal{R}}\ (\mathtt{R}\ X)\ (\mathtt{L}\ Y) & \longrightarrow & \mathtt{L}\ (\boldsymbol{\mathcal{R}}\ (\mathtt{R}\ X)\ Y) \\ \boldsymbol{\mathcal{R}}\ (\mathtt{R}\ X)\ (\mathtt{R}\ Y) & \longrightarrow & \mathtt{R}\ (\boldsymbol{\mathcal{R}}\ X\ Y) & & & \\ \boldsymbol{\mathcal{R}}\ (\mathtt{L}\ X)\ (\mathtt{L}\ Y) & \longrightarrow & \mathtt{L}\ (\boldsymbol{\mathcal{R}}\ X\ Y) & & & \end{array}\right\}$$

These rules are terminating, create no critical pairs and correctly compute the expected functions on closed terms of type $\boldsymbol{\mathcal{S}}$.

The case of $\uparrow$ is trickier since not all instances of $\uparrow : \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_{s_1} \to \mathtt{U}_{s_2}$ are legal in all CTS. A first solution is to rely on a family of pairs generating exclusively the allowed instances. For instance, we have $\mathcal{C} = \{(s, \max(s, s')) \mid s, s' \in \mathbb{N}\} = \{(s, s + s') \mid s, s' \in \mathbb{N}\}$ in the case of the single hierarchy so one of the following operator could work provided the $\mathtt{max}$ or $\mathtt{plus}$ symbols are defined.

$$\uparrow^\square_\square\ :\ \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_{s_1} \to \mathtt{U}_{\mathtt{max}(s_1, s_2)} \qquad\qquad \uparrow^\square_\square\ :\ \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_{s_1} \to \mathtt{U}_{\mathtt{plus}(s_1, s_2)}$$

With the first one we even have $\vdash_{\mathcal{D}} \uparrow^{s_2}_{s_1} : \mathtt{U}_{s_1} \to \mathtt{U}_{s_2}$ for closed terms s_1 and s_2 such that $s_1 \leq s_2$, which is quite convenient to use in the translation since the legality of an explicit lift is directly decided by the rewriting engine. The second requires to provide a level n such that $s_2 = s_1 + n$, acting as a witness that the origin sort is in fact a subtype of the target sort. This is a bit more cumbersome but spares the rewrite system some work. In both cases, however, to ensure the well-typedness of the necessary reflection rules of the encoding, these operators must not only correctly compute on closed terms, they should also have some convertibility properties on open terms. Consider for instance the case of the reflection of domain subtyping, $\boldsymbol{\pi}^{s_3}_s\ (\uparrow^{s_2}_{s_1} a)\ b \longrightarrow \uparrow^{\mathtt{max}\ s\ s_3}_{\mathtt{max}\ s_1\ s_3} (\boldsymbol{\pi}^{s_2}_{s_1}\ a\ b)$. The

right-hand side is only well-typed if $s \equiv_{\beta\mathcal{R}} \texttt{max}\ s_1\ s_2$ and for the rule to be type preserving we need $\texttt{max}\ s\ s_3 \equiv_{\beta\mathcal{R}} \texttt{max}\ (\texttt{max}\ s\ s_3)\ (\texttt{max}\ s_1\ s_3)$ which can be achieved, for instance, with distributivity rules such as $\texttt{max}\ (\texttt{max}\ s_1\ s_2)\ s_3 \longrightarrow \texttt{max}\ (\texttt{max}\ s_1\ s_3)\ (\texttt{max}\ s_2\ s_3)$ or by making max an associative and commutative symbol with the non-linear duplicate elimination rule: $\texttt{max}\ s\ s \longrightarrow s$. To properly represent levels as algebraic expressions in the $\{\texttt{max}, \texttt{S}\}$ or $\{\texttt{max}, +\}$ algebras by means of rewrite rules has already proven to be quite a challenge, even on closed expressions. When extending to universe polymorphism, universe variables are needed and it becomes quite hard to devise an efficient and correct rewrite system that decides when two algebraic expressions are equal and remains confluent on open expressions.

In Chapter 6 (see 6.1.2), we discuss a practical encoding of the particular case of the one-dimensional hierarchy of universes of the Calculus of Constructions.

5.4.2 From PTS$^{\preceq}$ embedding to CTS embedding

The main challenge when encoding CTS is that, just as in PTS$^{\preceq}$, subtyping breaks the uniqueness of type property which must however still hold for the image of the translation in $\lambda\Pi_{\equiv}$. In the case of PTS$^{\preceq}$ translation, Section 5.2, this was solved by considering a proven equivalent "à la Tarski" system, PTS$^{\uparrow}$, where subtyping is explicitly annotated in terms and conversion extended accordingly. This necessary extension of conversion comes from the fact that a term "à la Russell" can have multiple "à la Tarski" representations depending on its considered type and typing derivation.

Using minimal typing and the η rule

Assaf suggests in [Ass15b] to use the exact same encoding and to extend the translation function to represent product subtyping by η-expanding the translated term. He relies on two translation functions, a translation $[t]_\Gamma$ of a term t as an inhabitant of its (unique) *minimal type* and a translation $[t]_{\Gamma\vdash A}$ of t as an inhabitant of the type A.

$$
\begin{array}{rcll}
[t\ u]_\Gamma & := & [t]_\Gamma\ [u]_{\Gamma\vdash A} & \text{if } \Gamma \vDash_{\mathcal{C}} t \Rightarrow \Pi x : A.\, B \\
{[t]}_{\Gamma\vdash A} & := & \begin{cases} [t]_\Gamma & \text{if } \Gamma \vDash_{\mathcal{C}} t \Rightarrow A \\ \uparrow^{s}_{s'} [t]_\Gamma & \text{if } \Gamma \vDash_{\mathcal{C}} t \Rightarrow s' \preceq_{\mathcal{C}} s \\ \lambda x : [\![B]\!]_\Gamma.\, [t\ x]_{\Gamma, x:B\vdash C} & \text{if } A \equiv_\beta \Pi x : B.\, C \end{cases}
\end{array}
$$

Only types can be cast from a sort to a sort above. Terms with a product type need to be η-expanded. Assume $\mathcal{C}(s_1, s_2)$ and $\vdash_{\mathcal{C}} f : A \to s_1$ then we cannot directly translate it as $[f]$ since $[f] : [A] \to [s_1] \neq [A] \to [s_2]$. Instead we need to rely on: $\vdash_{\mathcal{D}} \lambda x : [A].\uparrow^{s_2}_{s_1} [f\ x] : [A] \to s_2$ which correspond to the translation of the η-expansion of f. This η-expansion must be done at translation time which is both expensive, since it requires to evaluate inferred types, and unsound if the translated system does not admit the η rule. Note that is also requires all terms to have a minimal type for the translation to be well-defined and this minimal type needs to be inferred at translation time.

With a general cast operator

An other possibility would be to rely on an encoding signature providing generalized *cast* operators explicitly lifting terms of type A to terms of type B every time $A \preceq_{\mathcal{C}} B$. This operator extends the lifting function on sorts $\uparrow_s^{s'}$ on any arbitrary well-formed types A and B, potentially containing computations, locally bounded variables that can be substituted or even other cast operations themselves. When the cumulativity relation is finite, it is possible to rely on a lifting symbol $\uparrow_s^{s'}$ for each $(s, s') \in \mathcal{C}$ as done in [Ass15b] and [CD07]. In the case of subtyping, however, we cannot simply provide a casting symbol for each legal subtyping since the pairs (A, B) such that $A \preceq_{\mathcal{C}} B$ are in infinite number. Instead the operator needs to be parameterized. Besides the `max` function on sorts needs to be extended to arbitrary types in a sort s: $\vee_s$.

$$\begin{array}{rcl} \vee_s & : & \mathtt{U}_s \to \mathtt{U}_s \to \mathtt{U}_s \\ \mathtt{cast}_s & : & \Pi A : \mathtt{U}_s.\ \Pi A' : \mathtt{U}_s.\ \mathtt{T}_s\ A \ \to \ \mathtt{T}_s\ (A \vee_s A') \\ A \vee_s B & \underset{\beta\mathcal{R}}{\twoheadleftarrow\!\!\!\twoheadrightarrow} & B \qquad\qquad (\text{if and only if } A \preceq_{\mathcal{C}} B) \end{array}$$

The last condition can be achieved using rewrite rules which would allow subtyping to be automatically checked using computation in $\lambda\Pi_{\equiv}$.

$$\begin{array}{rcl} A \vee_s A & \longrightarrow & A \\ \mathbf{u}_s \vee_r \mathbf{u}_{s'} & \longrightarrow & \mathbf{u}_{\max(s,s')} \\ (\boldsymbol{\pi}\ A\ B) \vee_s (\boldsymbol{\pi}\ A\ B') & \longrightarrow & \boldsymbol{\pi}\ A\ (B \vee_s B') \end{array}$$

Two of the required rewrite rules are however non-linear which usually breaks confluence. Instead of a definitional subtyping relation, we could use a judgmental relation:

$$\begin{array}{rrcl} & \mathtt{cast}_s & : & \Pi A : \mathtt{U}_s.\ \ \Pi B : \mathtt{U}_s.\ \ A \preceq_s B \to \mathtt{T}_s\ A \to \mathtt{T}_s\ B \\ & \preceq_s & : & \mathtt{U}_s \to \mathtt{U}_s \to \mathbb{B} \qquad (\text{for some new type } \mathbb{B} : *) \\ \exists p, & p & : & A \preceq_s B \qquad\qquad (\text{if and only if } A \preceq_{\mathcal{C}} B) \end{array}$$

The last condition can be achieved computationally with a single constructor $\mathbf{I} : \top$ and rewrite rules on the $\preceq_s$ symbol such that $(A \preceq_s B) \equiv \top$ if $A \preceq_{\mathcal{C}} B$. This would allow to rely on computation to check that the two provided levels correspond to a legal cast, similarly to the $\vee_s$ operator but with the same drawbacks. Instead of using rewriting, proving the legality of subtyping can also be done by means of extra proof constructors such as reflexivity $\mathtt{refl}_{\square} : \Pi s : \boldsymbol{\mathcal{S}}.\ \Pi A : \mathtt{U}_s.\ A \preceq_s A$ and a product codomain extension $\mathtt{prod}_{\square}^{\square}$ typed so that $\vdash_{\mathcal{D}} \mathtt{prod}_s^{s'}\ A\ B\ B' : (\Pi x : \mathtt{T}_s\ A.\ B\ x \preceq_{s'} B'\ x) \ \to \ (\boldsymbol{\pi}_s^{s'}\ A\ B \preceq_s \boldsymbol{\pi}_s^{s'}\ A\ B')$. Relying on provable predicates rather than computation to ensure subtyping raises the issue of non unicity of proof. There may be several inhabitants of a subtyping predicate $A \preceq_s B$ and since the `cast` operator relies on this proof, two occurrences $\mathtt{cast}_s\ A\ B\ p\ t$ and $\mathtt{cast}_s\ A\ B\ p'\ t$ may define nonconvertible terms even though they correspond to the same subtyping rule. For this reason, it is important that `cast` is defined so that its third argument is *irrelevant*. There must still exist a proof for the application to be well-typed but different choices of witness should yield convertible terms.

Translating derivations

Note that the "à la Tarski" system enjoys not only the uniqueness of type property but also the uniqueness of typing derivation property. In a sense well-typed "à la Tarski" terms are a representation of a particular *typing derivation* of the corresponding "à la Russell" term.

In presence of CTS's more general subtyping, however, the typing derivations rules are no longer syntax oriented because of the subtyping rules. It is therefore necessary to introduce *reflection* identities corresponding to conversion properties between the multiple possible typing derivations trees of a single well-typed term. When extending the $\text{PTS}^{\preceq}$ encoding, Assaf's translation [Ass15b] is defined on well-typed terms, yet heavily relies on their typing derivation. In order to be uniquely defined on terms, this translation needs to rely on a variation of CTS, called *minimal typing* or *bi-directional* CTS. Type checking in this systems requires two procedures, a type *inference* and a type *checking*. It was proven complete for several extensions of the Calculus of Constructions by Harper and Pollack [HP91] and for CTS with the *local minimum property* by Luo [Luo90]. Minimal typing allows to retrieve typing derivation uniqueness since the subtyping rule is exclusively and systematically used to *check* that the minimal type of the argument of an application is a subtype of the expected type.

Minimal typing makes one of the typing derivations canonical in some sense. However cut elimination, which corresponds to β-reduction in a shallow embedding, does not preserve the property to be a minimal typing derivation. Consider the two following judgments, both derivable in the two-sorts hierarchy $\{* \preceq_{\mathcal{C}} \Box\}$. The minimal derivation of the first one requires to subtype f and is therefore cut-eliminated to a non-minimal derivation in which subtyping is applied to the applicand rather than to the argument and overall application.

$$\dfrac{\dfrac{\dfrac{g : * \to \Box \quad t : *}{g\ t : \Box}}{\lambda g.\, g\ t : (* \to \Box) \to \Box} \qquad \dfrac{* \to * \preceq_{\mathcal{C}} * \to \Box}{f : * \to \Box}}{f : * \to *,\ t : * \vdash_{\mathcal{C}} (\lambda g : * \to \Box.\, g\ t)\ f : \Box} \xrightarrow[\beta]{} \dfrac{\dfrac{* \to * \preceq_{\mathcal{C}} * \to \Box}{f : * \to \Box} \qquad t : *}{f : * \to *,\ t : * \vdash_{\mathcal{C}} f\ t : \Box}$$

Because we still need to consider all typing derivation when translating terms, we chose in Chapter 8 to define our translation directly of typing derivation trees rather than on terms or judgments. Even though most well-behaved CTS systems do satisfy the *local minimum* property, this encoding paradigm allows to consider type systems where typing is not as straightforward. Besides, reasoning about the translation function is more direct and natural and does not require to know typing properties of the translated system which are no longer critical. Finally such a translation can be directly implemented as a side effect to an already existing typing algorithm, reusing more of the kernel's codebase instead of essentially re-implementing it.

As seen in Section 5.2 some terms, such as product types, can be typed multiple ways and therefore their translation is not uniquely defined. In order to reflect the conversion of the original system, the embedding must guarantee that all possible translations of a

type, at least in their "type" version, are convertible. In CTS this issue is exacerbated as subtyping no longer concerns exclusively types. Consider, for instance, a three-sorts hierarchy of universes, $\{* \preceq_{\mathcal{C}} \Box \preceq_{\mathcal{C}} \triangle\}$ the derivation $(\lambda x : \Box.\, x)\ * : \Box$ can be derived several ways depending on the position of the subtyping rule:

$$\dfrac{\dfrac{\cdots}{(\lambda x.\, x)\ * : \Box} \quad \dfrac{}{\Box \preceq_{\mathcal{C}} \triangle}}{(\lambda x.\, x)\ * : \triangle} \qquad \dfrac{\dfrac{\dfrac{\cdots}{x : \Box \vdash_{\mathcal{C}} x : \triangle}}{\lambda x.\, x : \Box \to \Box \to \triangle} \quad \dfrac{}{* : \Box}}{(\lambda x.\, x)\ * : \triangle} \qquad \dfrac{\dfrac{\dfrac{\cdots}{\lambda x.\, x : \Box \to \Box \preceq_{\mathcal{C}} \Box \to \triangle}}{\lambda x.\, x : \Box \to \triangle} \quad \dfrac{}{* : \Box}}{(\lambda x.\, x)\ * : \triangle}$$

These derivations correspond to three different well-typed translations of the same term $(\lambda x : \Box.\, x)\ *$ *seen as a term of type* $\triangle$:

$$\texttt{cast}\ \Box\ \triangle\ ((\lambda x : \Box.\, x)\ *) \qquad (\lambda x : \Box.\, \texttt{cast}\ \Box\ \triangle\ x)\ * \qquad (\texttt{cast}\ (\Box \to \Box)\ (\Box \to \triangle)\ \lambda x : \Box.x)\ *$$

While the first two are already β-convertible translations, the last one requires a new higher-order *reflection* rewrite rule:

$$\texttt{cast}\ (A \to B)\ (A \to C)\ \lambda x.\, F[x] \longrightarrow \lambda x : A.\, \texttt{cast}\ B\ C\ (F[x])$$

Similarly, both derivations of $f : * \to *\ ,\ t : * \vdash_{\mathcal{C}} f\ t : \Box$ require the following rewrite rule for their translations to be convertible:

$$\texttt{cast}\ (A \to B)\ (A \to C)\ F\ U \longrightarrow \texttt{cast}\ B\ C\ (F\ U)$$

5.5 Getting some privacy

The embedding techniques of CTS introduced in the previous section are already quite expressive and were used in practice to translate a large part of the `GeoCoq` library in Dedukti. Even though this library was written in Coq, it relies exclusively on its $\text{PTS}^{\preceq}$ structure and therefore the generated proofs are likely to be compatible with other developments in the same $\text{PTS}^{\preceq}$ fragment of CTS, independently from the system they were developed in.

These proofs are also fit to be incorporated in Logipedia and plugged into other tools allowing interoperability which is the main objective of this project

We provide, in this section, new encoding techniques to further extend the previous encodings. Our goal is to provide support for advanced features such as non-functional CTS systems or universe polymorphism, in a way that preserves the correctness and conservativity properties of the encoding. Overcoming these difficulties is necessary in order to encode Coq's most advanced features.

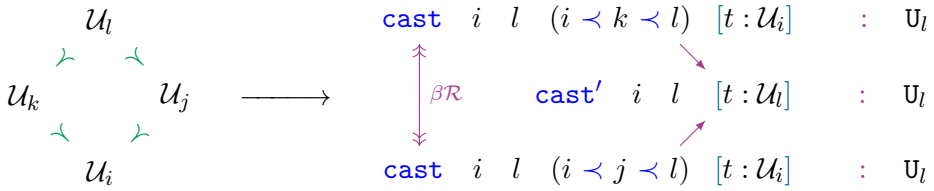
These techniques rely on *private* symbols in the signature which purpose is to extend the conversion between terms by providing "hidden" intermediate representations. The image of the translation will exclusively rely on the "public" signature which forbid these symbols. However convertibility between two public terms no longer requires them to

reduce to the same "public" canonical term. Instead they can both be computed a "private" common reduct which may not correspond to anything "public". We believe this technique is particularly useful to encode terms which conversion cannot be characterized by the existence of a canonical objects.

5.5.1 Private encodings for proof irrelevance

Using a judgmental subtyping relation allows to avoid non-linear rewrite rules. One of the main issue with this encoding is that the subtrees corresponding to subtyping judgments, $A \preceq_{\mathcal{C}} B$, in a typing derivation are not handled computationally with rewrite rules but translated as tree-encoding "proof" terms which can produce a sizable output. The other issue is that in order to fully reflect the conversion of the original system, the multiple ways to subtype a given term must all be translated to convertible $\mathtt{cast}$ expressions. For instance, if p and q are two inhabitants of the $A \preceq_s B$ type, then we must have $\mathtt{cast}_s\ A\ B\ p\ t \equiv_{\beta\mathcal{R}} \mathtt{cast}_s\ A\ B\ q\ t$. Put in different words, the fourth argument of $\mathtt{cast}$ must be *proof irrelevant*.

When translating floating universe levels with constraints, see Chapter 6, a convertibility issue arises when considering a diamond shaped local universe context ($i \prec_{\mathcal{C}} j \preceq_{\mathcal{C}} l$ and $i \prec_{\mathcal{C}} k \preceq_{\mathcal{C}} l$). In this case there are two different ways to inhabit the judgmental subtyping type $i \preceq l$: using transitivity of $\preceq$ either through j or through k. This can be a problem when the translation relies on both of these derivations in the translation of a subtyping derivation step, yielding two *non-convertible* translations of derivation trees of a same judgment compromising correctness.



As described in a joint work with Thiré [FcT19], one way to achieve proof-irrelevance of these universe constraints is to use a "private" version $\mathtt{cast}'$ of the $\mathtt{cast}$ operator. This new unsafe operator directly casts terms without requiring any witness of judgmental subtyping and therefore does not rely on the irrelevant argument. It is however necessary to control the usage of $\mathtt{cast}'$ to ensure that an expression $\mathtt{cast}'\ i\ l\ t$ is only used when $i \preceq l$ is inhabited. To this end, we split our encoding signature, $\Sigma_{\mathrm{CiC}} = \Sigma_{\mathrm{pub}} \uplus \Sigma_{\mathrm{pri}}$ into a *public* and a *private* signature:

- the public signature, Σ_{pub}, defines symbols freely available to the translation.
- the private signature, Σ_{pri}, defines symbols used only in the hidden representations of internal intermediate steps. They are generated by the rewrite engine to properly check convertibility while ensuring irrelevance when needed.

Restricting the image of the translation is critical to guarantee the conservativity of the encoding *in the public signature*. Terms built from private symbols may very well inhabit unexpected types but they are not considered.

As mentionned in [FcT19], an other way to achieve proof irrelevance is to ensure that all possible proofs of a judgmental relation are convertible. In confluent and terminating systems this means that they all reduce to the same normal form which is a canonical proof of the judgment. In the easiest cases, there is already only a single way to prove subtyping. For instance, if the only two proof constructors are $\texttt{refl}_i : i \preceq i$ and $\texttt{next}_{i,j} : i \preceq j \to i \preceq \texttt{S}\ j$ then the only closed inhabitant of the closed $i \preceq \texttt{S}^n\ i$ is the term $\texttt{next}^n\ (\texttt{refl}\ i)$. An other example is obtained if $\preceq$ is computationally defined and only provided a single constructor **I**. In both cases, proof irrelevance may not be needed as long as only closed terms are considered. In the general case and if we allow proof terms to contain variables, then we need a collapsing mechanism similar to our first proposition for proof irrelevance. We could decide that all constructor rewrite to an unsafe constructor, for instance using $\texttt{refl}_s\ A \longrightarrow \texttt{proof}_s\ A\ A$ and $\texttt{prod}_s^{s'}\ A\ B\ B'\ p \longrightarrow \texttt{proof}_{s'}\ (\boldsymbol{\pi}_s^{s'}\ A\ B)\ (\boldsymbol{\pi}_s^{s'}\ A\ B')$ which would guarantee that $\texttt{refl}_s\ (\boldsymbol{\pi}_s^s\ A\ B)\ (\boldsymbol{\pi}_s^s\ A\ B) \twoheadleftarrow\!\!\!\!\rightarrow \texttt{prod}_s^s\ A\ B\ B\ (\texttt{refl}_s\ B)$. This solution is however not sufficient since a single variable x can be a proof of $i \preceq j$, in particular if universes with local constraints are considered, see Chapter 6. In that case, it is still possible to achieve proof irrelevance by considering an extra version $\leq$ of the judgmental relation type which has two constructor: a public constructor relying on the previously defined type: $\texttt{cons}_s : \Pi A : \texttt{U}_s.\, \Pi B : \texttt{U}_s.\, A \preceq_s B \to A \leq_s B$ and a private collapsing symbol $\texttt{proof}_s : \Pi A : \texttt{U}_s.\, \Pi B : \texttt{U}_s.\, A \leq_s B$ together with the expected $\texttt{cons}_s\ A\ B\ p \longrightarrow \texttt{proof}_s\ A\ B$. Operators relying on subtype must now require an inhabitant of $A \leq_s B$ which can only be publicly constructed with cons and the usual constructors of $\preceq_s$ therefore keeping the associated guarantees. However this argument is now irrelevant since all occurrences of cons collapse to the same symbol.

This mechanism is quite general and could surely be adapted to properly encode other situations where proof irrelevance is required, such as PVS's predicate subtyping or Coq irrelevance of some sorts. Note that the private symbols are still part of the signature and must be defined with a type such that rewrite rules are well-typed.

5.5.2 Towards "codes"

We now describe and justify several incremental encoding adaptions leading up to the decision of having a private internal representation of terms.

0) The translation of subtyping $\uparrow_s^{s'}\ a : \texttt{U}_{(\texttt{max}\ s\ s')}$ highly relies on the hierarchical structure of sorts: not all cumulativity relations $\mathcal{C}$ define a join-semilattice. We have shown that it can be convenient and more general to consider instead an unsafe version of it, $\uparrow_s^{s'}\ a : \texttt{U}_{s'}$, together with a public version available to the translation but requiring an irrelevant guarantee that $s \preceq_{\mathcal{C}} s'$.

1) Similarly, it could be the case that the axiom, $\mathcal{A}$, and product, $\mathcal{R}$, relations do not satisfy the local minimum property, Definition 5.3.5. In that case, the $\mathbf{u}_s$ and $\boldsymbol{\pi}_s^{s'}$ operators cannot be given a type. Instead we need to replicate our handling of the lifting

operator and define

$$\begin{array}{rcl@{\qquad}rcl}
\boldsymbol{\mathcal{A}} & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to * & \boldsymbol{\mathcal{R}} & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to * \\
\underline{\mathbf{u}}^{s'}_{s} & : & \boldsymbol{\mathcal{A}}\ s\ s' \to \mathtt{U}_{s'} & \underline{\boldsymbol{\pi}}^{s_3}_{s_1,s_2} & : & \boldsymbol{\mathcal{R}}\ s_1\ s_2\ s_3 \to \Pi A : \mathtt{U}_{s_1}.\,(\mathtt{T}_{s_1}\ A \to \mathtt{U}_{s_2}) \to \mathtt{U}_{s_3} \\
\mathbf{u}^{s'}_{s} & : & \mathtt{U}_{s'} & \boldsymbol{\pi}^{s_3}_{s_1,s_2} & : & \Pi A : \mathtt{U}_{s_1}.\,(\mathtt{T}_{s_1}\ A \to \mathtt{U}_{s_2}) \to \mathtt{U}_{s_3} \\
\underline{\mathbf{u}}^{s'}_{s}\ p & \longrightarrow & \mathbf{u}^{s'}_{s} & \underline{\boldsymbol{\pi}}^{s_3}_{s_1,s_2}\ p\ A\ B & \longrightarrow & \boldsymbol{\pi}^{s_3}_{s_1,s_2}\ A\ B
\end{array}$$

The terms $\mathbf{u}^{s'}_{s}$ and $\boldsymbol{\pi}^{s'}_{s_1,s_2}\ A\ B$ are the private canonical representations of the sort $s \in \mathcal{S}$ and the product $\Pi x : A.\,B$, respectively, considered as –or rather assumed– inhabitants of the sort s'. Their underlined versions are public and require an extra irrelevant proof of the corresponding judgmental condition which must be implemented with constructors of, and rewrite rules on, the dependent types $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{R}}$.

2) Subtyping may interact with the type constructors and create several versions of the same type and forcing the following rules into the private signature to collapse them.

$$\begin{array}{rcl}
\uparrow^{s''}_{s'}\ \mathbf{u}^{s'}_{s} & \longrightarrow & \mathbf{u}^{s''}_{s} \\
\uparrow^{s'}_{s_3}\ (\boldsymbol{\pi}^{s_3}_{s_1,s_2}\ A\ B) & \longrightarrow & \boldsymbol{\pi}^{s'}_{s_1,s_2}\ A\ B \\
\boldsymbol{\pi}^{s_3}_{s_1,s_2}\ (\uparrow^{s_1}_{s'} A)\ \lambda x.\,B[x] & \longrightarrow & \boldsymbol{\pi}^{s_3}_{s',s_2}\ A\ \lambda x.\,B[x] \\
\boldsymbol{\pi}^{s_3}_{s_1,s_2}\ A\ \lambda x.\uparrow^{s_2}_{s'} B[x] & \longrightarrow & \boldsymbol{\pi}^{s_3}_{s_1,s'}\ A\ \lambda x.\,B[x]
\end{array}$$

3) The lift operator is eliminated in its argument in the first two rules and in the above $\boldsymbol{\pi}$ in the last two, therefore yielding four non-joinable critical pairs, such as:

$$\boldsymbol{\pi}^{s_3}_{s',s_2}\ \mathbf{u}^{s'}_{s}\ B \longleftarrow \boldsymbol{\pi}^{s_3}_{s_1,s_2}\ (\uparrow^{s_1}_{s'} \mathbf{u}^{s'}_{s})\ B \longrightarrow \boldsymbol{\pi}^{s_3}_{s_1,s_2}\ \mathbf{u}^{s_1}_{s}\ B$$

The left-hand side of any of these critical pairs correspond to a representation of the same term as the right-hand side but typed differently. In order to ensure reflection of the original system's conversion, we need these representations to be convertible. One way to achieve this is to make the sort argument of $\boldsymbol{\pi}$ and the second sort argument of $\mathbf{u}$ irrelevant. To do that we define a special private sort $\mathbf{s}_\bot$ such that products and sorts have a canonical representation in sort $\mathbf{s}_\bot$: $|\Pi x : A.\,B| := \boldsymbol{\pi}^{\mathbf{s}_\bot}_{\mathbf{s}_\bot,\mathbf{s}_\bot}\ |A|\ (\lambda x.\,|B|)$, $|\mathcal{U}_s| := \mathbf{u}^{\mathbf{s}_\bot}_{s}$. The canonical representation of a type t in an actual sort s is then simply $\uparrow^{s}_{\mathbf{s}_\bot}\ |t|$.

4) It is convenient to introduce more meaningful notations for term constructions relying on the $\mathbf{s}_\bot$ symbol. For instance $\mathtt{Univ}\ s_\bot$ is the type of canonical but unsafe term representations, we call it the type of *codes* and write it $\mathbb{C}$. The other short names are:

$$\begin{array}{rcl@{\quad}rcl}
\mathtt{Univ}\ s_\bot & \longmapsto & \mathbb{C} : * & \uparrow^{s_\bot}_{s} & \longmapsto & \mathtt{c} : \Pi s : \boldsymbol{\mathcal{S}}.\,\mathtt{U}_s \to \mathbb{C} \\
\mathtt{Term}\ s_\bot & \longmapsto & \mathtt{D} : \mathbb{C} \to * & \uparrow^{s}_{s_\bot} & \longmapsto & \mathtt{u} : \Pi s : \boldsymbol{\mathcal{S}}.\,\mathbb{C} \to \mathtt{U}_s \\
\mathbf{u}^{s_\bot}_{s} & \longmapsto & \mathbf{u}_\Box : \boldsymbol{\mathcal{S}} \to \mathbb{C} & \boldsymbol{\pi}^{s_\bot}_{s_\bot,s_\bot} & \longmapsto & \boldsymbol{\pi} : \Pi A : \mathbb{C}.\,(\mathtt{D}\ A \to \mathbb{C}) \to \mathbb{C}
\end{array}$$

and we ensure canonical representations by defining the public symbols using them, for instance $\underline{\mathbf{u}}^{s'}_{s}\ p \longrightarrow \mathtt{u}(s', \mathbf{u}_s)$ and $\underline{\boldsymbol{\pi}}^{s_3}_{s_1,s_2}\ p\ A\ \lambda x.\,B[x] \longrightarrow \mathtt{u}(s_3, \boldsymbol{\pi}\ \mathtt{c}(s_1, A)\ \lambda x.\,\mathtt{c}(s_2, B[x]))$. Finally we need a collapsing rule $\mathtt{c}(s, \mathtt{u}(s, A)) \longrightarrow A$ to ensure the canonical form is reached.

5) Extending sort lifting to `cast` is simply done by adapting types $\mathtt{c} : \Pi c : \mathbb{C}.\,\mathtt{D}\ c \to \mathbb{C}$ and $\mathtt{u} : \Pi c : \mathbb{C}.\,\mathbb{C} \to \mathtt{D}\ c$.

6) The two necessary reflection rules introduced in 5.4.2, expressed in this setting, generate two unjoinable critical pairs with the collapsing rule:

$$\begin{array}{ccc} & \mathtt{c}(\boldsymbol{\pi}\ A\ B', \mathtt{u}(\boldsymbol{\pi}\ A\ B', \mathtt{c}(\boldsymbol{\pi}\ A\ B, \lambda x : \mathtt{D}\ A.\, T[x]))) & \\ \mathtt{c}(\boldsymbol{\pi}\ A\ B, \lambda x : \mathtt{D}\ A.\, T[x]) \xleftarrow{\Lambda} & & \xrightarrow{2} \mathtt{c}(\boldsymbol{\pi}\ A\ B', \lambda x : \mathtt{D}\ A.\, \mathtt{u}(B'[x], \mathtt{c}(B[x], T[x]))) \end{array}$$

and

$$\begin{array}{ccc} & \mathtt{u}(\boldsymbol{\pi}\ A\ B', \mathtt{c}(\boldsymbol{\pi}\ A\ B, \mathtt{u}(\boldsymbol{\pi}\ A\ B, T)))\ U & \\ \mathtt{u}(B'[a], \mathtt{c}(B[a], \mathtt{u}(\boldsymbol{\pi}\ A\ B, T)\ U))\ U) \xleftarrow{\Lambda} & & \xrightarrow{12} \mathtt{u}(\boldsymbol{\pi}\ A\ B', T)\ U \end{array}$$

A way to understand them is that while types are guaranteed a single canonical representation, λ-abstractions and applications are not. In PTS$^{\preceq}$, only sort inhabitants could have multiple types (which are all sorts). In CTS however more objects may be subtyped, including abstractions and applications.

Both pairs can be collapsed by providing two extra private "code representations", $\boldsymbol{cL} : (\mathbb{C} \to \mathbb{C}) \to \mathbb{C}$ and $\boldsymbol{cA} : \mathbb{C} \to \mathbb{C} \to \mathbb{C}$. For the complete set of rules the Figure 8.3 from Chapter 8.

5.6 A new paradigm

5.6.1 Terms, types and trees

Previous work on type system embedding focus on translating *terms* and *types* well-typed in the original system to $\lambda\Pi_{\equiv}$. These translations and their study however often heavily rely on the corresponding *typing derivations* to provide the guarantees required to prove that they are correctly defined and well-behaved. In Cousineau and Dowek's paradigm, see Figure 5.1, and Assaf's paradigm, see Figure 5.3, a term T, respectively a judgment $\vdash_{\mathcal{P}} T : A$, may have two representations:

- a "type" version, $[\![T]\!]_{\Sigma;\Gamma} : *$ available whenever the term T is a type;
- a "well-typed" version, $[T] : [\![U]\!]_{\Sigma;\Gamma}$ available whenever the term T is of type U.

The decoding operator $\mathtt{T}$ is defined with term rewriting to link both representations: $\mathtt{T}\ [T] \equiv_{\beta\mathcal{R}} [\![T]\!]_{\Sigma;\Gamma}$.

This paradigm needs to be adapted in order to handle CTS. Indeed, the main difficulty is that CTS's typing rules are not *syntax oriented*. A solution to retrieve this property is to explicitely annotate subtyping in terms. There are then as many representations of a term t *"as typed"* A as there are derivations of the judgement $t : A$, including, for instance, the subtrees giving a sort to A.

To define a unique representation for terms one may constrain the typing system and restrict the considered universe structure. We argue however that, in order to represent more complex features such as an infinite set of sorts, CTS subtyping without η, non-functionality and even inductive types (see Chapter 9), it is more natural and convenient to translate *derivations trees* rather than terms. The several issues described in Section 5.4 and Section 5.5 and their proposed solutions led to define a new paradigm, see Figure 5.4, around the following key concepts:

- *Typing derivations* valid in the original system are translated into well-typed $\lambda\Pi_{\equiv}$ terms using a public signature of symbols. Therefore terms do not have a unique representation and in particular use of the subtyping rule are explicitly annotated.
- The public encoding extends Cousineau and Dowek's and provides "well-typed" and "type" representations of terms. In particular the translation of terms well-typed without subtyping is the same as their translation as terms of PTS.
- The conversion of well-typed terms in the original systems is reflected, at least at the type level, in the $\lambda\Pi_{\equiv}$ representation *regardless of the chosen typing derivations*. This reflection is obtained by means of well-typed rewrite rules, some of which may refer to a set of private symbols unavailable to the translator.
- The embedding is *shallow* and the conversion of well-typed terms in the original systems should be reflected at the type level in the encoding *regardless of the chosen typing derivations*. In particular the type representation of a product type is a product type and the $\mathcal{P}_@$ and $\mathcal{P}_\lambda$ rules are translated as an application and an abstraction respectively.

5.6.2 Private codes

In practice we define three translation functions: the translation of a term t as a "code", $|t|_{\Sigma;\Gamma} : \mathbb{C}$, the translation of a type A as a type, $[\![A]\!]_{\Sigma;\Gamma} : *$ and the translation of a judgment derivation $\left[\frac{\dots}{t:A}\right] : [\![A]\!]_{\Sigma;\Gamma}$ which relies exclusively on the safe public signature. The "decoding" operator, $\mathtt{D} : \mathbb{C} \to *$, from codes to types is such that $\mathtt{D}\ |A|_{\Sigma;\Gamma} \equiv_{\beta\mathcal{R}} [\![A]\!]_{\Sigma;\Gamma}$. The code of a term t may also be "uncoded" to the canonical representation of t as an inhabitant of its type B: $\mathtt{u}(|B|_{\Sigma;\Gamma}, |t|_{\Sigma;\Gamma}) : [\![B]\!]_{\Sigma;\Gamma}$. Typing derivations of a term t can be "coded" into their unique code representation

$$\mathtt{c}(|A|_{\Sigma;\Gamma}, \left[\frac{\pi}{t:A}\right]) \quad \underset{\beta\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow} \quad |t|_{\Sigma;\Gamma} \quad \underset{\beta\mathcal{R}}{\longleftarrow\!\!\!\!\!\leftarrow} \quad \mathtt{c}(|B|_{\Sigma;\Gamma}, \left[\frac{\pi'}{t:B}\right])$$

The code constructors $\mathbf{u}$, $\boldsymbol{\pi}$, $\boldsymbol{cA}$ and $\boldsymbol{cL}$ as well as the code operators $\mathtt{D}$, $\mathtt{c}$ and $\mathtt{u}$ are too permissive to be left available directly to the translation and must therefore all be private. In fact, adding the $\mathbb{C}$ symbol to the set of private symbols completely hides codes away as they are never mentioned in the public interface.

Note that the role of the internal code representation of terms and derivations is double.

- Since codes are a private intermediate representation, they are created from the public signature using the coding operator $\mathtt{c}(A, \pi_t)$ with requires a type code $\vdash_{\mathcal{D}} A : \mathbb{C}$ and a derivation π_t that a certain term t has type A. The underlying guarantee is that the only codes considered correspond to well-typed terms since they had to be "annotated" with them.
- At the same time, the code representation of t is meant to reduce to a canonical form representing only the syntax of the term t, erasing all typing details. Although we did not investigate in detail the connection between the two, this process is quite similar to *normalisation by evaluation* [Abe13], the untyped terms of the λ-calculus being the denotational semantics of typing derivations.

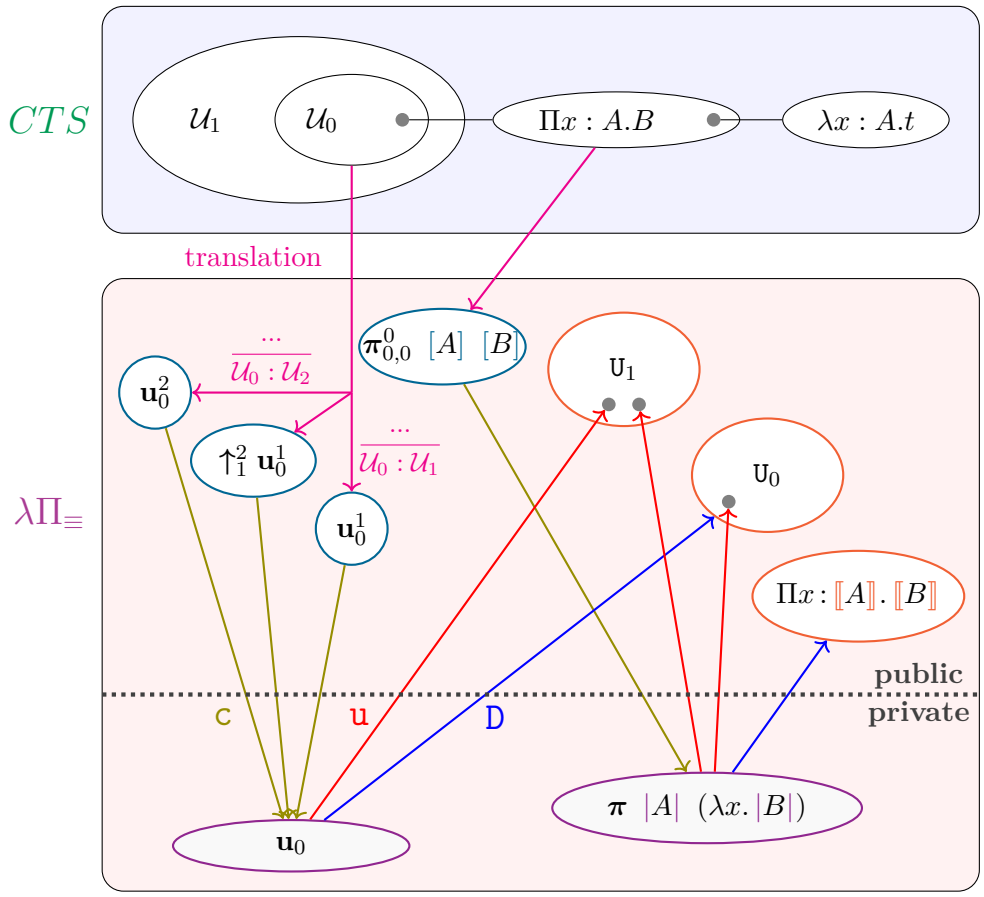


Figure 5.4: The new paradigm

Chapter 6

Calculi of Constructions with Universe Variables

In this chapter we describe how the several encoding techniques introduced in the previous chapter can be applied to the particular case of the Calculus of Constructions. We introduce successive extensions of the original Calculus of Constructions, CoC, from the literature and provide an encoding of their universe structure:

- the generalized, CC_ω, and extended, ECC, Calculus of Constructions both introducing an infinite hierarchy of universes;
- the $\mathrm{ACC}^{\sim}_{\mathcal{L}}$ introducing level variables and algebraic level expressions;
- and the $\mathrm{CCC}^{\sim}_{\mathcal{L}}$ introducing level constraints.

These systems all have principal types which can be inferred with bi-directional typing, therefore Assaf's encoding can be adapted to handle them. We do not need code representations of terms or typing derivation translation yet. In order to correctly embed them, it is however necessary that the set of their sorts and its structure are correctly represented which is the main focus of this chapter. We investigate in particular how algebraic sort expressions containing free *universe variables* can be encoded in a way compatible with level equality, inequality and instantiation. This will allow their embedding in $\lambda\Pi_{\equiv}$ in a way compatible with universe polymorphism which is our endgame objective and studied in Chapters 7 and 8. For each system we provide a partial signature encoding the set of its sorts using typed symbols and rewrite rules as well as a translation function. The sort structure is embedded with either functional operators when the sort representation allows it or with irrelevant inhabitable predicate arguments.

6.1 The infinite universe hierarchy

The original two-universes *Calculus of Constructions*, CoC, was introduced by Coquand in his thesis [Coq85] and studied with Huet [CH85, CH86]. It extends Church's simply typed lambda-calculus with all dimensions of Barendregt λ-cube: dependent types, polymorphic types and type constructors. Its expressivity means it is particularly suited to be both a

typed programming language and a constructive foundation of mathematics following the Curry-Howard isomorphism.

6.1.1 Definition

It can be seen as as particular Pure Type System with a sort of *small types* or *propositions* `Prop` and a sort of *large types* `Type`:

$$\text{CoC} := \text{PTS}\left[\begin{array}{l}\mathcal{S} := \{\ \mathtt{Prop}, \mathtt{Type}\ \} \\ \mathcal{A} := \{\ (\mathtt{Prop}, \mathtt{Type})\ \}\end{array}\quad \mathcal{R} := \left\{\begin{array}{l}(\mathtt{Prop}, \mathtt{Prop}, \mathtt{Prop}), (\mathtt{Type}, \mathtt{Prop}, \mathtt{Prop}), \\ (\mathtt{Prop}, \mathtt{Type}, \mathtt{Type}), (\mathtt{Type}, \mathtt{Type}, \mathtt{Type})\end{array}\right\}\right]$$

The sort `Prop` of propositions is said to be *impredicative* as there are well-typed propositions quantifying over the type of propositions themselves: $\vdash_{\mathcal{P}} \Pi P : \mathtt{Prop}.\, P : \mathtt{Prop}$.

It was already known since Martin Löf's intuitionistic type theory [ML75, ML84], that this setting can easily be extended to allow an infinite set of `Type` sorts, each allowing to reason about the one below: $\mathtt{Type}_0 \in \mathtt{Type}_1 \in \ldots$.

$$\text{PTS}\left[\begin{array}{ll}\mathcal{S} & := \{\mathtt{Prop}\} \cup \{\mathtt{Type}_n \mid n \in \mathbb{N}\} \\ \mathcal{A} & := \{(\mathtt{Prop}, \mathtt{Type}_0)\} \ \cup\ \{(\mathtt{Type}_n, \mathtt{Type}_{n+1}) \mid n \in \mathbb{N}\} \\ \mathcal{R} & := \left\{\begin{array}{ll}(\mathtt{Prop}, \mathtt{Type}_n, \mathtt{Type}_n) & \mid n \in \mathbb{N}\} \\ (\mathtt{Type}_n, \mathtt{Type}_m, \mathtt{Type}_{\max(n,m)}) & \mid n, m \in \mathbb{N}\} \\ (s, \mathtt{Prop}, \mathtt{Prop}) & \mid s \in \mathcal{S}\}\end{array}\right\}\end{array}\right] \qquad \begin{array}{c}\vdots \\ \mathtt{Type}_1 \\ \scriptstyle\cup\,| \\ \mathtt{Type}_0 \\ \scriptstyle\cup\,| \\ \mathtt{Prop}\end{array}$$

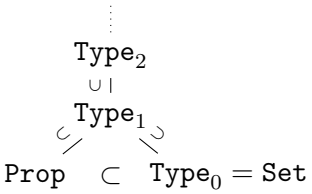
Coquand later extended CoC with an infinite hierarchy of universes [Coq86], defining the *Generalized Calculus of Constructions*: CC_ω. This system still features the impredicative sort of propositions `Prop` at the bottom of an infinite stratified hierarchy of types $\mathtt{Type}_n$ indexed with a *level* in $\mathbb{N}$. Lower sorts are typed with and are subsets of upper sorts introducing therefore a safe form of subtyping through sort cumulativity.

$$\text{CC}_\omega := \text{PTS}^{\preceq}\left[\begin{array}{ll}\mathcal{S} & := \{\mathtt{Prop}\} \cup \{\mathtt{Type}_n \mid n \in \mathbb{N}\} \\ \mathcal{A} & := \{(\mathtt{Prop}, \mathtt{Type}_0)\} \ \cup\ \{(\mathtt{Type}_n, \mathtt{Type}_{n+1}) \mid n \in \mathbb{N}\} \\ \mathcal{R} & := \left\{\begin{array}{ll}(\mathtt{Prop}, \mathtt{Type}_n, \mathtt{Type}_n) & \mid n \in \mathbb{N}\} \\ (\mathtt{Type}_n, \mathtt{Type}_m, \mathtt{Type}_{\max(n,m)}) & \mid n, m \in \mathbb{N}\} \\ (s, \mathtt{Prop}, \mathtt{Prop}) & \mid s \in \mathcal{S}\}\end{array}\right\} \\ \mathcal{C} & := \mathcal{A}\end{array}\right] \qquad \begin{array}{c}\vdots \\ \mathtt{Type}_1 \\ \scriptstyle\cup\,| \\ \mathtt{Type}_0 \\ \scriptstyle\cup\,| \\ \mathtt{Prop}\end{array}$$

This system was again extended by Luo [Luo90, Luo89] to allow covariant subtyping on codomain of products defining the *Extended Calculus of Constructions*, ECC. He also put the lowest level of large types, $\mathtt{Type}_0$ also called `Set`, at the same level as `Prop`, a predicative counterpart to represent constructions that do not correspond to propositions such as natural numbers, Boolean and so on. This system was thoroughly studied by Luo with and without Σ-types which we chose to remove from our presentation. It can be

expressed as the CTS with the following specification

$$\text{ECC} := \text{CTS}\left[\begin{array}{l} \mathcal{S} \text{ and } \mathcal{R} \text{ same as in } \text{CC}_\omega \\ \mathcal{A} := \{(\texttt{Prop}, \texttt{Type}_1)\} \\ \qquad \cup \{(\texttt{Type}_n, \texttt{Type}_{n+1}) \mid n \in \mathbb{N}\} \\ \mathcal{C} := \left\{\begin{array}{ll} (\texttt{Prop}, \texttt{Type}_n) & \mid n \in \mathbb{N}, \\ (\texttt{Type}_n, \texttt{Type}_m) & \mid n \leq m \end{array}\right\} \end{array}\right]$$



CoC, CC_ω and ECC all satisfy the properties of *subject reduction*, *strongly normalization* and *decidability of type-checking*. Besides CC_ω and ECC are *full* in the sense that all sorts are well-typed (with a sort) and all products $\Pi x : A.\, B$ are well-typed (with a sort) if both A and B are well-typed (with a sort). It also satisfies the property of *principal typing*: any well-typed term t can be inferred a unique (up to conversion) principal type T such that T is a subtype of all types of t. This property allows to have a sound and complete syntax oriented set of typing rules for CC_ω where the subtyping rule is used exclusively on subterms in argument positions.

6.1.2 Embedding in the lambda-Pi-calculus modulo

As explained in Section 5.4.1, in order to embed this CTS in $\lambda\Pi_{\equiv}$, we need to provide a representation for sorts as terms of a specific type $\boldsymbol{\mathcal{S}}$. This can simply be done here by defining $[n] := \texttt{S}^n\ \texttt{0}$, $[\texttt{Prop}] := \texttt{Prop}$ and $[\texttt{Type}_n] := \texttt{Type}_{[n]}$ in the following signature

$$\begin{array}{lll} \mathbb{N} : * & \boldsymbol{\mathcal{S}} : * & \texttt{max} : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \quad (\texttt{max} \in \mathcal{F}_{\text{AC}}) \\ \texttt{0} : \mathbb{N} & \texttt{Prop} : \boldsymbol{\mathcal{S}} & \texttt{max}\ \texttt{0}\ N \longrightarrow N \\ \texttt{S} : \mathbb{N} \to \mathbb{N} & \texttt{Type}_\square : \mathbb{N} \to \boldsymbol{\mathcal{S}} & \texttt{max}\ (\texttt{S}\ M)\ (\texttt{S}\ N) \longrightarrow \texttt{S}\ (\texttt{max}\ M\ N) \end{array}$$

This system is confluent and strongly normalizing. The normal form of any closed term of type $\mathbb{N}$ is of the shape $\texttt{S}^k\texttt{0}$ which can directly be back-interpreted as the integer k.

Assaf designed in [Ass15b], an encoding where the usual $\texttt{T}$, $\texttt{U}$, $\mathbf{u}$, $\boldsymbol{\pi}$ and $\uparrow$ operators of the finitely sorted CTS embeddings were parameterized with arguments of type $\boldsymbol{\mathcal{S}}$ rather than (finitely) duplicated for all their possible instances. This finite signature, without the reflection rules, can be found in Figure 9.7. The universe structure is represented with functional symbols $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{R}}$ for the $\mathcal{A}$ and $\mathcal{R}$ relations assumed functional, and $\boldsymbol{\mathcal{C}}$ such that for all $(s, s') \in \mathcal{C}$ we have $\boldsymbol{\mathcal{C}}(s, s') \equiv_{\beta\mathcal{R}} s'$.

Several problems arise when trying to represent the *reflection* equivalence relation (as defined in Section 5.2) with rewrite rules.

- In order for the reflection rules on products to be well-typed, we need rewrite rules to ensure some properties of the functional symbols $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{R}}$ and $\boldsymbol{\mathcal{C}}$ encoding the universe structure. For instance, we need to have $\boldsymbol{\mathcal{R}}(\boldsymbol{\mathcal{C}}(s_1, s_2), s_3) \equiv_{\beta\mathcal{R}} \boldsymbol{\mathcal{C}}(\boldsymbol{\mathcal{R}}(s_1, s_3), \boldsymbol{\mathcal{R}}(s_2, s_3))$ and at the same time $\boldsymbol{\mathcal{R}}(s_1, \boldsymbol{\mathcal{C}}(s_2, s_3)) \equiv_{\beta\mathcal{R}} \boldsymbol{\mathcal{C}}(\boldsymbol{\mathcal{R}}(s_1, s_2), \boldsymbol{\mathcal{R}}(s_1, s_3))$ creating a non-closing critical pair if oriented in the natural way, as pointed out in [Ass15b]. In [ADJL16], a system is provided that fixes this confluence issue by using two associative commutative symbols, $\texttt{max}, + \in \mathcal{F}_{\text{AC}}$.

- The elimination of identity lifts $\uparrow^N_N T \equiv T$ requires a non-linear rewrite rule. The non-linearity of this rule is a problem when proving confluence of the system. It can however be dealt with by syntactically confining integer representations in a confined level containing closed sort expressions exclusively below the representation of terms, allowing to use Theorem 4.4.9.

- The non-linear rule creates other non-closing critical pairs such as

$$\begin{array}{rcccl} \mathtt{T}_s\ A & \longleftarrow & \mathtt{T}_s\ \left(\uparrow^{s'}_{s'} A\right) & \longrightarrow & \mathtt{T}_{s'}\ A \\ \uparrow^{\mathcal{C}(s_2,s_3)}_{s_3} A & \longleftarrow & \uparrow^{s_2}_{s_1} \left(\uparrow^{s_3}_{s_3} A\right) & \longrightarrow & \uparrow^{s_2}_{s_1} A \end{array}$$

  Since in both cases, each side contains a variable missing from the other, they cannot be closed by rewriting one side to the other, *à la* Knuth-Bendix. Changing the conflicting linear rule into a non-linear version, $\mathtt{T}_{\boldsymbol{\mathcal{C}}(S,S')}\ \left(\uparrow^{S'}_{S} A\right) \longrightarrow \mathtt{T}_S\ A$, makes the rule impossible to use on closed terms. For instance $\mathtt{T}_{\mathtt{Type}_0}\ \left(\uparrow^{\mathtt{Type}_0 A}_{\mathtt{Prop}}\right)$ would be a normal form since $\mathtt{Type}_0$ does not match $\boldsymbol{\mathcal{C}}(S, S')$.

To overcome these issues, a solution is to consider a predicate $\boldsymbol{\mathcal{C}}(s_1, s_2)$ that is defined so as to be inhabited if only if $(s_1, s_2) \in \mathcal{C}$:

$$\begin{array}{rcl@{\qquad}rcl} \top & : & * & \boldsymbol{\mathcal{C}}(\mathtt{Prop}, N) & \longrightarrow & \top \\ \mathbf{I} & : & \top & \boldsymbol{\mathcal{C}}(\mathtt{Type}_0, \mathtt{Type}_N) & \longrightarrow & \top \\ \boldsymbol{\mathcal{C}}(\Box,\Box) & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to * & \boldsymbol{\mathcal{C}}(\mathtt{Type}_{(\mathtt{S}\,M)}, \mathtt{Type}_{(\mathtt{S}\,N)}) & \longrightarrow & \boldsymbol{\mathcal{C}}(\mathtt{Type}_M, \mathtt{Type}_N) \end{array}$$

This allows to define a public $\underline{\uparrow}$ operator together with an unsafe private counter part $\uparrow$:

$$\begin{array}{rcl} \underline{\uparrow}^{\Box}_{\Box}\ \Box\ \Box & : & \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \boldsymbol{\mathcal{C}}(s_1, s_2) \to \mathtt{U}_{s_1} \to \mathtt{U}_{s_1} \\ \uparrow^{\Box}_{\Box}\ \Box & : & \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_{s_1} \to \mathtt{U}_{s_1} \\ \underline{\uparrow}^{S'}_{S}\ P\ A & \longrightarrow & \uparrow^{S'}_{S} A \end{array}$$

All reflection rules can now be stated on the private version $\uparrow$ which is much better behaved with the rest of the encoding. The full encoding, $\mathcal{D}[\mathrm{CC}_\omega]$, of CC_ωin $\lambda\Pi_{\equiv}$, including the reflection rules, can be found in Figure 9.6. Note that the non-linearity from identity lift elimination must be propagated to many other rules to avoid critical pairs. However this non-linearity is kept on meta-variables of type $\boldsymbol{\mathcal{S}}$ in object rewriting rules. This means that sorts can still be syntactically put at a confined level to prove confluence together with β.

Using a non-linear rewrite system together with private symbols therefore allows to represent the infinite hierarchies of universes of Coq. Although it is usable in practice, the system previously introduced by Assaf [Ass15b] relied on a rewrite system for universe expressions which had non-joinable critical pairs and therefore was not confluent. While $\mathcal{D}[\mathrm{CC}_\omega] \cup \beta$ is not confluent either because of its non-linearities, results from Chapter 4 allow to prove the following key properties.

Lemma 6.1.1. $\underset{\mathcal{D}[\mathrm{CC}_\omega]}{\longrightarrow}$ *is confluent and strongly normalizing.*
$\underset{\mathcal{D}[\mathrm{CC}_\omega]}{\longrightarrow} \cup \underset{\beta}{\longrightarrow}$ *is locally confluent.*
$\mathcal{D}[\mathrm{CC}_\omega]$ *is confluent with* β *on terms such that sorts and natural number representations are syntactically enforced to be at a confined level.*

Proof. Termination of $\underset{\mathcal{D}[\mathrm{CC}_\omega]}{\longrightarrow}$ is done by building a non-erasing polynomial interpretation over $\mathbb{N}^+$. All rules but the last two are size decreasing therefore they decrease the interpretation w that maps all symbols but $\boldsymbol{\pi}$ to the constant 1 (to be precise we need $\mathrm{w}(\boldsymbol{\mathcal{A}}) := 2$) and such that $\mathrm{w}(u\ v) := \mathrm{w}(u)+\mathrm{w}(v)$ and $\mathrm{w}(\lambda x : A.\, t) = \mathrm{w}(\lambda x : A.\, t) := \mathrm{w}(A)+\mathrm{w}(t)$. It only remains to define a polynomial P such that $\mathrm{w}(\boldsymbol{\pi}^N_M\ A\ B) := P(\mathrm{w}(M), \mathrm{w}(N), \mathrm{w}(A), \mathrm{w}(B))$. In order for the last two rules to be decreasing, that polynomial should satisfy both $P(K, N, K+M+A+1, X+B) > M+2N+K+3+P(M, N, A, X+B)$ and $P(M, K, A, X+N+K+B+1) > 2M+N+K+3+P(M, N, A, X+B)$. It is then easy to see that $P(M, N, A, B) := (M+N+A+B)^n$ works for some n big enough.

Local confluence is done by checking that all critical pairs are joinable.

The confinement is such that $*, \Box, \mathbb{N}, \boldsymbol{\mathcal{S}}, \mathbf{I}, \top \in \mathcal{L}_1$ and $\mathtt{0}, \mathtt{S}, \mathtt{max}, \mathtt{Prop}, \mathtt{Type}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}} \in \mathcal{L}_0$ while $\mathtt{U}, \mathtt{T}, \mathbf{u}$, resp. $\boldsymbol{\mathcal{C}}, \boldsymbol{\pi}, \uparrow, \underline{\uparrow}$, build a term of $\mathcal{L}_1$ from one confined argument in $\mathcal{L}_0$, resp. from two confined arguments, and are therefore of arity 1, resp. 2. These symbols are therefore allowed to be non-linearly rewritten provided the non-linearity of their rules is at confined positions which we check is the case. Finally the strong normalization of rewriting at level $\mathcal{L}_1$ is compatible with the interpretation erasing subterms at level $\mathcal{L}_0$ and therefore Theorem 4.4.9 applies. □

This syntactical constraint restricts the set of terms considered while still guaranteeing that the image of the translation is syntactically allowed. The representation of sorts is defined to be confined terms while the representation of other terms is in $\mathcal{L}_1$. To be more precise, the constructors for the $\boldsymbol{\mathcal{S}}$ and $\mathbb{N}$ types can only be applied to terms built from these same constructors. They form a closed algebra of terms allowing λ abstractions but constraining them to be only applied to terms in the algebra of confined terms. It is easy to check that all the rewrite rules are compatible with this stratification.

Lemma 6.1.2. $\mathcal{D}[\mathrm{CC}_\omega]$ *is well-typed.*
CC_ω *with explicit subtyping can be directly translated in this encoding.*

Assaf showed that explicit subtyping is complete for CC_ω. However embedding ECC requires to go further and extend subtyping to products. This will be done in Chapter 8 using the more general casting techniques describe in Chapter 5. In the rest of this chapter we rather focus on ways to embed infinite structured sets of universes.

6.1.3 Towards universe polymorphism

In an infinite hierarchy of universe, some constructions may have to be placed at a certain level even though they do not necessarily need to be. For instance polymorphic lists can be defined identically at level $\mathtt{Type}_0$ or at level $\mathtt{Type}_1$. If both versions are mentioned

in a development, it is therefore necessary to define both $\texttt{list}_0 : \texttt{Type}_0 \to \texttt{Type}_0$ and $\texttt{list}_1 : \texttt{Type}_1 \to \texttt{Type}_1$. Cumulativity is already helpful as it allows to use $\texttt{list}_1$ even on types in $\texttt{Type}_0$ but the resulting type would have to be a type in $\texttt{Type}_1$, therefore putting all subsequent development one level higher than strictly necessary.

Russell and Whitehead introduced in *Principia Mathematica* an elegant way to deal in practice with this situation called *typical ambiguity*. Following this convention, universe levels could be omitted in proofs and terms and they are inferred while checking their well-typedness. It is checked that there exists concrete levels from the hierarchy that allow the terms to be well-typed. For instance the judgment $\vdash_{\mathcal{S}} \Pi f : \texttt{Type} \to \texttt{Type}.\ f\ \texttt{Type} : \texttt{Type}$ is derivable even though it seems that it would require the unsafe $\texttt{Type} : \texttt{Type}$. In fact it can be *elaborated* at type-checking into a well-typed judgment, for instance with the following universe levels assignment: $\vdash_{\mathcal{S}} \Pi f : \texttt{Type}_1 \to \texttt{Type}_1.\ f\ \texttt{Type}_0 : \texttt{Type}_2$. As a matter of fact, the actual levels assigned to sort occurrences is not as important as the relationship between them. In our example, we actually have $\vdash_{\mathcal{S}} \Pi f : \texttt{Type}_i \to \texttt{Type}_j.\ f\ \texttt{Type}_k : \texttt{Type}_l$ for all i, j, k and l such that $k < i$ and $\max(i, j) < l$. This paves the way to the definition of a Calculus of Constructions where universes can be algebraic expressions that may contain universe variables implicitly quantifying over all of their valid instances.

6.2 Algebraic universes

We define in this section the universe algebraic Calculus of Constructions, $\text{ACC}^{\sim}_{\mathcal{L}}$. We show that even if its set of sorts features level variables and arbitrary algebraic expressions, it remains well-behaved as a CTS and could therefore be embedded in $\lambda\Pi_{\equiv}$.

6.2.1 Definition

Definition 6.2.1 (Level Expressions)**.** *Assume a set of universe variables $\mathcal{L}$. The set $\mathcal{E}_{\mathcal{L}}$ of (algebraic)* level expressions *is inductively built from*

- *closed levels, $n \in \mathbb{N}$;*
- *$u + n$ for level expressions u and $n \in \mathbb{N}$;*
- *level variables, $i \in \mathcal{L}$;*
- *$\max(u, v)$ for level expressions u and v.*

We define the set of algebraic sorts as $\mathcal{S}_{\mathcal{L}} := \{\texttt{Prop}\} \cup \{\texttt{Type}_u \mid u \in \mathcal{E}_{\mathcal{L}}\}$ and the set of algebraic terms $\mathcal{T}_{\mathcal{L}}$ as the CTS terms built from this set of sorts. The set of level variables *in a level expression u is written $\mathcal{LV}ar(u)$ and is naturally extended to sorts and terms.*

We sometimes use the n-ary version of max, $\max(u, v, w) := \max(u, \max(v, w))$.

As defined in Harper and Pollack [HP91, HP89], we consider assignments of level variables to natural numbers.

Definition 6.2.2 (Level Assignments)**.** *A* level assignment*, or* valuation*, σ is a function from $\mathcal{L}$ to $\mathbb{N}$ (or sometimes to $\mathbb{Z}$ in which case it will be made explicit). Level assignments naturally extend to non-decreasing functions on algebraic level expressions of $\mathcal{E}_{\mathcal{L}}$ by interpreting $+$ and* max *respectively as the addition and maximum functions on $\mathbb{N}$. They extend to terms and contexts as well by substituting all of their level expressions, producing therefore terms and contexts of the usual Generalized Calculi of Constructions, CC_{ω} and ECC.*

We write $\vDash u = v$ *(resp.* $\vDash u \leq v$*) if for all assignment (over* $\mathbb{N}$*)* σ*,* $\sigma(u) = \sigma(v)$ *(resp.* $\sigma(u) \leq \sigma(v)$*). We define* level conversion *as the equivalence relation* $\equiv_{\mathcal{L}}$ *on levels (naturally extended to sorts and terms) such that* $u \equiv_{\mathcal{L}} v \iff \vDash u = v$*.*

We write σ_0 *the constant assignment mapping all variables to 0.*

For instance we have $\vDash i + 0 = i$, $\vDash 1 + 1 = 2$, $\vDash (i + 1) + 2 = i + 3$ and $\vDash \max(\max(j + 2, i), \max(i + 3, j)) = \max(j + 2, i + 3)$.

All CTS judgments containing universe variables should be derivable if and only if all of their instances are derivable in ECC. Free universe variable can therefore be seen as universally quantified, implicitly. They could be substituted any concrete level in $\mathbb{N}$ or other level expression. An assignment would then define a morphism from the system with universe variables to ECC, thus preserving its properties. We first prove a couple of utility lemmas which will be useful to characterize level expressions.

Definition 6.2.3. *A level expression* u *is guarded, written* $\boldsymbol{G}(u)$ *if* $\forall \sigma \in \mathbb{Z}^{\mathcal{L}}, \sigma(u) \geq \sigma_0(u)$*.*

For instance $\max(0, i)$, $\max(1 + 2, (u + 1) + 1)$ and $\max(2, i, j + 1)$ are guarded, their nullary valuations are the lower bound of their integer valuations, while $\max(i, j)$, $i + 2$ and $\max(1, i + 2)$ are not.

Lemma 6.2.4. *For all* u*, there exists* v *such that* $\boldsymbol{G}(v)$ *and* $\vDash u = v$*.*

Proof. Replacing all occurrences of variables i in u with $\max(0, i)$ yields v. □

Lemma 6.2.5. *If* $\mathcal{LV}ar(u) = \{i_1, \ldots, i_n\}$*, then* $\vDash u = \max(k_0, i_1 + k_1, \ldots, i_n + k_n)$ *for some* $(k_i)_i$*. This representation is unique if we force the guard condition:* $\forall i, k_0 \geq k_i$*.*

Proof. By induction on u, since $\vDash \max(u_1, \ldots, u_n) + k = \max(u_1 + k, \ldots, u_n + k)$ using the simple following identities on natural numbers:

- $\vDash (u + n) + k = u + (n + k)$
- $\vDash (n) + k = (n + k)$
- $\vDash \max(i + k, i + n, u) = \max(i + \max(k, n), u)$
- $\vDash i = i + 0$

and the associativity and commutativity property of max.

Assuming $\vDash \max(k_0, i_1 + k_1, \ldots, i_n + k_n) = \max(k'_0, i_1 + k'_1, \ldots, i_n + k'_n)$ then if $k_j \neq k'_j$ for some $j \geq 1$, then we can choose a large enough valuation for i_j in σ, and we get $\sigma(\max(k_0, i_1 + k_1, \ldots, i_n + k_n)) = \sigma(i_j) + k_j \neq \sigma(i_j) + k'_j = \max(k'_0, i_1 + k'_1, \ldots, i_n + k'_n)$. If $k_0 \neq k'_0$, we chose $\sigma : i \mapsto 0$ and since $k_0 \geq \max_j(k_j)$ and $k_0 \geq \max_j(k'_j)$, we have $\sigma(\max(k_0, i_1 + k_1, \ldots, i_n + k_n)) = k_0 \neq k'_0 = \max(k'_0, i_1 + k'_1, \ldots, i_n + k'_n)$.

The condition provided corresponds to the guard condition since if $k_0 < k_i$ then the max expression can be $\mathbb{Z}$-evaluated to k_0 while its σ_0 valuation is greater than k_i. To be more precise, $\mathbf{G}(\max(k_0, i_1 + k_1, \ldots, i_n + k_n)) \Leftrightarrow k_0 = \sigma_0(u) \geq \max((k_j)_j)$. □

Corollary 6.2.5.1. *For all level* u *and assignment* σ*,* $\sigma(u) \geq \max_{i \in \mathcal{LV}ar(u)}(\sigma(i))$*.*

Corollary 6.2.5.2. *If* $\vDash u \leq v$ *then* $\mathcal{LV}ar(u) \subseteq \mathcal{LV}ar(v)$*.*

Proof. Assume $i \in \mathcal{LV}ar(u) \setminus \mathcal{LV}ar(v)$, the assignment family $\sigma_k := \{j \mapsto k\delta_{ij}\}$ is such that $(\sigma_k(v))_{k \in \mathbb{N}}$ is constant while $\sigma_k(u) \geq k$ by Corollary 6.2.5.1. Therefore there exists an index n such that $\sigma_n(u) > \sigma_n(v)$ contradicting $\vDash u \leq v$. □

Definition 6.2.6. *Assume a set of universe variables $\mathcal{L}$. The Calculus of Constructions with algebraic universes can be defined as the* CTS *with the following specification*

$$\mathrm{ACC}_{\mathcal{L}} := \left[\begin{array}{ll} \mathcal{S} & := \mathcal{S}_{\mathcal{L}} \\ \mathcal{A} & := \{\ (\mathtt{Prop}, \mathtt{Type}_u), (\mathtt{Type}_u, \mathtt{Type}_{u+1})\ \} \\ \mathcal{R} & := \left\{\begin{array}{l} (\mathtt{Prop}, \mathtt{Prop}, \mathtt{Prop}), (\mathtt{Type}_u, \mathtt{Prop}, \mathtt{Prop}), \\ (\mathtt{Prop}, \mathtt{Type}_u, \mathtt{Type}_u), (\mathtt{Type}_u, \mathtt{Type}_v, \mathtt{Type}_{\max(u,v)}) \end{array}\right\} \\ \mathcal{C} & := \{(\mathtt{Prop}, \mathtt{Type}_u)\} \cup \{(\mathtt{Type}_u, \mathtt{Type}_v) \mid \vDash u \leq v\} \end{array}\right]$$

where the conversion rule is extended with $\equiv_{\mathcal{L}}$.

Lemma 6.2.7. *$\equiv_{\mathcal{L}}$ and $\underset{\beta}{\longrightarrow}$ sub-commute: if $u \equiv_{\mathcal{L}} t \underset{\beta}{\longrightarrow} v$, then $u \underset{\beta}{\longrightarrow} t' \equiv_{\mathcal{L}} v$.*

Proof. Note that $\equiv_{\mathcal{L}}$ is not a rewrite rule. We consider the parallel sets P of sort positions in t. We have, $t = \underline{t}_P \overline{t}^P$, $\overline{t}^P \equiv_{\mathcal{L}} \sigma$ and $u = \underline{t}_P \sigma$. Since sorts are neither abstractions nor applications, necessarily, if $t \overset{p}{\underset{\beta}{\longrightarrow}} v$ then $p \cdot F_\beta \not>_{\mathcal{P}} P$. By Corollary 3.1.11.1, $\underline{t}_P \underset{\beta}{\longrightarrow} u'$ such that $u \underset{\beta}{\longrightarrow} u'\sigma = t' \equiv_{\mathcal{L}} u' \overline{t}^P = v$. □

Lemma 6.2.8. *$\underset{\beta\mathcal{L}}{\longrightarrow} := (\underset{\beta}{\longrightarrow} \cup \equiv_{\mathcal{L}})$ is Church-Rosser and $\underset{\beta\mathcal{L}}{\twoheadleftarrow\!\!\twoheadrightarrow} = (\underset{\beta}{\longrightarrow\!\!\!\!\twoheadrightarrow} \equiv_{\mathcal{L}} \underset{\beta}{\twoheadleftarrow})$.*

Proof. By induction and sub-commutation, $\underset{\beta\mathcal{L}}{\longrightarrow\!\!\!\!\twoheadrightarrow} = (\underset{\beta}{\longrightarrow\!\!\!\!\twoheadrightarrow} \equiv_{\mathcal{L}})$. By confluence of β and sub-commutation again, we prove $\underset{\beta\mathcal{L}}{\twoheadleftarrow} \underset{\beta\mathcal{L}}{\longrightarrow\!\!\!\!\twoheadrightarrow} \subseteq \equiv_{\mathcal{L}} \underset{\beta}{\longrightarrow\!\!\!\!\twoheadrightarrow} \underset{\beta}{\twoheadleftarrow} \equiv_{\mathcal{L}} \subseteq \underset{\beta}{\longrightarrow\!\!\!\!\twoheadrightarrow} \equiv_{\mathcal{L}} \underset{\beta}{\twoheadleftarrow}$. The last part is done by induction on the number of peaks in $\underset{\beta\mathcal{L}}{\twoheadleftarrow\!\!\twoheadrightarrow}$ using sub-commutation. □

Because of this commutation property, this system may instead be seen as a particular CTS where algebraic sorts are considered modulo $\equiv_{\mathcal{L}}$:

Definition 6.2.9 (Algebraic Level CoC)**.** *Assume a set of universe variables $\mathcal{L}$. We define the* universe algebraic Calculus of Constructions *as the following* CTS

$$\mathrm{ACC}^{\sim}_{\mathcal{L}} := \left[\begin{array}{ll} \mathcal{S} & := \mathcal{S}_{\mathcal{L}} / \equiv_{\mathcal{L}} \\ \mathcal{A} & := \{\ (\mathtt{Prop}, \mathtt{Type}_w) \mid \vDash w = 1\ \} \\ & \quad \cup \{\ (\mathtt{Type}_u, \mathtt{Type}_w) \mid \vDash w = u + 1\ \} \\ \mathcal{R} & := \{\ (\mathtt{Prop}, \mathtt{Prop}, \mathtt{Prop})\ \} \\ & \quad \cup \{\ (\mathtt{Type}_u, \mathtt{Prop}, \mathtt{Prop})\ \} \\ & \quad \cup \{\ (\mathtt{Prop}, \mathtt{Type}_u, \mathtt{Type}_w) \mid \vDash w = u\ \} \\ & \quad \cup \{\ (\mathtt{Type}_u, \mathtt{Type}_v, \mathtt{Type}_w) \mid \vDash w = \max(u, v)\ \} \\ \mathcal{C} & := \{\ (\mathtt{Prop}, \mathtt{Type}_w)\ \} \cup \{\ (\mathtt{Type}_u, \mathtt{Type}_w) \mid \vDash u \leq w\ \} \end{array}\right]$$

Proof. The $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{C}$ relation are well defined on classes of equivalence of $\mathcal{S}$. For instance, if $(s_1, s_2) \in \mathcal{A}$, $s_1 \equiv_{\mathcal{L}} s_1'$ and $s_2 \equiv_{\mathcal{L}} s_2'$, then $(s_1', s_2') \in \mathcal{A}$. □

Lemma 6.2.10. *The $\mathcal{A}$ and $\mathcal{R}$ relations of $\mathrm{ACC}^{\sim}_{\mathcal{L}}$ are total functions.*

Proof. Functionality and totality are both easily verified by definition for both $\mathcal{A}$ and $\mathcal{R}$: if $(s, s_1) \in \mathcal{A}$ and $(s, s_2) \in \mathcal{A}$ then $\vDash s_1 = s_2$ by case disjunction on s. □

Lemma 6.2.11. *An assignment σ defines a morphism from $\mathrm{ACC}^{\sim}_{\mathcal{L}}$ to CC_{ω}.*

Proof. σ defines a function from sorts of $\mathrm{ACC}^{\sim}_{\mathcal{L}}$ to sorts of CC_{ω}. All representatives of an equivalence class in $\mathcal{S}_{\mathcal{L}}$ are mapped to the same concrete universe in CC_{ω}. It is easily verified that this function satisfies all properties defining a morphism.

For example, if $(\mathtt{Type}_u, \mathtt{Type}_w) \in \mathcal{A}_{\mathrm{ACC}^{\sim}_{\mathcal{L}}}$, then by definition $\vDash w = u + 1$, therefore $\sigma(w) = \sigma(u) + 1$ and $(\mathtt{Type}_{\sigma(u)}, \mathtt{Type}_{\sigma(w)}) = (\mathtt{Type}_{\sigma(u)}, \mathtt{Type}_{\sigma(u)+1}) \in \mathcal{A}_{\mathrm{CC}_{\omega}}$. □

Corollary 6.2.11.1. *$\mathrm{ACC}^{\sim}_{\mathcal{L}}$ is strongly normalizing.*

Lemma 6.2.12. *The strict part of the $\mathcal{C}$ relation of $\mathrm{ACC}^{\sim}_{\mathcal{L}}$ is well-founded.*

Proof. Assume $(s_i)_{i\in\mathbb{N}}$ such that for all i, $(s_{i+1}, s_i) \in \mathcal{C}$ and $\nvDash s_i = s_{i+1}$. By definition of $\mathcal{C}$, since there is no sort s such that $(s, \mathtt{Prop}) \in \mathcal{C}$, we know that for all i, $s_i = \mathtt{Type}_{u_i}$ for levels u_i such that $\vDash u_{i+1} \leq u_i$ (1) and $\nvDash u_{i+1} = u_i$ (2). By Corollary 6.2.5.2 and induction, $\mathcal{LV}ar(u_i) \subseteq \mathcal{LV}ar(u_0) = \{j_1, \ldots, j_n\}$. By Lemma 6.2.5, $\vDash u_i = \max(k_0, j_1 + k^i_1, \ldots, j_n + k^i_n)$ where some of the $j_m + k^i_m$ may be omitted. For all $m \in \mathbb{N}$, $(k^i_m)_i$ must be decreasing to satisfy (1) and for all i, it must be the case that $k^{i+1}_m < k^i_m$ for some m to satisfy (2). This means that $\left(\sum_m k^i_m\right)_m$ is a strictly decreasing sequence of natural number, we conclude. □

Lemma 6.2.13. *$\mathrm{ACC}^{\sim}_{\mathcal{L}}$ has the local minimum property.*

Proof. By total functionality, there is always a unique r_2 such that $(r_1, r_2) \in \mathcal{A}$ (resp. r_3 such that $(r_1, r_2, r_3) \in \mathcal{R}$). Assuming $(s_1, s_2) \in \mathcal{A}$, $(s'_1, s'_2) \in \mathcal{A}$, $(r_1, s_1) \in \mathcal{C}$, $(r_1, s'_1) \in \mathcal{C}$ and $(r_1, r_2) \in \mathcal{A}$, then $(r_2, s_2) \in \mathcal{C}$, $(r_2, s'_2) \in \mathcal{C}$. If $r_1 = \mathtt{Prop}$, then $r_2 = \mathtt{Type}_1$ and s_2, s'_2 are both $\mathtt{Type}_u$ for some u such that $\vDash 1 \leq u$, we conclude. Otherwise $r_1 = \mathtt{Type}_u$, $r_2 = \mathtt{Type}_{u+1}$ and necessarily $s_1 = \mathtt{Type}_v$, for some v such that $\vDash u \leq v$. We conclude since $s_2 = \mathtt{Type}_{v+1}$ and $\vDash u + 1 \leq v + 1$ therefore $(r_2, s_2) \in \mathcal{A}$. The same goes for s'_1 and s'_2. The condition on $\mathcal{R}$ is checked similarly. □

Note that this property would still hold when considering for example the completion of the CTS $\mathrm{ACC}^{\sim}_{\mathcal{L}}$ where $\mathcal{A}$ and $\mathcal{R}$ would no longer be functional. We will see later that when considering a constrained universe structure it is easier to consider this completion.

Theorem 6.2.14. *Bidirectional typing is complete for $\mathrm{ACC}^{\sim}_{\mathcal{L}}$.*

Proof. Luo showed [Luo90] that it is complete for any CTS satisfying the local minimum property (Lemma 6.2.13) and such that the strict part of $\mathcal{C}$ is well-founded (Lemma 6.2.12). □

6.2.2 Embedding in the lambda-Pi-calculus modulo

The minimal typing property allows to use Assaf's encoding to simply encode this particular CTS, provided we are able to correctly encode and translate sorts. A quotient set is simply encoded in Dedukti with an encoding of its elements' representatives together with a rewrite system reflecting the equivalence relation between them as the term convertibility of their representations.

Definition 6.2.15. *We define the following encoding and translation of level expressions to terms of $\lambda\Pi_{\equiv}$:*

$$\begin{array}{rl} \mathbb{N}: & * \\ \mathtt{0}: & \mathbb{N} \\ \mathtt{S}: & \mathbb{N} \to \mathbb{N} \\ +: & \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\ \mathtt{max}: & \mathbb{N} \to \mathbb{N} \to \mathbb{N} \end{array} \qquad \begin{array}{rl} [i] := & \mathtt{max}\ \mathtt{0}\ i \\ [n] := & \mathtt{S}^n\ \mathtt{0} \\ [u+n] := & +\ [u]\ [n] \\ [\max(u,v)] := & \mathtt{max}\ [u]\ [v] \end{array}$$

In the context of translation the Agda system to Dedukti, Genestier designed a terminating rewrite system to decide level conversion [Gen20a]. This system requires an associative and commutative symbol `max` and assumes that the second argument of the $+$ symbol is always a closed term.

Lemma 6.2.16. *Assume two level expressions u and v of $\mathrm{ACC}_{\mathcal{L}}$, then $\vDash u = v$ iff $[u] \underset{\mathcal{R}_l}{\twoheadleftarrow\!\!\twoheadrightarrow} [v]$ with $\mathcal{R}_l$ the following rewrite system:*

$$\begin{array}{rcl} \mathtt{max} & \in & \mathcal{F}_{\mathrm{AC}} \\ +\ u\ \mathtt{0} & \longrightarrow & u \\ +\ \mathtt{0}\ u & \longrightarrow & u \\ +\ (\mathtt{S}\ n)\ k & \longrightarrow & +\ n\ (\mathtt{S}\ k) \\ +\ (+\ u\ n)\ k & \longrightarrow & +\ u\ (+\ n\ k) \\ +\ (\mathtt{max}\ u\ v)\ k & \longrightarrow & \mathtt{max}\ (+\ u\ k)\ (+\ v\ k) \end{array} \qquad \begin{array}{rcl} \mathtt{max}\ \mathtt{0}\ \mathtt{0} & \longrightarrow & \mathtt{0} \\ \mathtt{max}\ (\mathtt{S}\ n)\ \mathtt{0} & \longrightarrow & \mathtt{S}\ n \\ \mathtt{max}\ (\mathtt{S}\ n)\ (\mathtt{S}\ k) & \longrightarrow & \mathtt{S}\ (\mathtt{max}\ n\ k) \\ \mathtt{max}\ u\ u & \longrightarrow & u \\ \mathtt{max}\ (+\ u\ n)\ (+\ u\ k) & \longrightarrow & +\ u\ (\mathtt{max}\ n\ k) \\ \mathtt{max}\ u\ (+\ u\ k) & \longrightarrow & +\ u\ k \end{array}$$

This system is not confluent however it is terminating (modulo AC) and has the uniqueness (modulo AC) of normal forms property on the images of the translation.

This system simply reflects all the identities used in the proof of Lemma 6.2.5. For instance $\vDash (u+n)+k = u+(n+k)$ is reflected with the fourth rewrite rule:

$$[(u+n)+k] := +\ (+\ [u]\ [n])\ [k] \longrightarrow +\ [u]\ (+\ [n]\ [k]) =: [u+(n+k)]$$

Proof. This system is non-linear which forbids its confluence together with β on all terms. Besides this system is not even locally confluent since it has several non joinable critical pairs such as, for instance:

$$\begin{array}{rcccl} \mathtt{S}\ n & \longleftarrow & +\ (\mathtt{S}\ n)\ \mathtt{0} & \longrightarrow & +\ n\ (\mathtt{S}\ \mathtt{0}) \\ +\ (\mathtt{S}\ u)\ (+\ n\ k) & \longleftarrow & +\ (+\ (\mathtt{S}\ u)\ n)\ k & \longrightarrow & +\ (+\ u\ (\mathtt{S}\ n))\ k \end{array}$$

It is terminating since in can be checked that the following interpretation function w from terms to the well founded $(\mathbb{N}^*, \leq)$ is compatible with AC, $t \equiv_{\mathrm{AC}} u \Rightarrow \mathrm{w}(t) = \mathrm{w}(u)$, and strictly decreasing along reduction:

$$\begin{array}{rcl@{\qquad}rcl} \mathrm{w}(\mathtt{0}) & := & 1 & \mathrm{w}(\mathtt{max}\ u\ v) & := & \mathrm{w}(u) + \mathrm{w}(v) + 1 \\ \mathrm{w}(\mathtt{S}\ t) & := & 2\mathrm{w}(t) + 2 & \mathrm{w}(+\ u\ v) & := & \mathrm{w}(u)\mathrm{w}(v) + 3\mathrm{w}(u) + \mathrm{w}(v) + 1 \end{array}$$

However the following properties on terms, written $\mathbf{C}(t)$, hold on the image of the translation and are preserved by rewriting with $\mathcal{R}_l$:

- t is well-typed in a context where all variables have type $\mathbb{N}$;
- $\mathtt{max}$, $+$ and $\mathtt{S}$ occur fully applied in t and variables and $\mathtt{0}$ occur unapplied in t;
- if $+\ u\ n \lhd t$ or $\mathtt{S}\ n \lhd t$ then n is closed;

Terms with these properties can (implicitly) be interpreted as natural number if they are $\mathtt{0}$ or $\mathtt{S}$-headed (since they are closed) and as level expressions in any case. Because of the definition of variable translation, $[u]$ is always (interpreted as) a guarded expression.

By induction on it, a closed term in $\mathcal{R}_l$-normal form is of the shape $\mathtt{S}^n\ \mathtt{0}$ for some $n \in \mathbb{N}$. Similarly, normal terms satisfying the $\mathbf{C}(t)$ condition are either

- $\mathtt{S}^n\ \mathtt{0}$ for some $n \in \mathbb{N}$;
- i for some variable i;
- $+\ i\ (\mathtt{S}^n\ \mathtt{0})$ for some variable i and $n > 0$;
- $\mathtt{max}\ u_1\ \ldots\ u_n$ (modulo AC) for some u_i of the above shape and with pairwise disjoint (singleton) sets of free variables and such that no two u_i are closed.

All other terms satisfying the $\mathbf{C}$ condition reduce with one of the rules in $\mathcal{R}_l$.

It is easily checked that if $\mathbf{C}(u)$ and $u \longrightarrow v$, then $\mathbf{C}(v)$. Besides, for all $\mathbb{Z}$-valuation σ, $\sigma(u) = \sigma(v)$, so that we have both $\vDash u = v$ and if $\mathbf{G}(u)$, then $\mathbf{G}(v)$.

Assume $[u] \longrightarrow\!\!\!\!\!\rightarrow t$ with t in $\mathcal{R}_l$-normal form, then both $\mathbf{C}(t)$ and $\mathbf{G}(t)$ hold and, necessarily, $t =_{\mathrm{AC}} \mathtt{max}\ (\mathtt{S}^{k_0}\ \mathtt{0})\ (+\ i_1\ (\mathtt{S}^{k_1}\ \mathtt{0}))\ \ldots\ (+\ i_n\ (\mathtt{S}^{k_n}\ \mathtt{0}))$ (or i_j if $k_j = 0$) and since $\mathbf{G}(t)$ holds, we have $k_0 \geq k_j$ for all j and this representation is unique modulo AC. □

Corollary 6.2.16.1. $\vDash u \leq v$ *if and only if* $\mathtt{max}\ [u]\ [v] \underset{\mathcal{R}_l}{\twoheadleftarrow\!\!\!\!\!\rightarrow} [v]$.

This result means that a lift operator $\mathtt{lift} : \Pi u\ v : \mathbb{N}.\ \mathtt{U}_{\mathtt{Type}_u} \to \mathtt{U}_{\mathtt{Type}_{\mathtt{max}(u,v)}}$ would allow to explicitly cast terms of type $\mathtt{Type}_u$ to their representation as type $\mathtt{Type}_v$ whenever $\vDash u \leq v$. Previously studied encoding techniques could therefore be used to faithfully represent this particular CTS with algebraic universe levels. The lack of confluence could be problematic to prove this encoding is well-behaved but this is probably manageable with extra restrictions on $\lambda\Pi_{\equiv}$, for instance by syntactically enforcing level expressions to be "confined" and to satisfy the $\mathbf{C}$ and $\mathbf{G}$ predicates.

6.3 Universe constraints

6.3.1 The constrained Calculus of Constructions

In $\mathrm{ACC}^{\sim}_{\mathcal{L}}$, we have $\vdash_{\mathcal{C}} (\Pi t : \mathtt{Type}_i.\ \Pi f : \mathtt{Type}_i \to \mathtt{Prop}.\ f\ t) : \mathtt{Prop}$. However, due to cumulativity, the two instances of the universe level i do not necessarily have to be instanti-

ated with the same concrete level to yield a judgment well-typed in ECC. For instance $\vdash_{\mathcal{C}} (\Pi t : \mathtt{Type}_1.\, \Pi f : \mathtt{Type}_2 \to \mathtt{Prop}.\, f\ t) : \mathtt{Prop}$. The more general (ill-typed) judgment $\not\vdash_{\mathcal{C}} \left(\Pi t : \mathtt{Type}_i.\, \Pi f : \mathtt{Type}_j \to \mathtt{Prop}.\, f\ t\right) : \mathtt{Prop}$ could be instantiated into this judgment but it could also yield the ill-typed $\not\vdash_{\mathcal{C}} (\Pi t : \mathtt{Type}_2.\, \Pi f : \mathtt{Type}_1 \to \mathtt{Prop}.\, f\ t) : \mathtt{Prop}$. In order to enforce only legal instances of the general judgment, we need to introduce a constraint mechanism allowing to derive $i \leq j \vdash_{\mathcal{C}} \left(\Pi t : \mathtt{Type}_i.\, \Pi f : \mathtt{Type}_j \to \mathtt{Prop}.\, f\ t\right) : \mathtt{Prop}$ where the *constraint* $i \leq j$ means that only some valuations should be considered to preserve well-typedness.

Definition 6.3.1. *The* interpretation *of a set of constraints along an assignment σ is defined by interpreting the inequalities on the natural numbers. We say that σ* validates *the constraint set ϕ, and we write $\sigma \vDash \phi$, if the inequalities of ϕ, interpreted in $\mathbb{N}$, are true.*

A set of constraints ϕ is stratifiable*, written $\phi \vDash$, if there exists and assignment that validates ϕ. A set of constraints ϕ* entails *a (set of) constraint(s) ψ written $\phi \vDash \psi$, if for all interpretation σ, $\sigma \vDash \phi \Rightarrow \sigma \vDash \psi$.*

As before, a term t with level variables should be considered *well-typed in a set of constraints* ϕ if for all interpretation σ validating ϕ, $t\sigma$ is well-typed in ECC. This statement becoming empty in the case where ϕ is not stratifiable, well-formed signatures should only allow definition and declaration which body and type annotation are well-typed in a stratifiable set of constraints.

Definition 6.3.2 (Constrained Calculus of Constructions)**.** *Assume a set of universe variables $\mathcal{L}$ and a stratifiable set of constraints ϕ such that $\mathcal{LV}ar(\phi) \subseteq \mathcal{L}$.*

We define the level conversion modulo ϕ*, as the relation $\equiv_{\mathcal{L}\phi}$ on levels (and extended to sorts and terms) such that $u \equiv_{\mathcal{L}\phi} v \iff \phi \vDash u = v$.*

We define the CTS *with following specification*

$$\mathrm{CCC}^{\sim}_{\mathcal{L}} := \mathrm{CTS}\left[\begin{array}{l} \mathcal{S} := \mathcal{S}_{\mathcal{L}}/\equiv_{\mathcal{L}\phi} \\ \mathcal{A}, \mathcal{R} \text{ and } \mathcal{C} \text{ same as in } \mathrm{ACC}^{\sim}_{\mathcal{L}} \end{array}\right]$$

As for previous extensions, we still have the following properties.

Lemma 6.3.3. *The $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{C}$ relation are well defined on classes of equivalence of $\mathcal{S}$.*
The $\mathcal{A}$ and $\mathcal{R}$ relations of $\mathrm{CCC}^{\sim}_{\mathcal{L}}$ are total functions.
Any assignment σ defines a morphism from $\mathrm{CCC}^{\sim}_{\mathcal{L}}$ to CC_{ω}.
$\mathrm{CCC}^{\sim}_{\mathcal{L}}$ is strongly normalizing.

Lemma 6.3.4. *The strict part of the $\mathcal{C}$ relation of $\mathrm{CCC}^{\sim}_{\mathcal{L}}$ is well-founded.*

Proof. The proof is similar to that of Lemma 6.2.12.

The differences are that now $\forall i, \mathcal{LV}ar(u_i) \subseteq \mathcal{LV}ar(u_0) \cup \mathcal{LV}ar(\phi)$ which is still finite and the $(k^i_m)_{i\in\mathbb{N}}$ must all be bounded (for instance by $\sigma(u_0)$ for any assignment σ such that $\sigma \vDash \phi$). This means that the possible $(n+1)$-tuples $(k^i_m)_{0\leq m\leq n}$ are in finite number and therefore $\vDash u_i = u_j$ for some $i < j$. Since $\phi \nvDash u_j = u_{j-1}$ there is some assignment σ validating ϕ and such that $\sigma(u_{j-1}) < \sigma(u_j)$. Besides, since $\sigma \vDash \phi$, we also have $\sigma(u_i) \leq \sigma(u_{i+1}) \leq \cdots \leq \sigma(u_{j-1})$ so we get $\sigma(u_i) < \sigma(u_j) = \sigma(u_i)$ allowing to conclude. □

All properties required for the correctness of the usual encoding are again satisfied for $\mathrm{CCC}^{\sim}_{\mathcal{L}}$. However, correctly encoding this CTS now requires a level representation allowing to decide sort equality and subtyping in a set of constraints.

6.3.2 Constraint atomicity

As discussed later, Herbelin showed that in the particular case where the signature is elaborated from a signature where universe levels are left anonymous (satisfying the typical ambiguity property), only *atomic* constraints are inferred. Therefore, enforcing the property that constraints are atomic is enough to derive the well-typedness of all signatures elaborated from typical ambiguous ones. Restricting to atomic constraints provides a simple stratifiability check and makes the problems of deciding inequality and equality of levels much easier which is helpful in the context of constraint and universe encoding in $\lambda\Pi_{\equiv}$.

Definition 6.3.5. *A level expression u is* atomic *if it is either a concrete level or a level variable, $u \in \mathbb{N} \cup \mathcal{L}$. A constraint $u \leq l$ is* atomically upper-bounded *if l is atomic. Constraints $l \leq l'$ and $l + k \leq l'$ are* atomic *if l and l' are. We write $u = v$ for the pair, or conjunction, of constraints $u \leq v$ and $v \leq u$. $u = v$ is atomic if both u and v are.*

Lemma 6.3.6. *For all set of atomically upper-bounded constraints there is an* equivalent *set of atomic constraints with the exact same validating interpretations.*

Proof. By induction on the multiset of constraint sizes in the right-hand side, using the following easy equivalences

$$\begin{array}{lll} \phi \wedge \quad \max(u,v) \leq l & \iff & \phi \wedge u \leq l \wedge v \leq l \\ \phi \wedge \max(u,v) + k \leq l & \iff & \phi \wedge u + k \leq l \wedge v + k \leq l \\ \phi \wedge \quad (u+k) + n \leq l & \iff & \phi \wedge u + (k+n) \leq l \end{array}$$

These equivalences cover all cases of atomically upper-bounded sets of constraints that are not all already atomic. □

Definition 6.3.7. *If ϕ is a set of atomic constraints, we define the relation $\leq^k_\phi$ on atomic levels ($\mathcal{L} \cup \mathbb{N}$) such that $i \leq^k_\phi j \Leftrightarrow i + k \leq j \in \phi$, $i \leq^0_\phi n \Leftrightarrow i \leq n \in \phi$ and $n \leq^k_\phi i \Leftrightarrow ((n+k) \leq i \in \phi \vee n = k = 0)$. We write $\leq_\phi := \bigcup_k \leq^k_\phi$ and $\leq^{(K)}_\phi := \bigcup_{\sum k_i = K} \leq^{k_1}_\phi \cdots \leq^{k_n}_\phi$.*

A set of atomic constraints can be seen as a graph with nodes in $\mathcal{L} \cup \mathbb{N}$ and edge labels in $\mathbb{N}$. $l \leq^{(K)}_\phi l'$ corresponds to the existence of a path or chain of constraints from l to l'.

Sets of atomic constraints are convenient because their stratifiability can be checked by simply looking for either non-trivial cycles $i \leq^{(K)}_\phi i$ or chains of constraints between concrete levels $m \leq^{(K)}_\phi n$ such that $K > n - m$. We also assume implicit edges $n \leq^1_\phi n + 1$ which allow to characterize this second condition using only chains starting at 0.

Theorem 6.3.8. *A set ϕ of atomic constraints is stratifiable if and only if there exists no cycle $i \leq^{(K)}_\phi i$ for $K > 0$ and no chain $0 \leq^{(K)}_\phi n$ for $K > n \in \mathbb{N}$.*

Proof. It is easy to see that if $\sigma \vDash \phi$, then $l \leq^k_\phi l' \Rightarrow \sigma(l) + k \leq \sigma(l')$ and $l \leq^{(K)}_\phi l' \Rightarrow \sigma(l) + K \leq \sigma(l')$. Assuming there exists $i \leq^{(K)}_\phi i$ (resp. $0 \leq^{(K)}_\phi n$) then any assignment σ validating ϕ must satisfy $\sigma(i) + K \leq \sigma(i)$ (resp. $K \leq n$) which is impossible.

Assuming there is no such cycle, since ϕ is finite, $M := 2|\mathcal{LV}ar(\phi)| \cdot \max(\{k \mid i + k \leq j \in \phi\})$ is well-defined and $l \leq^{(N)}_\phi l' \Rightarrow N \leq M$. Indeed, assuming the smallest n such that $l \leq^{k_1}_\phi \cdots \leq^{k_n}_\phi l'$ and $\Sigma_p k_p > M$, then either $n \leq 2|\mathcal{LV}ar(\phi)|$ and the bound is respected or $n > 2|\mathcal{LV}ar(\phi)|$ and since $p \not\leq^k_\phi q$ for $p, q \in \mathbb{N}$, there must be a level variable i and a non-empty cycle $l \leq^{k_0}_\phi \cdots \leq^{k_p}_\phi i \leq^{k_{p+1}}_\phi \cdots \leq^{k_{p+q}}_\phi i \leq^{k_{p+q+1}}_\phi \cdots \leq^{k_n}_\phi l'$ which can be eliminated to form a strictly smaller sequence since, by assumption, $\sum_{1 \leq j \leq q}, k_{p+j} = 0$.

This allows to define the assignment $\sigma_m(i) := \max\left(\left\{K \mid 0 \leq^{(K)}_\phi i\right\}\right)$. It is easy to check that σ_m validate ϕ since for all $i + k \leq j \in \phi$ we have

$$\begin{aligned}\sigma_m(j) &= \max(\{\Sigma_p k_p \mid 0 \leq^{k_1}_\phi \cdots \leq^{k_n}_\phi j\}) \geq \max(\{\Sigma_p k_p + k \mid 0 \leq^{k_1}_\phi \cdots \leq^{k_n}_\phi i \leq^k_\phi j\}) \\ &\geq \max(\{\Sigma_p k_p \mid 0 \leq^{k_1}_\phi \cdots \leq^{k_n}_\phi i\}) + k = \sigma_m(i) + k\end{aligned}$$

Similarly σ_m validates the $n \leq i$ constraints in ϕ. Assume $i \leq n \in \phi$, then $0 \leq^{\sigma_m(i)}_\phi i \leq^0_\phi n$ and by assumption, $\sigma_m(i) \leq n$.

Since the $\sigma_m(i)$ is always obtained from some $0 \leq^{(N)}_\phi j$ Is is easy to see that σ_m is minimal: for all assignment σ validating ϕ, $\sigma_m(i) \leq \sigma(i)$. □

For instance the sets $\phi_1 = \{i \leq j, j + 1 \leq i\}$ and $\phi_2 = \{j + 2 \leq i, i \leq 1\}$ are not stratifiable since there exists $j \leq^1_{\phi_1} i \leq^0_{\phi_1} j$ and $0 \leq^0_{\phi_2} j \leq^2_{\phi_2} i \leq^0_{\phi_2} 1$.

Checking the absence of cycle in a set of atomic constraints can be done in polynomial time by checking this property is preserved when successively adding constraints to a set, as it is done in the Coq system. One way to do this is to keep a map from pairs of level variables (i, j) to the smallest K, if any, such that $i \leq^{(K)}_\phi j$. This map can be updated in polynomial time to take a new constraint into account. Since Coq exclusively features $i \leq j$ and $i + 1 \leq j$ constraints stratifiability is even more easily checked by simply looking for cycles with a strict inequality.

6.4 Deciding cumulativity under constraints

6.4.1 Decision procedure for level constraints

Courant defined [Cou02] an algorithm to decide constraints validity in a stratifiable set of variable upper-bounded constraints. In his presentation level variables are introduced successively with a single constraint $u \leq i$ where i is a fresh variable and u is a level expression relying exclusively on already defined level variables. This condition restricts the allowed sets of constraints considered, forbidding for instance the stratifiable $i \leq j \leq i + 2$. However, stratifiable set of variable upper-bounded constraints can always be defined this way. Constraints with the same right-hand side variable can be by reordered and grouped together: $u \leq i \wedge v \leq i \Rightarrow \max(u, v) \leq i$. Variables in cycles can be identified

$i \leq j \leq k \leq i \Rightarrow i = j = k$ and collapsed. Level variable declaration can be intertwined with variable declarations in the context, unlike in $\mathrm{CCC}^{\sim}_{\mathcal{L}}$ where all levels and constraints are fully defined before type checking. We show that Courant's algorithm can be adapted to handle any stratifiable set of atomic constraints, including $i \leq n$ constraints and cyclic sets, $j \leq i \wedge j \leq i$. This allows to define an encoding of constraint as predicate in $\lambda\Pi_{=}$: $\vdash_{\mathcal{D}} [u \leq v] : *$ such that $[u \leq v]$ is inhabitable if and only if $\phi \vDash u \leq v$.

This algorithm relies on a special constraint $u \leq_n v$ which is interpreted as $u \leq v + n$ for $n \in \mathbb{Z}$.

$$\frac{0 \leq n}{\phi \vdash_{\mathcal{S}} 0 \leq_n u}\,\mathcal{L}_0 \qquad \frac{0 \leq n}{\phi \vdash_{\mathcal{S}} u \leq_n u}\,\mathcal{L}_a \qquad \frac{u \leq i \in \phi}{\phi \vdash_{\mathcal{S}} u \leq_0 i}\,\mathcal{L}_{\mathrm{ax}}$$

$$\frac{\phi \vdash_{\mathcal{S}} u \leq_{m-n} v}{\phi \vdash_{\mathcal{S}} u + n \leq_m v}\,\mathcal{L}_l \qquad \frac{\phi \vdash_{\mathcal{S}} u \leq_{m+n} v}{\phi \vdash_{\mathcal{S}} u \leq_m v + n}\,\mathcal{L}_r \qquad \frac{\phi \vdash_{\mathcal{S}} (u \leq_n w) \wedge (v \leq_n w)}{\phi \vdash_{\mathcal{S}} \max(u, v) \leq_n w}\,\mathcal{L}_{\max}$$

$$\frac{\phi \vdash_{\mathcal{S}} u \leq_n v \qquad \phi \vdash_{\mathcal{S}} v \leq_0 w}{\phi \vdash_{\mathcal{S}} u \leq_n w}\,\mathcal{L}_{\mathrm{tr}} \qquad \frac{\phi \vdash_{\mathcal{S}} c_1 \qquad \phi \vdash_{\mathcal{S}} c_2}{\phi \vdash_{\mathcal{S}} c_1 \wedge c_2}\,\mathcal{L}_{\wedge}$$

$$\frac{\phi \vdash_{\mathcal{S}} u \leq_n v}{\phi \vdash_{\mathcal{S}} u \leq_n \max(v, w)}\,\mathcal{L}_{\mathrm{maxL}} \qquad \frac{\phi \vdash_{\mathcal{S}} u \leq_n w}{\phi \vdash_{\mathcal{S}} u \leq_n \max(v, w)}\,\mathcal{L}_{\mathrm{maxR}}$$

Figure 6.1: Constraint derivation rules

Lemma 6.4.1 (Correctness of $\vdash_{\mathcal{S}}$)**.** *Assume a stratifiable set ϕ of atomic constraints. If $\phi \vdash_{\mathcal{S}} u \leq_0 v$ can be derived using the inference rules of Figure 6.1, then $\phi \vDash u \leq v$.*

Proof. This is done by a simple induction on the derivation $\phi \vdash_{\mathcal{S}} u \leq v$. All inference rules are correct: if a level assignment σ validates the premises, it validates the conclusion. □

While correctness is easy, completeness is going to require two technical lemmas.

Lemma 6.4.2. *Assume a stratifiable set ϕ of atomic constraints. We have $\phi \vDash i+p \leq j+q$ if and only if either $i \leq^{(K)}_{\phi} j$ for some K such that $p \leq K + q$ or $i \leq^{(K)}_{\phi} N$, $M \leq^{(K')}_{\phi} j$ and $N - K + p \leq M + K' + q$.*

Proof. If $i \leq^{(K)}_{\phi} j$ then in all model σ validating ϕ, we have $\sigma(i)+p \leq \sigma(i)+K+q \leq \sigma(j)+q$. The same goes for the other criteria.

To prove the other direction we consider the following assignment for $M \in \mathbb{N}$:

$$\sigma_{(M)}(i) := \max\left(\left\{ k \;\middle|\; \begin{array}{ll} i \leq^{(K)}_{\phi} j & \Rightarrow k + K \leq \sigma_m(j) \\ i \leq^{(K)}_{\phi} n & \Rightarrow k + K \leq n \\ i \leq^{(K)}_{\phi} i' \in \mathcal{L} & \Rightarrow k + K \leq M \end{array} \right\}\right)$$

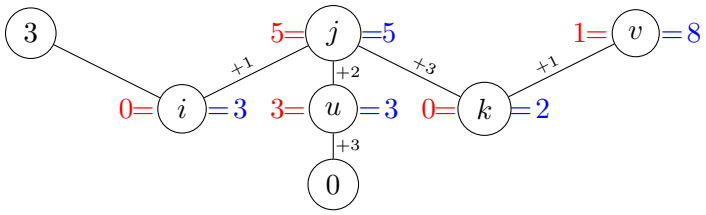
For M large enough, $\sigma_{(M)}(j) = \sigma_m(j)$ and $\sigma_{(M)}$ validates ϕ. This is easy to see for $i \leq n$ and $i + k \leq j$ constraints. For $n \leq i$ constraints, there are two cases:

- $\sigma_{(M)}(i) = \sigma_m(j) - K$ for some $i \leq_\phi^{(K)} j$ or $\sigma_{(M)}(i) = n' - K$ for some $i \leq_\phi^{(K)} n'$. In both cases, necessarily $n \leq \sigma_{(M)}(i)$ by stratifiability of ϕ.
- $\sigma_{(M)}(i) = M - \max(\{K | i \leq_\phi^{(K)} i'\})$ which is greater than n if M is chosen large enough.

When M is chosen large enough we are in one of three cases:

- $\sigma_{(M)}(i) = \sigma_m(j) - K$ for some K such that $i \leq_\phi^{(K)} j$ and we get the result immediately.
- $\sigma_{(M)}(i) = n - K$ for some K such that $i \leq_\phi^{(K)} n$. and we have $n - K + p \leq \sigma_m(j) + q$. By definition of σ_m, $0 \leq_\phi^{(\sigma_m(j))} \sigma_m(j)$ which gives the result.
- $\sigma_{(M)}(i) = M - \max(\{K | i \leq_\phi^{(K)} i'\})$ and if M is chosen large enough, $\sigma_{(M)} \not\models i + p \leq j + q$, contradicting the assumption. □

Example 1: For instance, the set of constraints represented below (black) is validated with both σ_m (in red) and $\sigma_{(8)}$ (in blue). The σ_m assignment maps variables to the lowest level compatible with the constraints while $\sigma_{(M)}$ maps j to $\sigma_m(j)$ and all other to the highest possible level below M.



We have $\phi \models k + 2 \leq j$ since $k \leq_\phi^{(3)} j$ and $\phi \models i + 2 \leq j$ since $i \leq_\phi^{(0)} 3$ and $0 \leq_\phi^{(5)} j$.

Lemma 6.4.3. *Assume a stratifiable set ϕ of atomic constraints. We have $\phi \models i + p \leq \max(u, v)$ if and only if $\phi \models i + p \leq u$ or $\phi \models i + p \leq v$.*

Proof. By Lemma 6.2.5, $\phi \models u = \max(i_1^u + k_1^u, ...i_n^u + k_n^u)$ and $\phi \models v = \max(i_1^v + k_1^v, ...i_m^v + k_m^v)$. We introduce a new variable ι and add the new constraints $i_j^u + k_j^u \leq \iota$ and $i_j^v + k_j^v \leq \iota$ to the set ϕ for all j. The defined set ψ is still stratifiable as any assignment σ validating ϕ can be extended in an assignment σ' validating ψ by defining $\sigma'(\iota) := \sigma(\max(u, v))$. We also have $\psi \models i + p \leq \iota$ so, by Lemma 6.4.2, we have either

- $i \leq_\psi^{(K)} \iota$ for some K such that $p \leq K$. The chain must start with one of the introduced constraint since only they refer to ι and these constraint occur nowhere else since none of them has ι as a lower bound. Therefore $i \leq_\phi^{k_1} \cdots \leq_\phi^{k_n} i_j^u \leq_\psi^{k_j^u} \iota$. We conclude that $\phi \models i + p + K - k_j^w \leq i_j^w + p$ and $\phi \models i + p \leq i_j^w + k_j^w$ for $w \in \{u, v\}$ and some j.
- $i \leq_\psi^{(K)} N$, $M \leq_\psi^{(K')} \iota$ and $N - K + p \leq M + K'$ which is handled the same way with the same conclusion.

We conclude that $\phi \models i + p \leq w$ for $w \in \{u, v\}$. The other direction is immediate. □

Lemma 6.4.4 (Completeness of $\vdash_{\mathcal{S}}$)**.** *Assume a stratifiable set ϕ of atomic constraints. If $\phi \vDash u \leq v$, then $\phi \vdash_{\mathcal{S}} u \leq_0 v$ can be derived using the inference rules of Figure 6.1.*

Proof. The proof is done in [Cou02] for a slightly simpler but just as powerful system. We adapted the system here to later simplify the definition of the translation function on these derivations.

The $\mathcal{L}_{\text{ax}}$, $\mathcal{L}_{\text{tr}}$, $\mathcal{L}_{\text{maxL}}$ and $\mathcal{L}_{\text{maxR}}$ rules are not invertible, their premises are not equivalent to their conclusions so they should be used with more care.

We prove a more general result, if $\phi \vDash u \leq v + n$ then $\phi \vdash_{\mathcal{S}} u \leq_n v$, by induction on the size of (u, v). The invertible rules $\mathcal{L}_0$, $\mathcal{L}_{\text{ax}}$, $\mathcal{L}_a$, $\mathcal{L}_l$, $\mathcal{L}_{\text{max}}$ and $\mathcal{L}_\wedge$ are terminating on any judgment $u \leq_n v$ as their premises have smaller sizes than their conclusion. Any judgment that matches the conclusion of one of these rules can be derived, by induction hypothesis, using it.

A judgment that matches the conclusion of no invertible rule is either of the shape $i \leq_n \max(u, v)$ or $i \leq_n j$. If it is of the first shape then, by Lemma 6.4.3, either $\phi \vDash i \leq_n u$ or $\phi \vDash i \leq_n v$ and we conclude using the $\mathcal{L}_{\text{maxL}}$ or $\mathcal{L}_{\text{maxR}}$ rule and by induction hypothesis.

Otherwise we have $\phi \vDash i \leq j + n$, or rather $\phi \vDash i - n \leq j$ if $n < 0$. By Lemma 6.4.2 we are in one these two cases:

- $i \leq_\phi^{(K)} j$ for some K such that $0 \leq K + n$.
  By induction on the chain length, $\phi \vdash_{\mathcal{S}} i \leq_n j$ can be derived using the $\mathcal{L}_{\text{ax}}$, $\mathcal{L}_{\text{tr}}$ rules and the invertible rules for all $0 \leq K + n$. For empty chains, $\phi \vdash_{\mathcal{S}} i \leq_n i$ is derived with $\mathcal{L}_a$ if $0 \leq n$. If $i \leq_\phi^{(K)} l \leq_\phi^k j$ and $0 \leq n + (K + k)$, then $\mathcal{L}_{\text{ax}}$ derives $\phi \vdash_{\mathcal{S}} l + k \leq_0 j$ and by $\mathcal{L}_r$,$\phi \vdash_{\mathcal{S}} i \leq_n l + k$ can be derived from $\phi \vdash_{\mathcal{S}} i \leq_{n+k} l$ which is itself derivable by induction hypothesis since $0 \leq (n + k) + K$ and $i \leq_\phi^{(K)} l$. Note that this work if $j \in \mathbb{N}$ as well but that case is a bit more tedious and requires to use either $\mathcal{L}_r$ backward or rely on a more general $\mathcal{L}_{\text{tr}}$ with the derivable $\phi \vdash_{\mathcal{S}} n \leq_n 0$.
- $i \leq_\psi^{(K)} N$, $M \leq_\psi^{(K')} j$ and $N - K + p \leq M + K' + q$ which is handled the same way with the same conclusion.

All cases are covered and $\phi \vdash_{\mathcal{S}} u \leq_n v$ is derivable if $\phi \vDash u \leq v + n$. □

The inference rules define therefore a non-deterministic algorithm which can actually be done in polynomial time in practice since, in the max case, it is possible to decide in linear time which branch is derivable.

Using a level representation similar to that of Definition 6.2.15, this allows to encode provable subtyping in $\lambda\Pi_\equiv$.

Definition 6.4.5. *We consider the $\lambda\Pi_\equiv$ signature Σ_{c} encoding levels and constraints defined in Figure 6.2. Algebraic level expressions are translated as terms of type $\mathbb{N}$ and constraints are translated as terms of type $\mathbb{B}$ as follows:*

$$
\begin{aligned}
[i] &:= i & [u + n] &:= \mathtt{S}^n\ [u] \\
[n] &:= \mathtt{S}^n\ \mathtt{0} & [\max(u, v)] &:= \mathtt{max}\ [u]\ [v] \\
[u \leq v] &:= [u]\ \leq\ [v] & [u = v] &:= [u \leq v]\ \wedge\ [v \leq u]
\end{aligned}
$$

$\mathbb{N} : *$	$\mathbb{B} : *$	
$\mathtt{0} : \mathbb{N}$	$\square \leq \square : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$	
$\mathtt{S} : \mathbb{N} \to \mathbb{N}$	$\square \wedge \square : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$	$(\wedge \in \mathcal{F}_{\mathrm{AC}})$
$\mathtt{max} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$	$\mathtt{max}\ (\mathtt{S}\ X)\ (\mathtt{S}\ Y) \longrightarrow \mathtt{S}\ (\mathtt{max}\ X\ Y)$	(max-S)
$\top : \mathbb{B}$	$\top \wedge C \longrightarrow C$	($\wedge$-$\top$)

Figure 6.2: Public encoding signature (levels and constraints): $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{pub}}$

Finite sets $\mathcal{L}$ of level variables are supposed ordered and are translated to a context of variable declarations: $[\{i_1, \ldots, i_n\}] := i_1 : \mathbb{N}, \ldots, i_n : \mathbb{N}$. *Finite sets of atomic constraints with level variables in $\mathcal{L}$ are supposed ordered and are translated to a context of variable declarations* $[\{\phi_1, \ldots, \phi_n\}] := c_1 : \boldsymbol{\epsilon}\ [\phi_1], \ldots, c_n : \boldsymbol{\epsilon}\ [\phi_n]$.

We also consider the $\lambda\Pi_{\equiv}$ signature Σ_{p} of constraint proofs *defined in Figure 6.3.*

Having defined a representation of levels and constraints in $\lambda\Pi_{\equiv}$ we can now state and prove its *correctness*. In our case, correctness must link constraint entailment in the original system, $\phi \vDash c$, to the inhabitance of the type representing c in $\lambda\Pi_{\equiv}$.

Theorem 6.4.6 (Correctness)**.** *Assume a finite set $\mathcal{L}$ of level variables.*

- *For all level expression u, such that $\mathcal{LV}ar(u) \subseteq \mathcal{L}$, we have $\Sigma_{\mathrm{c}}, \Sigma_{\mathrm{p}}; [\mathcal{L}] \vdash_{\mathcal{D}} [u] : \mathbb{N}$.*
- *For all constraint c such that $\mathcal{LV}ar(c) \subseteq \mathcal{L}$, we have $\Sigma_{\mathrm{c}}, \Sigma_{\mathrm{p}}; [\mathcal{L}] \vdash_{\mathcal{D}} [c] : \mathbb{B}$.*
- *For all set of atomic constraints ϕ such that $\mathcal{LV}ar(\phi) \subseteq \mathcal{L}$, $\Sigma_{\mathrm{c}}, \Sigma_{\mathrm{p}} \vdash_{\mathcal{D}} [\mathcal{L}], [\phi]$ $\mathbf{WF}_{\mathcal{D}}$.*
- *If $\phi \vDash c$ then there exists a term t such that $\Sigma_{\mathrm{c}}, \Sigma_{\mathrm{p}}; [\mathcal{L}], [\phi] \vdash^{\mathrm{WF}}_{\mathcal{D}} t : \boldsymbol{\epsilon}\ [c]$.*
- *The inhabitance of $\boldsymbol{\epsilon}\ ([c])$ in the signature $\Sigma_{\mathrm{c}}, \Sigma_{\mathrm{p}}$ and context $[\mathcal{L}], [\phi]$ is decidable.*

Proof. Well-typedness of level expressions and constraints is straightforward by definition. For $n \in \mathbb{Z}$, we write $n^+ := \max(0, n) \in \mathbb{N}$ and $n^- := \max(0, -n) \in \mathbb{N}$. We extend the translation to $u \leq_n v$ constraints: $[u \leq_n v] := [u]\ {}_{[n^-]}\!\leq_{[n^+]}\ [v]$. We prove by induction on the derivation that if $\phi \vdash_{\mathcal{S}} u \leq_n v$ is derivable, then $\boldsymbol{\epsilon}\ ([u \leq_n v])$ is inhabited in signature $\Sigma_{\mathrm{c}}, \Sigma_{\mathrm{p}}$ and context $[\mathcal{L}], [\phi]$:

- $\mathcal{L}_0$: $\boldsymbol{\epsilon}\ (\mathtt{0}\ {}_{\mathtt{0}}\!\leq_{(\mathtt{S}^n\mathtt{0})}\ [v]) \underset{\mathcal{R}}{\longrightarrow} \boldsymbol{\epsilon}\ \top$ which is inhabited with $\mathbf{I}$.
- $\mathcal{L}_a$: $\boldsymbol{\epsilon}\ ([u]\ {}_{\mathtt{0}}\!\leq_{(\mathtt{S}^n\mathtt{0})}\ [u])$ is inhabited with $\mathtt{refl}\ [u]\ (\mathtt{S}^n\mathtt{0})$.
- $\mathcal{L}_l$, $\mathcal{L}_r$, $\mathcal{L}_{\max}$: in all cases, the translation of the unique premise is convertible with the translation of the conclusion, we conclude by induction hypothesis.
- $\mathcal{L}_{\mathrm{ax}}$: the inhabitant is the variable c_j from the context $[\phi]$ such that $\phi_j = (u \leq i)$.
- $\mathcal{L}_{\mathrm{tr}}$, $\mathcal{L}_{\wedge}$, $\mathcal{L}_{\mathrm{maxL}}$ and $\mathcal{L}_{\mathrm{maxR}}$: the inhabitant is built with those (π_1, π_2) obtained by induction hypothesis on the premises and using the constructors. Respectively:
  - $\mathtt{tr}\ (\mathtt{S}^{n^-}\ [u])\ (\mathtt{S}^{n^+}\ [v])\ (\mathtt{S}^{n^+}\ [w])\ \pi_1\ \pi_2$
  - $\mathtt{pair}\ [c_1]\ [c_2]\ \pi_1\ \pi_2$
  - $\mathtt{maxl}\ (\mathtt{S}^{n^-}\ [u])\ (\mathtt{S}^{n^+}\ [v])\ (\mathtt{S}^{n^+}\ [w])\ \pi_1$

$$
\begin{aligned}
\square\ {}_{\square}{\leq}_{\square}\square &: \mathbb{N} \to \mathbb{N} \to \mathbb{N} \to \mathbb{N} \to \mathbb{B} \\
\boldsymbol{\epsilon} &: \mathbb{B} \to * \\
\mathbf{I} &: \boldsymbol{\epsilon}\ \top \\
\texttt{refl} &: \Pi i\ n{:}\mathbb{N}.\ \boldsymbol{\epsilon}\ (i\ {}_{0}{\leq}_{n} i) \\
\texttt{tr} &: \Pi i\ j\ k{:}\mathbb{N}.\ \boldsymbol{\epsilon}\ (i \leq j) \to \boldsymbol{\epsilon}\ (j \leq k) \to \boldsymbol{\epsilon}\ (i \leq k) \\
\texttt{maxl} &: \Pi i\ j\ k{:}\mathbb{N}.\ \boldsymbol{\epsilon}\ (i \leq j) \to \boldsymbol{\epsilon}\ (i \leq \texttt{max}\ j\ k) \\
\texttt{maxr} &: \Pi i\ j\ k{:}\mathbb{N}.\ \boldsymbol{\epsilon}\ (i \leq k) \to \boldsymbol{\epsilon}\ (i \leq \texttt{max}\ j\ k) \\
\texttt{pair} &: \Pi c\ d{:}\mathbb{B}.\ \boldsymbol{\epsilon}\ c \to \boldsymbol{\epsilon}\ d \to \boldsymbol{\epsilon}\ (c \wedge d)
\end{aligned}
$$

$$
\begin{aligned}
U \leq V &\longrightarrow U\ {}_{0}{\leq}_{0} V && (\leq) \\
\texttt{0}\ {}_{0}{\leq}_{N} X &\longrightarrow \top && (\texttt{0}\text{-}\leq) \\
(\texttt{S}\ X)\ {}_{N}{\leq}_{M} Y &\longrightarrow X\ {}_{(\texttt{S}\ N)}{\leq}_{M} Y && (\texttt{S}\text{-}\leq) \\
X\ {}_{N}{\leq}_{M} (\texttt{S}\ Y) &\longrightarrow X\ {}_{N}{\leq}_{(\texttt{S}\ M)} Y && (\leq\text{-}\texttt{S}) \\
X\ {}_{(\texttt{S}\ N)}{\leq}_{(\texttt{S}\ M)} Y &\longrightarrow X\ {}_{N}{\leq}_{M} Y && (\texttt{S}\text{-}\leq\text{-}\texttt{S}) \\
(\texttt{max}\ X\ Y)\ {}_{N}{\leq}_{M} Z &\longrightarrow (X\ {}_{N}{\leq}_{M} Z) \wedge (Y\ {}_{N}{\leq}_{M} Z) && (\texttt{max}\text{-}\leq)
\end{aligned}
$$

Figure 6.3: Public encoding signature (constraints proofs): $\Sigma_{\mathrm{p}} \subseteq \Sigma_{\mathrm{pub}}$

– $\texttt{maxr}\ (\texttt{S}^{n^-}\ [u])\ (\texttt{S}^{n^+}\ [v])\ (\texttt{S}^{n^+}\ [w])\ \pi_1$

Assuming $c = u \leq v$ for level expressions u and v. By Lemma 6.4.4, $\phi \vdash_{\mathcal{S}} u \leq_0 v$ is derivable. Therefore $\boldsymbol{\epsilon}\ [u \leq_0 v]$ is inhabited since $[u \leq v] \equiv_{\beta\mathcal{R}} [u \leq_0 v]$ by the $\leq$ rule. Besides the above proof is a terminating algorithm to compute one of its inhabitants or conclude that none exist in the case of a constraint for which none of the rules from Figure 6.1 apply. The $c = (u = v)$ case is similar. □

Note that the use of the non-invertible rules $\mathcal{L}_{\mathrm{ax}}$, $\mathcal{L}_{\mathrm{tr}}$, $\mathcal{L}_{\mathrm{maxL}}$ or $\mathcal{L}_{\mathrm{maxR}}$ needs to be represented with explicit constructors while other rules may be reflected with rewrite rules.

Example 2: We consider the same set ϕ of constraints as in Example 1. We have

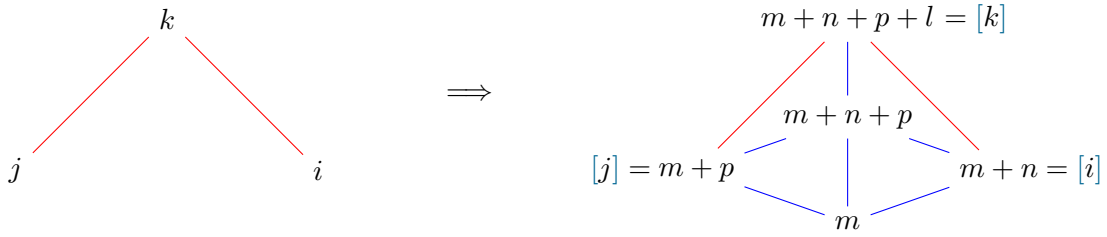
$$\Sigma_{\mathrm{c}}, \Sigma_{\mathrm{p}} \vdash_{\mathcal{D}} i, j, k, u, v : \mathbb{N}\ ,\ c_{i3} : \boldsymbol{\epsilon}\ (i{\leq}\texttt{S}^3\ \texttt{0})\ ,\ \cdots\ ,\ c_{kv} : \boldsymbol{\epsilon}\ (\texttt{S}\ k{\leq}v)\ \mathbf{WF}_{\mathcal{D}}$$

- $\boldsymbol{\epsilon}\ [\max(i,1) + 2 \leq i + 3] \longrightarrow\!\!\!\!\rightarrow \boldsymbol{\epsilon}(i\ {}_{0}{\leq}_{(\texttt{S}\ \texttt{0})} i)$ is inhabited with $\texttt{refl}\ i\ (\texttt{S}\ \texttt{0})$,
- $\boldsymbol{\epsilon}\ [k + 2 \leq j]$ inhabited with $c_{kj} \in [\phi]$.
- $\boldsymbol{\epsilon}\ [5 \leq j]$ inhabited with $\pi := \texttt{tr}\ (\texttt{S}^5\texttt{0})\ (\texttt{S}^2 u)\ j\ c_{0u}\ c_{uj}$ with $c_{0u}, c_{uj} \in [\phi]$.
- $\boldsymbol{\epsilon}\ [i + 2 \leq j]$ inhabited with $\texttt{tr}\ (\texttt{S}^2 i)\ (\texttt{S}^5\texttt{0})\ j\ c_{i3}\ \pi$

6.4.2 Encoding constrained universe levels

While allowing to correctly reflect the satisfiability of constraints, the representation of universes from Definition 6.2.15 or Definition 6.4.5 does not computationally reflect the universe equality under constraints. For instance, the equality $i \leq j \vDash \max(i,j) = j$ is only provably reflected in the encoding $\exists t, \Sigma_c, \Sigma_p; i, j : \mathbb{N}, c : \boldsymbol{\epsilon}\ (i \leq j) \vdash^{\mathrm{WF}}_{\mathcal{D}} t : \boldsymbol{\epsilon}\ [\max(i,j) = j]$ which does not ensure $[\max(i,j)] \equiv_{\beta\mathcal{R}} [j]$. The guard condition which ensured a canonical representation now depends on the constraint set. For instance, in the constraint $i+1 \leq j$, the level expression $\max(0, j)$ is not properly guarded since it may be mapped to 0 with a $\mathbb{Z}$-assignment and at least to 1 with $\mathbb{N}$-assignments. We investigate here several ways to encode these levels as $\lambda\Pi_{\equiv}$ terms. Our main concerns when designing these embeddings are that if $\phi \vDash u = v$ then u and v should be convertible. We explore several ways to ensure this property or go around it.

Embed constraints with sums: Assaf, Dowek, Jouannaud and Liu investigated [ADJL16] a representation of universe levels relying on the `max`/$+$ algebra with both symbols in $\mathcal{F}_{\mathrm{AC}}$ together with rewrite rules. The idea is that a constraint $i \leq j$ can be represented by introducing a new level k such that $j := i + k$. Therefore universe instances of j should directly be translated as $(+\ i\ k)$ and in our previous example we could have $[\max(i,j)] = \texttt{max}\ [i]\ [j] = \texttt{max}\ i\ (\texttt{plus}\ i\ k) \underset{\mathcal{R}}{\longrightarrow} i$. This encoding requires both `max` and $+$ to be AC and and is proven confluent modulo AC. It suffers however from a critical drawback. Assume you need to reflect two inequalities $i \leq k$ and $j \leq k$, it is not possible to have at the same time $k := i + k_i$ and $k := j + k_j$ otherwise the translation of the universe variable k would be ambiguous. Instead we use two other levels such that $i = \min(i,j) + n$ and $j = \min(i,j) + p$. This allows to have $i + p = \max(i,j) = j + n$ and we can introduce a last level l such that $k = \max(i,j) + l$.



When translating i, j and k as their sum representation, the rewrite rules are now able to reflect both constraints since the translation satisfies $[i] + u = [k] = [j] + v$, for some levels $u = n + l$ and $v = p + l$. This representation however quickly blows up. For instance a single extra constraint $h \leq j$ require to build a representation $[h]$ such that $[j] = m + p = [h] + q$. This can be done by re-assigning $m := m' + q_m$, $p := p' + q_p$ and having $[h] := m' + p'$ and $q := q_m + q_p$. This doubles the size of $[j]$ and also extends the sizes of $[i]$ and $[k]$ even though they are not directly concerned with this extra constraint. Besides, with each step the newly introduced level variables are less directly linked to the original levels: n represents $i - \min(i,j)$ but q_p is harder to link to i and h.

Embed constraints in level definition: An other solution would be to define

level translation such that $[j] := \texttt{plus}\ n\ (\texttt{max}\ \{\texttt{plus}\ [i]\ [k] \mid i + k \leq j \in \phi\})$ which is linear in the size of ϕ, however unfolding definitions, yields the exponentially sized: $\texttt{plus}\, n\ \left(\texttt{max}\ \{\texttt{plus}\ [i]\ [k_0]\ \ldots\ [k_m] \mid i \leq_\phi^{(k_0 \cdots k_m)} j\}\right)$ where all the longest paths with root i in the constraint graph are considered. The weak head normal form of this definition is however polynomial but requires an exponential number of steps to compute. Since AC-matching is typically slower than regular rewriting this drawback is bound to quickly become a roadblock.

Translating directly to the normal form: Instead of computing the normal form, we could also translate universe level directly as the `max`-expression representing this canonical form.

Lemma 6.4.7. *For all level variable i, $\{j + k \mid \phi \vDash j + k \leq i\}$ is finite and we define $I_i^\phi := \max(\{j + k \mid \phi \vDash j + k \leq i\})$ We have:*

- $\phi \vDash i = I_i^\phi$.
- $\phi \vDash u = u\{i \mapsto I_i^\phi\}$
- $\phi \vDash i = j$ *iff* $\vDash I_i^\phi = I_j^\phi$

This allows to reuse the encoding from Definition 6.2.15 (without constraints) where variable translation is fully expanded: $[i] := \texttt{max}\ \{\texttt{S}^k\ j \mid \phi \vDash j + k \leq i\}$.

For instance, if $\phi = \{i \leq j, 2 \leq i, l+1 \leq j, i+1 \leq l$ then we would define the translation $[j] \equiv_{\beta\mathcal{R}} \texttt{max}\ (\texttt{S}^4\ \texttt{0})\ (\texttt{plus}\ i\ (\texttt{S}\ (\texttt{S}\ \texttt{0})))\ (\texttt{plus}\ l\ (\texttt{S}\ \texttt{0}))$.

This representation is a bit cumbersome but works fine and is stable by substitution assuming we use the encoding defined in [ADJL16].

With explicit level conversion: Rather than designing systems that reflects universe conversion (in presence of constraints) as conversion of images of the translation, we could instead rely on "provable conversion" exclusively. Inequality constraints are already represented as predicates that can be proved in the encoding and which proofs can be used to build other proof of other constraints. Equality constraints are therefore also represented as provable predicates which we can use to define an explicit universe level substitution in a type T: if $\phi \vDash u = v$ then u can be substituted with v in T to yield a convertible term T'. This property is sufficient to express universe level conversion and can be enforced by the following typing rule which allows to identify inhabitants of $T = A\{j \mapsto u\}$ and $T' = A\{j \mapsto v\}$ if $\phi \vDash u = v$:

$$\frac{\Gamma \vdash_{\mathcal{S}} t : A\{j \mapsto u\} \qquad \phi \vDash u = v}{\Gamma \vdash_{\mathcal{S}} t : A\{j \mapsto v\}}\ \mathcal{P}_{\equiv}$$

From a proof π of $[u = v]$ it would then be possible to "lift" a term t from on type to the other using the following constructor (its type is in two lines):

$$\Uparrow\ :\ \Pi u\ v : \mathbb{N}.\ \Pi c : \boldsymbol{\epsilon}(u = v).\ \Pi s : \mathbb{N} \to \boldsymbol{\mathcal{S}}.\ \Pi a : \left(\Pi j : \mathbb{N}.\, \boldsymbol{\epsilon}(u = j) \to \texttt{U}_{(s\ j)}\right).$$
$$\texttt{T}_{(s\ u)}\ (a\ u\ \mathbf{I}) \to \texttt{T}_{(s\ v)}\ (a\ v\ c)$$

Provided $\phi \vDash u = v$ holds and $\boldsymbol{\epsilon}\ (u = v)$ is inhabited with c, this constructor can be used to explicitly cast a term of type $a\ u\ \mathbf{I} \equiv_{\beta\mathcal{R}} [T]$ to the type $a\ v\ c \equiv_{\beta\mathcal{R}} [T']$.

It requires however to define a well-sorted type A where the occurrence of u to substitute is replaced with a fresh level variable j. Since $A\{j \mapsto u\}$ is well-sorted, A is well-sorted in a context extended with the $j = u$ constraint, explaining the type of the a parameter in the definition of $\Uparrow$.

Since handling proofs of equality has to be done explicitly during the translation and is not longer computed during type checking, this encoding is much "deeper" than the previous ones. The translation must explicit all universe-related conversion steps making it highly impractical.

6.4.3 Issues

A first issue with the first three propositions is that they do not support constraints of the shape $i \leq n$. This is however not a problem in the context of the translation of Coq since these constraints are not supported by the Coq tool either. Besides the limitations in complexity and size, these embeddings define a translation of level variables which depends on the set on constraints on these variable which cannot therefore be extended without changing the translation of already defined variables. This makes them non-modular and forbids the successive and incremental addition of global constraints as allowed in Coq. Note that these three embeddings also require non-left-linear rewrite rules, like the one introduced in Lemma 6.2.16, which makes it harder to prove that it preserves the confluence of β. This can however be dealt with by forcing levels to be syntactically "confined" expressions as defined in 4, forbidding interactions with β.

For all of these reasons we didn't further consider these encodings for the translation of Coq's universe polymorphism. However they would be fit for the translation of a Calculus of Constructions with algebraic levels and fixed constraints such as the $\mathrm{CCC}^{\sim}_{\mathcal{L}}$ or the Explicit Polymorphic Extended Calculus of Constructions (EPECC) as defined by Courant [Cou02].

6.4.4 Eluding level conversion

In order to ease the presentation of universe polymorphism encoding in Chapter 7, we chose to actually ignore the issue of equality between algebraic level expressions. Instead we rely exclusively on derivations where only syntactically equal expressions are considered equal. This means for instance that $\mathtt{Type}_{\max(u,u)}$ and $\mathtt{Type}_u$ are two distinct non-convertible sorts but they are subtypes of one another and behave identically .

Herbelin showed that under certain conditions of the type system (see 7.3.1) the conversion of universe levels need only be reflected on level variables. In particular, terms elaborated from typical ambiguity can always be considered *conclusion-algebraic* which means that sorts of non-atomic level expression need only to occur in the ultimate codomain of product types. When translating conclusion-algebraic terms, one only need to identify level variables. In order to reflect that the translations of equal level variables are convertible we will simply forbid sets that allow equal variables. Indeed, whenever $\phi \vDash i = j$,

we can simply substitute i by j (or the other way around) everywhere in the considered definition and remove one of these variables from the set. This is sound by the following lemma on constraints.

Lemma 6.4.8. *Assume $\phi \vDash i = j$. Then $\phi\{i \mapsto j\}$ is a stratifiable set of atomic constraints, $\phi \vDash u \leq v \iff \phi\{i \mapsto j\} \vDash u\{i \mapsto j\} \leq v\{i \mapsto j\}$ and if $\Gamma \vdash_{\mathcal{S}} t : A$, then $\Gamma\{i \mapsto j\} \vdash_{\mathcal{S}} t\{i \mapsto j\} : A\{i \mapsto j\}$.*

Proof. The set $\phi\{i \mapsto j\}$ is still atomic and if $\sigma \vDash_{\mathcal{S}} \phi$, then $\sigma(i) = \sigma(j)$ and therefore $\sigma(u\{i \mapsto j\}) = \sigma(u)$ for all universe expression u, proving the first two points. Besides the function $u \mapsto u\{i \mapsto j\}$ is a morphism of CTS, see Definition 5.3.7 and therefore preserves typing by Lemma 5.3.8. □

Definition 6.4.9. *A stratifiable set of atomic constraints ϕ is* acyclic *iff for all level variables i and j, if $\phi \vDash i = j$, then $i = j$.*

Lemma 6.4.10. *Assume a stratifiable set of atomic constraints ϕ. Then there exists a mapping $\sigma : \mathcal{L} \to \mathcal{L}$ of level variables to level variables compatible with ϕ, $\forall i, \phi \vDash i = \sigma(i)$, such that $\psi = \sigma(\phi)$ is* acyclic.

Besides, if $\phi \vDash u \leq v$, then $\psi \vDash \sigma(u) = \sigma(v)$ and if $\Gamma \vdash_{\mathcal{S}} t : A$, then $\Gamma\sigma \vdash_{\mathcal{S}} t\sigma : A\sigma$.

Proof. For all set of equivalent level variables, $\phi \vDash i_1 = \cdots = i_n$ we pick a representative i_1 and have σ map all other i_k to i_1. We conclude using Lemma 6.4.8. □

Chapter 7

A Universe Polymorphic Calculus of Constructions

We introduce in this chapter our main target system, the Extended Calculus of Constructions further extended with universe polymorphic declarations and definitions. This system is the core logic of the Coq proof assistant which features many more functionalities. We will define and prove correct an embedding of this system into $\lambda\Pi_{\equiv}$ in Chapter 8 and mention practical encoding technique to represent its more advanced functionalities in Chapter 9.

The main idea behind universe polymorphism is to consider symbols which type may refer to some *local universe variables*. Once the well-sortedness of T has been checked in a context of local universe variables, $\Gamma; i \vdash_{\mathcal{S}} T(i) : s(i)$, it should be possible to add a universe polymorphic symbol to the context, $\Gamma,\ x[i] : T_i\ \mathbf{WF}_{\mathcal{S}}$ and use any instance of it later on, $\Gamma,\ x[i] : T_i \vdash_{\mathcal{S}} x_* : T_*$ and $\Gamma,\ x[i] : T_i \vdash_{\mathcal{S}} x_\triangle : T_\triangle$. Universe polymorphism is a convenient way to handle polymorphic symbols in a more general way than already allowed by cumulativity. For instance the polymorphic identity $\mathtt{id} : \Pi A : \mathtt{Type}_0.\ A \to A$ is of type $\mathtt{Type}_1$ which prevents it to be used as the first argument of $\mathtt{id}$ itself: $\mathtt{id}\ (\Pi A : \mathtt{Type}_0.\ A \to A)\ \mathtt{id}$ is not well-typed. Using universe polymorphism, it is however possible to have

$$\mathtt{id}[i] : \Pi A : \mathtt{Type}_i.\ A \to A \vdash_{\mathcal{S}} \mathtt{id}_1\ (\Pi A : \mathtt{Type}_0.\ A \to A)\ \mathtt{id}_0 : \Pi A : \mathtt{Type}_0.\ A \to A$$

Universe variables may not be freely quantified over and moved in the context like the usual variables. Instead we enforce a *prenex* polymorphism: local universe variables and their constraints are declared *before* the type of the polymorphic symbol. This requires to separate our context into a *signature* defining potentially universe polymorphic symbols and a *context* containing monomorphic variables. Universe variables $\bar{i}$ and the associated set of constraints ϕ are both declared in between: $\Sigma; \bar{i}/\phi; \Gamma$.

Naturally there are many other things to consider in order to properly define a usable universe polymorphic system that remains consistent. In particular, we would like to be able to instantiate universe polymorphic symbols with algebraic levels containing other local universe variables. Conversion of universe expression is highly dependent on

constraints and therefore now depends on the context. Terms other than sorts may now contain level expressions.

It is also quite convenient to allow to extend the signature Σ with the universe polymorphic *definition* of a symbol c. Defined symbols are provided a well-sorted type T as well as a *body*, or *definition*, which is a term t of type T. Instances of the symbol c are considered as convertible with the corresponding instance of its provided definition which may refer to the local universe variables. Assuming $\Sigma; (\mathsf{c}[i] := t(i) : T(i))$ $\mathbf{WF}_{\mathcal{S}}$ then the definition may later be instantiated with other algebraic universe expressions: $\Sigma; \Gamma \vdash_{\mathcal{S}} \mathsf{c}[u] : T\{i \mapsto u\}$.

Even though the system we describe below is meant to represent the Calculus of Constructions with (invariant) universe polymorphic definitions and declarations as implemented in the Coq tool, we work a bit differently from the standard presentation of this system. Instead of inferring ever-growing sets of constraints while building derivation trees and then checking their stratifiability, we work backward and assume an already inferred set of stratifiable constraints sufficiently large to entail the well-typedness of the definition.

Even though it is often implicit, its specificities voluntarily being hidden away from the user, we make universe polymorphism explicit in our presentation. Besides we chose to present it in a more declarative way which is closer to the Explicit Polymorphic Extended Calculus of Constructions (EPECC) as defined by Courant [Cou02]. Polymorphic universe levels and constraints are not inferred, elaborated and minimized as in Coq but rather fully declared beforehand and it is then checked that the declared constraints are sufficient to *entail* all the required properties on universe levels to check the well-typedness of the following declaration or definition. Our version is declarative while Coq's inference algorithm is a search of constraints and variables. It can be seen as a "re-checking" system for the constraint inference system of Coq.

7.1 Definition

7.1.1 Syntax

We extends the CTS as defined in previous chapter with *definitions* and *declarations* in a *signature* and locally bounded *universe variables* that can be abstracted and constrained before definitions and declarations.

Definition 7.1.1 ($\mathrm{CC}^{\forall}_{\omega}$ Syntax)**.** *The syntax of* $\mathrm{CC}^{\forall}_{\omega}$ *is the following:*

$$
\begin{array}{rrcl}
\textit{(Variable)} & x, y & \in & \mathcal{X} \\
\textit{(Symbol)} & \mathtt{c} & \in & \mathcal{F} \\
\textit{(Level)} & n & \in & \mathbb{N} \\
\textit{(Level Variable)} & i, j & \in & \mathcal{L} \\
\textit{(Level Expression)} & u, v & := & n \mid i \mid u + n \mid \max(u, v) \\
\textit{(Constraint)} & \phi, \psi & := & u \leq v \mid \phi \wedge \psi \\
\textit{(Sort)} & s & \in & \mathtt{Prop} \mid \mathtt{Type}_u \\
\textit{(Term)} & t, \tau, A, B & := & x \mid s \mid A\ B \mid \lambda x : A.t \mid \Pi x : A.B \mid \mathtt{c}_{\overline{u}} \\
\textit{(Signature)} & \Sigma & := & \varnothing \mid \Sigma, \mathtt{c}[\overline{i}/\phi] : \tau \mid \Sigma, \mathtt{c}[\overline{i}/\phi] := t : \tau \quad (\mathtt{c} \notin \Sigma) \\
\textit{(Context)} & \Gamma, \Delta & := & \varnothing \mid \Gamma, x : t \quad (x \notin \Gamma) \\
\textit{(Signature WF Judgment)} & & := & \Sigma\ \mathbf{WF}_{\mathcal{S}} \\
\textit{(Context WF Judgment)} & & := & \Sigma \vdash_{\mathcal{S}} \overline{i}/\phi; \Gamma\ \mathbf{WF}_{\mathcal{S}} \\
\textit{(Typing Judgment)} & & := & \Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : A \\
\textit{(Subtyping Judgment)} & & := & \phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B \\
\textit{(CTS Judgment)} & & := & \phi \vdash_{\mathcal{S}} \mathcal{E}(\overline{s}) \qquad \mathcal{E} \in \{\mathcal{A}, \mathcal{R}, \mathcal{C}\} \\
\textit{(Constraint Judgment)} & & := & \phi \vdash_{\mathcal{S}} \psi
\end{array}
$$

The Coq system also supports constraints of the shape $u < v$ and $u = v$ which we respectively represent as $u + 1 \leq v$ and ignore since identical universe level variables can be identified.

7.1.2 Level expressions

When considering a typing judgment, $\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : A$, the level variables $\overline{i}$ are said to be *bound levels*. Of course not all level variables in ϕ, t or A have to be bound.

Definition 7.1.2. *We define $\mathcal{LV}ar(t) := \mathcal{V}ar(t) \cap \mathcal{L}$ the set of level variables occurring in t and $\mathcal{LV}ar(\Sigma)$ the set of level variables* global *in the signature Σ:*

$$
\begin{array}{ll}
\mathcal{LV}ar(\varnothing) & := \varnothing \\
\mathcal{LV}ar(\Sigma, \mathtt{c}[\overline{i}/\phi] : \tau) & := \mathcal{LV}ar(\Sigma) \cup (\mathcal{LV}ar(\phi) \cup \mathcal{LV}ar(\tau)) \setminus \overline{i} \\
\mathcal{LV}ar(\Sigma, \mathtt{c}[\overline{i}/\phi] := t : \tau) & := \mathcal{LV}ar(\Sigma) \cup (\mathcal{LV}ar(\phi) \cup \mathcal{LV}ar(t) \cup \mathcal{LV}ar(\tau)) \setminus \overline{i}
\end{array}
$$

Global level variables are not quantified over in declarations and definitions of symbols and are not substituted in their occurrences. The sets of local constraints may however freely refer to these levels. In the Coq system they would correspond to *floating universes* to which *global constraints* can be associated.

In order to ease the presentation of this system floating universes and global constraints are essentially ignored. From here on, all universe variables in definitions and declarations are assumed to be locally bounded. This assumption does not change the expressiveness of our system. Symbols can simply be made universe polymorphic in all of their available global universes by adding them to the set of prenex-quantified universes in their declaration (or definition). Global constraints can be duplicated in all the constraint sets of definitions and declarations in the signature. This would obviously be cumbersome in

practice as any reference to these symbols must now provide the global-now-local universe variables as extra arguments.

Constraints on local universe variables are handled in a different way here than in the work of Harper and Pollack where they are inferred from typical ambiguous terms. Since our objective is to translate well-typed term into a logical framework, we are not concerned here with the techniques allowing to generate a set of stratifiable constraints ϕ sufficient to ensure the well-typedness of the polymorphic definitions or declarations in the signature. Instead we assume the set of constraints is explicitly declared, as in Courant's EPECC, and is sufficiently large to entail all constraints required to check the definitions and declarations of the signature. Instead we rely on a presentation of universe polymorphism closer to Sozeau and Tabareau [ST14] where the type checking inference rules are defined assuming a set of constraints and then a separate elaboration algorithm is provided to build this set in practice.

7.1.3 Conversion

δ reduction

Conversion in $\mathrm{CC}^{\forall}_{\omega}$ is the usual β conversion extended with the δ rewrite rule performing the unfolding of previously defined symbols Since conversion now depends on the signature Σ we write $t \equiv_{\beta\Sigma} u$ to express that $t \underset{\beta\delta}{\twoheadleftarrow\!\!\twoheadrightarrow} u$ holds in the signature Σ.

Definition 7.1.3. *Assume Σ a signature, δ-reduction, written $\underset{\delta}{\longrightarrow}$, is the smallest monotonic relation such that $\forall(\mathtt{c}[\overline{j}/\psi] := t : \tau) \in \Sigma\ ,\ \mathtt{c}_{\overline{u}} \underset{\delta}{\longrightarrow} t\{\overline{j} \mapsto \overline{u}\}$.*

Lemma 7.1.4. $\underset{\beta}{\longrightarrow} \cup \underset{\delta}{\longrightarrow}$ *is confluent.*

Proof. Since symbols cannot be defined more than once in the signature, $\underset{\delta}{\longrightarrow}$ is the rewrite relation generated from a set of left-linear rewrite rules headed with the (pairwise distinct) rewritten symbols. They are therefore non-overlapping. □

In practice, as symbols are defined successively, any definition may only refer to previously defined symbols and the δ relation is therefore also terminating. Recursive definitions are not introduced using this mechanism but rather using fixpoints on inductives types which guarantee their well-foundedness. Fixpoints are ignored in this formal embedding but supported in practice using the representation introduced in Chapter 9.

Universe level conversion

As in most presentations of the Calculus of Constructions with abstract universes the conversion is also extended with a conversion on level expressions:

$$\frac{\phi \vDash u = v}{\mathtt{Type}_u \equiv_\phi \mathtt{Type}_v} \qquad \frac{\phi \vDash \overline{u} = \overline{v}}{\mathtt{c}_{\overline{u}} \equiv_\phi \mathtt{c}_{\overline{v}}}$$

However, as motivated in 6.4.4 and justified in 7.3.1, we chose to ignore this conversion extension in our translation and stick with a β-δ conversion only. We chose however to keep level conversion in our presentation. We write $\equiv_{\beta\Sigma\phi} := (\equiv_\phi \cup \equiv_{\beta\Sigma})^*$.

Subtyping

The rules $\preceq_{\mathrm{r}}$, $\preceq_{\mathcal{C}}$, $\preceq_\pi$ and $\preceq_\equiv$ from Figure 7.1 define a subtyping relation in a set of level constraints: $\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B$. This relation extends the usual conversion with similar properties to the corresponding relation of CTS:

Lemma 7.1.5. *If $\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B$ then either $A \equiv_{\beta\Sigma\phi} B$ or there exists $(s, s') \in \mathcal{C}$ such that $A \equiv_{\beta\Sigma\phi} \Pi x_1 : C_1 \ldots \Pi x_n : C_n . s$ and $B \equiv_{\beta\Sigma\phi} \Pi x_1 : C_1 \ldots \Pi x_n : C_n . s'$.*

Proof. By induction on $\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B$. □

7.1.4 Inference rules

Definition 7.1.6 (Typing in $\mathrm{CC}^\forall_\omega$)**.** *A term t* has type *A in the environment $\Sigma; \bar{i}/\phi; \Gamma$ if the judgment $\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : A$ is derivable using the inference rules of Figure 7.1 and Figure 7.2. A context Γ is* well-formed *in the signature Σ if $\Sigma \vdash_{\mathcal{S}} \bar{i}/\phi; \Gamma$ $\mathbf{WF}_{\mathcal{S}}$ is derivable with the same inference rules. A signature Σ is* well-formed *if the judgment Σ $\mathbf{WF}_{\mathcal{S}}$ is derivable with the same inference rules.*

Just like our presentation of $\lambda\Pi_\equiv$, this presentation differs from the usual definition of Calculi of Constructions. Signature and context well-formedness is not enforced here by the axiom derivation rules so that it is possible to derive $\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : A$ in an ill-formed context or signature. This choice allows a better correspondence between typing derivations and their translation to $\lambda\Pi_\equiv$ as defined later. Most theorems about $\mathrm{CC}^\forall_\omega$ consider exclusively well-formed signatures and contexts. We write $\Sigma \vdash^{\mathrm{WF}}_{\mathcal{S}} \bar{i}/\phi; \Gamma$ $\mathbf{WF}_{\mathcal{S}}$ iff $\Sigma \vdash_{\mathcal{S}} \bar{i}/\phi; \Gamma$ $\mathbf{WF}_{\mathcal{S}}$ and Σ $\mathbf{WF}_{\mathcal{S}}$. We write $\Sigma; \bar{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_{\mathcal{S}} t : A$ iff $\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : A$ and $\Sigma \vdash^{\mathrm{WF}}_{\mathcal{S}} \bar{i}/\phi; \Gamma$ $\mathbf{WF}_{\mathcal{S}}$.

Essentially universe level variables can be prenex-quantified over in definition and declarations and sets of constraints are used to guard them. An instance of a polymorphic definition or declaration is only valid if the substituted guard is consistent. Context may be further enriched with so called "floating universes" represented by a set I of global universes and a set Φ of stratifiable global constraints. The set I can even be dropped as the rules already allow to rely on "free" level variables that are not locally bounded.

Our stripped down version, $\mathrm{CC}^\forall_\omega$, of Coq is very close to the PCUIC system from the MetaCoq project [SBF+20, SAB+20] except it doesn't feature inductive types, letin, or fixpoint constructions. Indeed, our endeavor shares similarities with that of the MetaCoq project which aims at formally specifying, within Coq, the semantics of Coq and its type checking algorithm. It is therefore not surprising that we ended with a similar definition for an easily studied subset of Coq's logic.

7.2 Conservative extensions

Of all the successive extensions studied in Chapter 6, only the extensions from PTS to CTS were not conservative and extended typing, even of terms in the original CoC. An infinite hierarchy of universes, extended with abstract universe variables and constraints all preserve the well-typedness and inhabitability of terms that do not rely on these mechanisms.

Extending the Calculus of Constructions with a signature of universe polymorphic declarations and definitions of symbols is easily shown conservative as well if that signature is empty. However, it is possible to get a stronger result and link judgments derivable in $\mathrm{CC}^{\forall}_{\omega}$ in any well-formed signature to slightly adapted judgments derivable in ECC.

We also point out the extra premise in two of our rewrite rules. This adaptation is merely an artifact to better fit the needs of our derivation tree translation function. We show nonetheless that it is inconspicuous.

7.2.1 Extra premises

Besides the introduction of universe polymorphic declaration and definition of symbols in the signature, our presentation contains altered inference rules for the core logic. The application and subtyping rules both have an extra premise. It is easy to see that this adaptation is sound and does not compromise the consistency of the system. We show that it is also complete and does not make the system any less expressive than the ECC it's based on.

The purpose of this extra premise is to conveniently explicit typing sorts in derivations. It is similar in principle to making subtyping explicit in terms, as done in the "Tarski-style" $\mathrm{PTS}^{\uparrow}$ (see Section 5.2). In our case, however, derivations trees, rather than terms, are annotated since we will be translating them.

Definition 7.2.1 (Explicitly Sorted CTS)**.** *We define* CTS^{e} *as the* CTS *where the application,* $\mathcal{P}_{@}$ *and subtyping* $\mathcal{P}_{\preceq_{\mathcal{C}}}$ *typing rules both have an extra premise:*

$$\frac{\Gamma \vdash_{\mathcal{C}} u : \Pi x : A.\, B \qquad \Gamma \vdash_{\mathcal{C}} A : s \qquad \Gamma \vdash_{\mathcal{C}} t : A}{\Gamma \vdash_{\mathcal{C}} u\ t : B\{x \mapsto t\}}\ \mathcal{P}_{@}$$

$$\frac{\Gamma \vdash_{\mathcal{C}} A : s_A \qquad \Gamma \vdash_{\mathcal{C}} B : s_B \qquad \Gamma \vdash_{\mathcal{C}} t : A \qquad A \preceq_{\mathcal{C}} B}{\Gamma \vdash_{\mathcal{C}} t : B}\ \mathcal{P}_{\preceq_{\mathcal{C}}}$$

Lemma 7.2.2 (Soundness)**.** *If* $\Gamma \vdash_{\mathcal{C}} t : A$ *in* CTS^{e}*, then* $\Gamma \vdash_{\mathcal{C}} t : A$ *in* CTS*.*

Proof. Pruning out the subtrees corresponding to the new premises in the derivation of $\Gamma \vdash_{\mathcal{C}} t : A$ in CTS^{e} yields a derivation of the same judgment in CTS. □

While soundness is straightforward, completeness is a bit trickier.

Lemma 7.2.3 (Substitution)**.** *If* $\Gamma, x : A, \Gamma'$ $\mathbf{WF}_{\mathcal{C}}$*,* $\Gamma \vdash_{\mathcal{C}} t : A$ *and* $\Gamma, x : A, \Gamma' \vdash_{\mathcal{C}} u : B$ *in* CTS^{e}*, then* $\Gamma, \Gamma'\{x \mapsto t\}$ $\mathbf{WF}_{\mathcal{C}}$ *and* $\Gamma, \Gamma'\{x \mapsto t\} \vdash_{\mathcal{C}} u\{x \mapsto t\} : B\{x \mapsto t\}$*.*

Proof. The latter is proven by induction on the derivation of $\Gamma, x : A, \Gamma' \vdash_{\mathcal{C}} u : B$. In all cases, a derivation of $\Gamma, \Gamma'\{x \mapsto t\} \vdash_{\mathcal{C}} u\{x \mapsto t\} : B\{x \mapsto t\}$ can be built from the induction hypothesis on the premises.

- In the $[\mathcal{P}_\Pi]$ and $[\mathcal{P}_\mathcal{S}]$ cases, we need the stability of sorts by substitution: $s\{x \mapsto t\} = s$.
- In the $\mathcal{P}_{\preceq_\mathcal{C}}$ case, we need the stability of subtyping by substitution, which is easily deduced from stability of conversion and sorts: $A \preceq_\mathcal{C} B \Rightarrow A\{x \mapsto t\} \preceq_\mathcal{C} B\{x \mapsto t\}$.
- The $[\mathcal{P}_@]$ and $[\mathcal{P}_\lambda]$ cases are straightforward.
- In the case of variable introduction, $[\mathcal{P}_\mathcal{X}]$:
  - If $u = x$ then $B = A$ and $x \notin \mathcal{FV}ar(A)$, therefore we have $u\{x \mapsto t\} = t$ and $B\{x \mapsto t\} = A$, we conclude since $\Gamma, \Gamma'\{x \mapsto t\} \vdash_\mathcal{C} t : A$ by weakening.
  - If $u = y \neq x$ then either $(y : Y) \in \Gamma$ or $(y : Y) \in \Gamma'$. In both cases, we can conclude by $[\mathcal{P}_\mathcal{X}]$.

$\Gamma, \Gamma'\{x \mapsto t\}$ $\mathbf{WF}_\mathcal{C}$ is then proven by induction on Γ' using the previous result. □

Lemma 7.2.4 (Product Inversion)**.** *In* CTS^e*, assuming* $\Gamma \vdash^{\mathrm{WF}}_\mathcal{C} \Pi x\!:\!A.\,B : s$ *then there exists sorts* $s_A, s_B \in \mathcal{S}$ *such that* $\Gamma \vdash^{\mathrm{WF}}_\mathcal{C} A : s_A$ *and* $\Gamma, x : A \vdash^{\mathrm{WF}}_\mathcal{C} B : s_B$*.*

Proof. By induction on the derivation. The $\mathcal{P}_{\preceq_\mathcal{C}}$ case is done by induction hypothesis and the $[\mathcal{P}_\Pi]$ case yields the result. □

Lemma 7.2.5. *In* CTS^e*, if* $\Gamma \vdash^{\mathrm{WF}}_\mathcal{C} t : A$*, then* $\Gamma \vdash_\mathcal{C} A : s$ *for some sort* s*.*

Proof. By case analysis of the typing derivation. Case $[\mathcal{P}_\mathcal{X}]$ is deduced from Γ $\mathbf{WF}_\mathcal{C}$. Case $[\mathcal{P}_@]$ requires Lemma 7.2.4 to have $\Gamma, x : A \vdash_\mathcal{C} B : s'$ and then Lemma 7.2.3 to conclude $\Gamma \vdash_\mathcal{C} B\{x \mapsto N\} : s'$. The other cases follow either from the premises or the completeness of the specification: $\mathcal{A}$ is total, all sorts are well-typed. □

This property is already known to hold in CTS but because the derivation of $\Gamma \vdash_\mathcal{C} A : s$ is not necessarily structurally smaller than that of $\Gamma \vdash_\mathcal{C} t : A$, we needed it to also hold directly on CTS^e for the induction to go through.

Theorem 7.2.6 (Completeness)**.** *If* $\Gamma \vdash^{\mathrm{WF}}_\mathcal{C} t : A$ *holds in* CTS*, then it holds in* CTS^e*.*

Proof. By induction on the length of Γ and on the derivation of $\Gamma \vdash_\mathcal{C} t : A$. The only problematic cases are the rules with extra premises, $\mathcal{P}_{\preceq_\mathcal{C}}$ and $[\mathcal{P}_@]$. In both cases the type A is inhabited, by induction hypothesis on the corresponding premise, and by Lemma 7.2.5 it is also well-typed with a sort in CTS^e. □

Corollary 7.2.6.1. *If* $\Gamma \vdash^{\mathrm{WF}}_\mathcal{C} t : A$ *holds in* ECC*, then it holds in* ECC^e*.*

7.2.2 Universe polymorphic declarations

Declarations are a conservative extension to the ECC in the sense that

- a universe polymorphic declaration in a well-formed signature, $\mathsf{c}[\overline{j}/\psi] : T) \in \Sigma$ can be seen as a finite representation of the infinite set of all possible instances of that declaration;

- a well-formed signature, can be seen as a finite representation of the infinite context of all possible instances of all of its declarations;
- an instance of a term well-typed in a well-formed signature is well-typed in this infinite context, provided symbol occurrences are replaced with the corresponding instantiated symbol.

Lemma 7.2.7 (Signature Weakening)**.** *Assume* $\Sigma; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_{\mathcal{S}} t : A$ *and* $\Sigma; \Sigma'$ $\mathbf{WF}_{\mathcal{S}}$*, then* $\Sigma, \Sigma'; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_{\mathcal{S}} t : A$*.*

Proof. By induction on the typing and context well-formedness derivations. □

Lemma 7.2.8. *We assume* $\overline{i} \cap \mathcal{LV}ar(\Sigma) = \varnothing$*. If* $s \xrightarrow[\beta\delta]{} t$*, then* $s\{\overline{i} \mapsto \overline{u}\} \xrightarrow[\beta\delta]{} t\{\overline{i} \mapsto \overline{u}\}$ *and if* $s \equiv_{\beta\Sigma\phi} t$*, then* $s\{\overline{i} \mapsto \overline{u}\} \equiv_{\beta\Sigma\phi} t\{\overline{i} \mapsto \overline{u}\}$*.*

Proof. Follows simply by definition, stability of β and confluence of β-δ (Lemma 7.1.4). □

Lemma 7.2.9 (Level Substitution)**.** *Assume two stratifiable sets of atomic constraints* ϕ *and* ψ *such that* $\phi \vDash \psi\{\overline{i} \mapsto \overline{u}\}$ *and* $\Sigma \vdash_{\mathcal{S}} \overline{j}/\phi; \Theta$ $\mathbf{WF}_{\mathcal{S}}$*. Then*

1. *If* $\psi \vDash v \leq w$*, then* $\phi \vDash v\{\overline{i} \mapsto \overline{u}\} \leq w\{\overline{i} \mapsto \overline{u}\}$*;*
2. *If* $\psi \vdash_{\mathcal{S}} \mathcal{E}(\overline{s})$ *for* $\mathcal{E} \in \{\mathcal{A}, \mathcal{R}, \mathcal{C}\}$*, then* $\phi \vdash_{\mathcal{S}} \mathcal{E}(\overline{s}\{\overline{i} \mapsto \overline{u}\})$*;*
3. *If* $\psi \vdash_{\mathcal{S}} A \preceq_{\Sigma\psi} B$*, then* $\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B$*;*
4. *If* $\Sigma; \overline{i}/\psi; \Gamma \vdash_{\mathcal{S}} t : A$*, then* $\Sigma; \overline{j}/\phi; \Theta, \Gamma\{\overline{i} \mapsto \overline{u}\} \vdash_{\mathcal{S}} t\{\overline{i} \mapsto \overline{u}\} : A\{\overline{i} \mapsto \overline{u}\}$*.*
5. *If* $\Sigma \vdash_{\mathcal{S}} \overline{i}/\psi; \Gamma$ $\mathbf{WF}_{\mathcal{S}}$*, then* $\Sigma \vdash_{\mathcal{S}} \overline{j}/\phi; \Theta, \Gamma\{\overline{i} \mapsto \overline{u}\}$ $\mathbf{WF}_{\mathcal{S}}$*.*

Proof. 1. follows directly from Lemma 6.4.8, $\psi\{\overline{i} \mapsto \overline{u}\} \vDash v\{\overline{i} \mapsto \overline{u}\} \leq w\{\overline{i} \mapsto \overline{u}\}$. All other proofs are done by induction on the premise, using the previous points and Lemma 7.2.8 for 3. □

Definition 7.2.10. *Assume a term* t *of the* $\mathrm{CC}^{\forall}_{\omega}$ *syntax and a valuation* $\sigma : \mathcal{LV}ar(t) \to \mathbb{N}$*. We define* Φ_σ *as the function mapping a term* t *to the term* $\sigma(t)$ *in which all symbol occurrences* $\mathsf{c}[\overline{n}]$ *have been replaced with the fresh variable* $c_{\overline{n}}$*. For all term* t*,* $\Phi_\sigma(t)$ *is a term in the* ECC *syntax where the set of variables,* $\mathcal{X}$*, is extended with* $\{c_{\overline{n}} \mid \overline{n} \in \bigcup_k \mathbb{N}^k, \mathsf{c} \in \mathcal{F}\}$*.* Φ_σ *naturally extends to contexts. For* Σ *a signature, we define the following infinite context.*

$$\Phi_\sigma(\Sigma) \;:=\; \{\; c_{\overline{n}} : \Phi_\sigma(T\{\overline{j} \mapsto \overline{n}\}) \;\mid (\mathsf{c}[\overline{j}/\psi] : T) \in \Sigma \;\wedge\; \sigma \vDash \psi\{\overline{j} \mapsto \overline{n}\}\}$$

Theorem 7.2.11. *Assume* Σ *containing no definitions such that* $\Sigma; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_{\mathcal{S}} t : A$ *in* $\mathrm{CC}^{\forall}_{\omega}$*. Then for all valuations* σ *such that* $\sigma \vDash \phi$ *there exists a finite context* $\Delta \subseteq \Phi_\sigma(\Sigma)$ *such that* $\Delta, \Phi_\sigma(\Gamma) \vdash^{\mathrm{WF}}_{\mathcal{C}} \Phi_\sigma(t) : \Phi_\sigma(A)$ *in* ECC*.*

Proof. We prove the result by induction on the length of Σ, Γ and on the derivation of $\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : A$. Since Σ $\mathbf{WF}_{\mathcal{S}}$, for all symbol $c_{\overline{n}} \in \Phi_\sigma(\Gamma)$, we have $(\mathsf{c}[\overline{j}/\psi] : T) \in \Sigma$ and $\Sigma'; \overline{j}/\psi \vdash^{\mathrm{WF}}_{\mathcal{S}} T : s$ for some $\Sigma' \subseteq \Sigma$. By Lemma 6.4.8, we have $\Sigma' \vdash^{\mathrm{WF}}_{\mathcal{S}} T\{\overline{j} \mapsto \overline{n}\} : s\{\overline{j} \mapsto \overline{n}\}$. By induction hypothesis, there is a finite subset $\Delta \subseteq \Phi_\sigma(\Sigma') \subseteq \Phi_\sigma(\Sigma)$ such that Δ $\mathbf{WF}_{\mathcal{C}}$ and $\Phi_\sigma(T\{\overline{j} \mapsto \overline{n}\})$ is well-sorted in Δ. Therefore, $\Delta, c_{\overline{n}} : \Phi_\sigma(T\{\overline{j} \mapsto \overline{n}\})$ $\mathbf{WF}_{\mathcal{C}}$.

This result easily extends to any number of symbol instances $c_{\overline{n}}$, using weakening in CTS. In all generality, for any finite subset $\Delta \subseteq \Phi_\sigma(\Sigma)$, there is a finite superset of it $\Delta' \supseteq \Delta$ such that Δ' $\mathbf{WF}_\mathcal{C}$. We chose a well-formed finite context $\Delta \subseteq \Phi_\sigma(\Sigma)$ such that for all instances $\mathsf{c}_{\overline{n}}$ in Γ, t and A, $c_{\overline{n}} \in \Delta$.

We can now use the induction hypothesis on Γ to prove that $\Delta, \Phi_\sigma(\Gamma)$ $\mathbf{WF}_\mathcal{C}$ since all variables declarations are annotated with a type well-sorted in a strictly shorter context.

Finally, we proceed by induction on the typing derivation of t and prove that we have $\Delta, \Phi_\sigma(\Gamma) \vdash_\mathcal{C} \Phi_\sigma(t) : \Phi_\sigma(A)$ in ECC^e and therefore in ECC. All cases where inference rules are generalizations of those of CTS follow directly by induction hypothesis. Since Σ contains no definition, we can ignore the $\mathcal{P}_{def}$ rule. In the $\mathcal{P}_{decl}$ case, since $\sigma \vDash \phi$ and $\phi \vDash \psi\{\overline{j} \mapsto \overline{u}\}$, then $\sigma \vDash \psi\{\overline{j} \mapsto \sigma(\overline{u})\}$. Therefore $(c_{\sigma(\overline{u})} : \Phi_\sigma(T\{\overline{j} \mapsto \sigma(\overline{u})\})) \in \Delta$ and we can use an instance of $[\mathcal{P}_\mathcal{X}]$ to conclude. □

7.2.3 Universe polymorphic definitions

Lemma 7.2.12 (δ-SR)**.** *If* $\Sigma; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_\mathcal{S} t : A$ *and* $t \xrightarrow[\delta]{} u$*, then* $\Sigma; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_\mathcal{S} u : A$*.*

Proof. By induction on Σ and on the typing derivation. All cases follow directly by induction hypothesis except from the $\mathcal{P}_{def}$ case if the reduction occurs at the head. In that case, since Σ $\mathbf{WF}_\mathcal{S}$, we have $\Sigma'; \overline{j}/\psi; \varnothing \vdash_\mathcal{S} t : \tau$ and using substitution lemma (6.4.8) and signature weakening (7.2.7), we get $\Sigma; \overline{i}/\phi; \Gamma \vdash_\mathcal{S} t\{j \mapsto \overline{u}\} : \tau\{j \mapsto \overline{u}\}$ and conclude. □

Lemma 7.2.13. *If* $\Sigma; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_\mathcal{S} t : \tau$ *and* $\mathsf{c} \lhd t$*. Then* c *is defined or declared in* Σ*.*

Proof. By a simple induction. □

Lemma 7.2.14. *If* Σ $\mathbf{WF}_\mathcal{S}$*, then* δ*-reduction is terminating.*

Proof. A δ-reduction step replaces an occurrence of a symbol c defined in the signature with arbitrary occurrences of other symbols all defined or declared before c in the signature. The vector of all symbol occurrence counts indexed with their position in Σ is therefore a measure decreasing by the reverse-lexicographic ordering. □

Definition 7.2.15. *We write* $t{\downarrow_\delta}$ *the unique normal form of* t *and extend it to contexts. The signature* $\Sigma{\downarrow_\delta}$ *is the signature of its declarations with normalized type annotations:* $\Sigma{\downarrow_\delta} := \{\mathsf{c}[\overline{i}/\phi] : T{\downarrow_\delta} \mid (\mathsf{c}[\overline{i}/\phi] : T) \in \Sigma\}$*.*

Lemma 7.2.16. *If* $\Sigma; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_\mathcal{S} t : A$ *then* $\Sigma; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_\mathcal{S} t{\downarrow_\delta} : A$*.*

Proof. We first prove by induction and using Lemma 7.2.9 that δ-reduction has the subject reduction property. We conclude since it is terminating. □

Lemma 7.2.17. *If* $\Sigma; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_\mathcal{S} t : A$ *then* $\Sigma{\downarrow_\delta}; \overline{i}/\phi; \Gamma{\downarrow_\delta} \vdash^{\mathrm{WF}}_\mathcal{S} t{\downarrow_\delta} : A{\downarrow_\delta}$*.*

Proof. By induction on $\Sigma; \overline{i}/\phi; \Gamma \vdash^{\mathrm{WF}}_\mathcal{S} t : A$. $\mathcal{P}_{decl}$ requires a δ-normal context. $\mathcal{P}_{def}$ is impossible since t is δ-normal. All other cases are straightforward. □

7.3 System restrictions

We define in this section three properties of derivations in the $\mathrm{CC}^{\forall}_{\omega}$. Restricting to judgments and derivations satisfying these properties will be particularly helpful to ensure that the term conversion, $t \equiv_{\beta\Sigma\phi} u$, can be reflected in the translation of the typing derivations of t and u, see Chapter 8. We argue in this section that these properties are safe to consider and not too restrictive. They should in fact be seen as a mere convenience for practical type checking rather than absolutely necessary for the system to be well behaved.

The first property, **(H1)**, restricts the algebraic universe expressions considered in typing judgments. It requires the body of definitions to be atomic and type annotations to be *depth-1 atomically-upper-bounded*. It is not exactly conservative in the sense that not all derivable judgments of $\mathrm{CC}^{\forall}_{\omega}$ directly correspond to an instance of judgment with this property. However, Herbelin showed that any judgment satisfying the *typical ambiguity* convention can be elaborated as a "most general" judgment satisfying this assumption [Her05]. Restricting to conclusion-algebraic type annotations is therefore acceptable when considering term elaborated from typical ambiguity. In particular the typing relation checked by the Coq system satisfies this property.

The second, **(H2)**, restricts the sets of constraints considered to be acyclic. Restricting to acyclic sets of constraints is not restrictive, as shown in 6.4.4, since level variables may simply be identified to remove cycles in a constraint set.

As discussed in Section 6.3, the conversion under constraint, $i \leq j \vDash \max(i, j) = j$, is complicated to embed in $\lambda\Pi_{\equiv}$ in a satisfying way. The **(H1)** restrictions allow to consider a simpler conversion between universe variables only while **(H2)** ensures only equal universe variables are convertible, completely erasing the difficulty. Together they allow not to reflect, in our translation, the conversion of universe level expressions.

The last restriction, **(H3)**, consists in considering only the subset of derivation trees in which subtyping steps immediately follow the application rule on the argument side. This restriction is similar to Assaf's bidirectional typing assumption but a bit less restrictive since subtyping steps are also allowed to occur anywhere else. We require that the argument subtree of any $\mathcal{P}_{@}$ inference rule occurrence has a subtyping $\mathcal{P}_{\preceq}$ rule at the root. This restriction does not change the expressiveness of the system in the sense that any derivable judgment is derivable using a derivation tree satisfying this assumption.

7.3.1 Typical ambiguity and atomic constraints

Herbelin studied [Her05] the particular case of terms that satisfy the typical ambiguity property, meaning that they were elaborated from terms where universe polymorphic sorts are systematically *anonymous* (`Type`) and interpreted as $\texttt{Type}_i$ for some fresh level metavariable i. He showed that in that setting, it is only necessary to infer constraints of the shape $l' \leq l$ or $i+1 \leq l$ for *atomic* universe levels l and l' (either a variable or concrete level) and only necessary to check the validity of so-called *depth-1 atomically-upper-bounded* constraints where the upper-bounding level is atomic and the lower-bounding one is of the shape $\max(n, i_1, \ldots, i_n, j_1 + 1, \ldots, j_k + 1)$.

Definition 7.3.1 (Typical Ambiguity)**.** *A meta-term t has the* typical ambiguity *property, written* $\mathbf{TA}(t)$*, if all level expressions in t are fresh distinct meta-variables of arity 0. For instance if* $\mathtt{Type}_u \lhd t$ *or* $\mathtt{c}_{\overline{v}} \lhd t$*, $u = X \in \mathcal{Z}$ and for all i, $v_i = Y_i \in \mathcal{Z}$.*

If $\mathbf{TA}(t)$*,* $\mathbf{TA}(T)$ *and* $\mathcal{MV}ar(t) \cap \mathcal{MV}ar(T) = \varnothing$ *then* $(\mathtt{c} : T)$ *(resp.* $(\mathtt{c} := t : T)$*) is called an* ambiguous declaration *(resp.* ambiguous definition*).*

A signature is ambiguous if all its definitions and declarations are and the sets of their meta-variables are pairwise disjoints.

Assume a valuation Θ mapping meta-variables to level expressions and a function Φ from symbols to level variables and sets of constraints. The ambiguous signature is elaborated *with Θ and Φ into a signature by replacing all ambiguous declaration* $(\mathtt{c} : T)$ *(resp. ambiguous definition* $(\mathtt{c} := t : T)$*) with the declaration* $(\mathtt{c}[\Phi(\mathtt{c})] : T\Theta)$ *(resp. the definition* $(\mathtt{c}[\Phi(\mathtt{c})] := t\Theta : T\Theta)$*).*

An ambiguous signature Σ is well-formed if there exists a well-formed elaboration of Σ.

In practice, designing proofs with universe polymorphism is often done ambiguously by omitting the level in occurrences of `Type`. It is left to the type checker to infer an affectation of the universe level meta-variables and a sufficiently large set of constraints such that the corresponding elaboration is well-formed. In the case of Coq, this set of constraints is then minimized to eliminate redundancies in the constraints graph.

Definition 7.3.2. *Assume level expressions u, v and a term t.*

- *u is* of depth *n iff it is either atomic, of depth $n-1$,* $\max(v, w)$ *for v, w of depth n or $v + k$ for some v of depth $n - k$.*
- *$u \leq v$ is* depth-n atomically-upper-bounded *iff u is of depth n and v is atomic.*
- *The term t is* non-algebraic*, written* $\mathbf{NA}(t)$*, iff all levels in t are atomic:*
  - *for all occurrences of* $\mathtt{Type}_u \lhd t$ *in the term t, the level u is* atomic*;*
  - *for all occurrences of* $\mathtt{c}_{\overline{u}} \lhd t$ *in the term t, the levels $\overline{u}$ are all* atomic*.*
- *The term t is* depth-n conclusion-algebraic*, written* $\mathbf{CA}^n(t)$*, iff its weak head normal form is either*
  - $\mathtt{Type}_u$ *with u of depth n.*
  - $\Pi x : T.\, T'$ *with* $\mathbf{NA}(T)$ *and* $\mathbf{CA}^n(T')$*.*
  - `Prop` *or an applied symbol or variable (this covers all other cases).*
- *The term t is* conclusion-algebraic*, written* $\mathbf{CA}(t)$*, iff* $\mathbf{CA}^n(t)$ *for some n.*
- *A declaration* $(\mathtt{c}[\overline{i}/\phi] : T)$ *(resp. definition* $(\mathtt{c}[\overline{i}/\phi] := t : T)$*) is* conclusion-algebraic *iff ϕ is atomic and* $\mathbf{CA}^1(T)$ *(resp. and* $\mathbf{NA}(t)$*).*
- *A signature is* conclusion-algebraic*,* $\mathbf{CA}(\Sigma)$*, iff its declarations and definitions are.*

Lemma 7.3.3 (Herbelin)**.** *Assume a conclusion-algebraic well-formed signature Σ. If an ambiguous declaration* $(\mathtt{c} : T)$ *(resp. ambiguous definition* $(\mathtt{c} := t : T)$*) can be elaborated into a declaration (resp. definition) well-typed in Σ, then there exists a conclusion-algebraic elaboration well-typed in Σ.*

Proof. The proof can be found in [Her05]. It relies on the fact that in a conclusion-algebraic signature and context, non-algebraic terms can always be inferred a conclusion algebraic

type. It is therefore only required to consider atomically-upper-bounded constraints $u \leq l$ which are equivalent to a set of atomic constraints. □

Corollary 7.3.3.1. *A well-formed ambiguous signature can be elaborated into a conclusion-algebraic well-formed signature.*

Example 1: Consider the following signature Σ:

$$\begin{array}{lcll} \mathtt{f}[i/\varnothing] & := & \lambda g : \mathtt{Type}_{i+1} \to \mathtt{Type}_{i+2}.\, (\Pi A : \mathtt{Type}_i.\, g\ A) & : & (\mathtt{Type}_{i+1} \to \mathtt{Type}_{i+2}) \to \mathtt{Type}_{i+2} \\ \mathtt{t}[j,k/k \leq j] & := & \mathtt{f}[\max(j,k)]\ (\lambda A : \mathtt{Type}_{j+1}.\, A \to \mathtt{Type}_{k+1}) & : & \mathtt{Type}_{\max(j,k+1)+2} \end{array}$$

We check that Σ is well-formed. Since $\vdash_{\mathcal{S}} (\mathtt{Type}_{i+1} \to \mathtt{Type}_{i+2}) \to \mathtt{Type}_{i+2} : \mathtt{Type}_{i+3}$, `f` is a well-formed definition. We check that the application in the definition of `t` is well-typed. The argument, $\lambda A : \mathtt{Type}_{j+1}.\, A \to \mathtt{Type}_{k+1}$ can be inferred the product type $\mathtt{Type}_{j+1} \to \mathtt{Type}_{\max(j+1,k+2)}$. Its domain, $\mathtt{Type}_{j+1}$, is convertible with the domain of the expected product type, $\mathtt{Type}_{\max(j,k)+1}$, since $k \leq j \vDash j+1 = \max(j,k)+1$. Its codomain is a subtype of the expected product type since $k \leq j \vDash \max(j+1, k+2) \leq \max(j,k)+2$. Finally the inferred type for the definition's body is a subtype of the declared type of `t`, since $k \leq j \vDash \max(j,k)+2 \leq \max(j,k+1)+2$. We conclude that Σ is well-formed.

However the corresponding typical ambiguous signature

$$\begin{array}{lcll} \mathtt{f} & := & \lambda g : \mathtt{Type}_{X_1} \to \mathtt{Type}_{X_2}.\, (\Pi A : \mathtt{Type}_{X_3}.\, g\ A) & : & (\mathtt{Type}_{X_4} \to \mathtt{Type}_{X_5}) \to \mathtt{Type}_{X_6} \\ \mathtt{t} & := & \mathtt{f}[\overline{Y_1}]\ (\lambda A : \mathtt{Type}_{Y_2}.\, A \to \mathtt{Type}_{Y_3}) & : & \mathtt{Type}_{Y_4} \end{array}$$

can be elaborated into the more general and better behaved

$$\begin{array}{lcll} \mathtt{f}[i,j,k/k \leq i] & := & \lambda g : \mathtt{Type}_i \to \mathtt{Type}_j.\, (\Pi A : \mathtt{Type}_k.\, g\ A) & : & (\mathtt{Type}_i \to \mathtt{Type}_j) \to \mathtt{Type}_{\max(k,j)} \\ \mathtt{t}[i,j,k,l/i \leq j, l < j] & := & \mathtt{f}[i,j,k]\ (\lambda A : \mathtt{Type}_i.\, A \to \mathtt{Type}_l) & : & \mathtt{Type}_{\max(k,j)} \end{array}$$

All constraints are atomic, all definitions are non-algebraic and all type annotations are conclusion-algebraic. In practice Coq would actually go one step further and consider only atomic type annotations using an extra polymorphic level for the ultimate codomain together with extra atomic constraints:

$$\mathtt{f}[i,j,k,l/k \leq i, k \leq l, j \leq l] \ := \ \ldots \ : \ (\mathtt{Type}_i \to \mathtt{Type}_j) \to \mathtt{Type}_l$$

Note however that symbol declarations can be seen as axioms of a development and therefore forcing a more general version of these axioms weakens the development.

Definition 7.3.4. *A typing judgment $\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : A$ such that ϕ atomic, $\mathbf{NA}(t)$ and $\mathbf{CA}(A)$ is said to satisfy the hypothesis **(H1)**. We extend it to derivation trees such that the conclusion of all subtrees satisfy **(H1)**.*

7.3.2 Acyclic constraints

As justified with Lemma 6.4.10, if only sets of atomic constraints are considered, it is possible to identify universe variables so that these constraints are also acyclic: if $\phi \vDash i = j$, then $i = j$.

Definition 7.3.5. *Judgments and derivations satisfy **(H2)** iff all their sets of constraints are acyclic.*

Lemma 7.3.6. *If ϕ stratifiable and $A \equiv_{\beta\Sigma\phi} B$, then $A \underset{\beta\delta}{\longrightarrow\!\!\!\!\!\rightarrow} A' \equiv_{\phi} B' \underset{\beta\delta}{\longleftarrow\!\!\!\!\!\leftarrow} B$.*

Proof. By confluence of $\underset{\beta\delta}{\longrightarrow}$ and sub-commutation of sort conversion. The proof is similar to that of Lemma 6.2.7. □

Lemma 7.3.7. *If $\mathbf{NA}(t)$ and $t \underset{\beta\delta}{\longrightarrow} u$, then $\mathbf{NA}(u)$. If $\mathbf{CA}(t)$ and $t \underset{\beta\delta}{\longrightarrow} u$, then $\mathbf{CA}(u)$.*

Proof. By definition of **NA**, **CA** and weak-head normal forms. □

Lemma 7.3.8. *Assume ϕ stratifiable, atomic and acyclic.*

- *If $\mathbf{NA}(A)$, $\mathbf{NA}(B)$ and $A \equiv_{\phi} B$, then $A = B$.*
- *If $\mathbf{NA}(A)$, $\mathbf{NA}(B)$ and $A \equiv_{\beta\Sigma\phi} B$, then $A \underset{\beta\delta}{\longrightarrow\!\!\!\!\!\rightarrow} \underset{\beta\delta}{\longleftarrow\!\!\!\!\!\leftarrow} B$.*
- *If $\mathbf{CA}(A)$, $\mathbf{CA}(B)$ and $A \preceq_{\Sigma\phi} B$, then either $A \underset{\beta\delta}{\longrightarrow\!\!\!\!\!\rightarrow} \underset{\beta\delta}{\longleftarrow\!\!\!\!\!\leftarrow} B$ or there exists s, s' such that $\phi \vdash_{\mathcal{S}} \mathcal{C}(s, s')$, $A \underset{\beta\delta}{\longrightarrow\!\!\!\!\!\rightarrow} \Pi x_1 : C_1 . \ldots \Pi x_n : C_n . s$ and $B \underset{\beta\delta}{\longrightarrow\!\!\!\!\!\rightarrow} \Pi x_1 : C_1 . \ldots \Pi x_n : C_n . s'$.*

Proof. The first point is by induction on A and B. The second follows using Lemma 7.3.6 and Lemma 7.3.7. The last point relies on Lemma 7.1.5 and the definition of **CA**: all domains must be **NA** and convertible and the ultimate co-domains must be sorts in the cumulative relation. □

7.3.3 Systematic subtyping above argument

In the context of derivation tree translation where subtyping must be explicitly annotated and yet term conversion must be reflected, we need to have some control over the positions where subtyping may occur in a derivation. Reflecting the conversion can be done using rewrite rules, however the elimination of so-called "identity casts", use of subtyping from a type A to A itself, requires a non-linear rewrite rule. It is possible to spare this non-linear rewrite rule in the encoding by relying instead on constraints of the considered derivation trees.

Definition 7.3.9. *A derivation satisfies **(H3)** iff all occurrences of the $\mathcal{P}_{@}$ rule is immediately preceded by an occurrence of the $\mathcal{P}_{\preceq}$ in the typing derivation of the argument.*

Bidirectional typing describes typing derivations where subtyping is not only required at the root of arguments but also forbidden anywhere else in the term. In Assaf's translation, although it was not explicitly stated as a derivation of tree, bidirectional typing was

required to be complete. As shown by Lasson [Las12], bidirectional typing is complete for any CTS full ($\mathcal{R}$ is a total relation) and satisfying the local minimum property and well-foundedness of cumulativity, the last two ensuring the existence of principal types. In particular it is complete for all systems introduced in Chapter 6.

Our relaxed variant of bidirectional typing allows subtyping to be used, for instance, in chain or anywhere else in the term. It also has the property to be cut-eliminated into a derivation satisfying the same property which is not the case for bidirectional typing.

Example 2: Assume $\Gamma = g : \mathtt{Type}_0 \to \mathtt{Type}_0, x : \mathtt{Type}_0$ and consider the three following derivations.

$$\dfrac{\dfrac{\dfrac{\dots \quad \mathtt{Type}_0 \preceq_{\Sigma\phi} \mathtt{Type}_0}{\dots \quad \vdash_{\mathcal{S}} x : \mathtt{Type}_0}}{\vdash_{\mathcal{S}} f\ x : \mathtt{Type}_1}}{\vdash_{\mathcal{S}} \lambda f.\, f\ x : (\mathtt{Type}_0 \to \mathtt{Type}_1) \to \mathtt{Type}_1} \qquad \dfrac{\dots \quad \mathtt{Type}_0 \to \mathtt{Type}_0 \preceq_{\Sigma\phi} \mathtt{Type}_0 \to \mathtt{Type}_1}{\vdash_{\mathcal{S}} g : \mathtt{Type}_0 \to \mathtt{Type}_1}}{\Gamma \vdash_{\mathcal{S}} (\lambda f : \mathtt{Type}_0 \to \mathtt{Type}_1.\, f\ x)\ g : \mathtt{Type}_1}$$

$$\dfrac{\dfrac{\mathtt{Type}_0 \to \mathtt{Type}_0 \preceq_{\Sigma\phi} \mathtt{Type}_0 \to \mathtt{Type}_1}{\vdash_{\mathcal{S}} g : \mathtt{Type}_0 \to \mathtt{Type}_1} \qquad \dfrac{\mathtt{Type}_0 \preceq_{\Sigma\phi} \mathtt{Type}_0}{\vdash_{\mathcal{S}} x : \mathtt{Type}_0}}{\Gamma \vdash_{\mathcal{S}} g\ x : \mathtt{Type}_1}$$

$$\dfrac{\dfrac{\vdash_{\mathcal{S}} g : \mathtt{Type}_0 \to \mathtt{Type}_0 \qquad \vdash_{\mathcal{S}} x : \mathtt{Type}_0}{\vdash_{\mathcal{S}} g\ x : \mathtt{Type}_0}}{\Gamma \vdash_{\mathcal{S}} g\ x : \mathtt{Type}_1}$$

The first one is cut-eliminated into the second. Both satisfy **(H3)** since terms in argument positions are subtyped at the root: for the first one, g and x, and only x for the second. Only the first one is a valid bidirectional derivation: subtyping is used exclusively to type arguments. The last one does not satisfy **(H3)** since subtyping is not used to type x.

Lemma 7.3.10. *The subset of derivations satisfying **(H3)** is* complete*: it allows to derive all derivable judgments.*

Proof. We show by induction that any derivation π can be transformed to a derivation with the same conclusion but satisfying **(H3)**. All cases are directly by induction hypothesis except for $\mathcal{P}_@$ in which case we simply need to insert an identity $\mathcal{P}_\preceq$ instance with $A = B$. Note that this proof is made easy by the choice of the extra premise in the $\mathcal{P}_@$ rule, see 7.2.1, which can be directly used to build the $\mathcal{P}_\preceq$ instance. □

SIGNATURE WELL-FORMEDNESS

$$\frac{}{\varnothing\ \mathbf{WF}_{\mathcal{S}}}\ \mathcal{S}^{\mathbf{WF}}_{\varnothing} \qquad \frac{\Sigma\ \mathbf{WF}_{\mathcal{S}} \qquad \Sigma \vdash_{\mathcal{S}} \overline{i}/\phi; \varnothing\ \mathbf{WF}_{\mathcal{S}} \qquad \Sigma; \overline{i}/\phi; \varnothing \vdash_{\mathcal{S}} \tau : s}{\Sigma, (\mathsf{c}[\overline{i}/\phi] : \tau)\ \mathbf{WF}_{\mathcal{S}}}\ \mathcal{S}^{\mathbf{WF}}_{decl}$$

$$\frac{\Sigma\ \mathbf{WF}_{\mathcal{S}} \qquad \Sigma; \overline{i}/\phi; \varnothing \vdash_{\mathcal{S}} t : \tau \qquad \Sigma \vdash_{\mathcal{S}} \overline{i}/\phi; \varnothing\ \mathbf{WF}_{\mathcal{S}} \qquad \Sigma; \overline{i}/\phi; \varnothing \vdash_{\mathcal{S}} \tau : s}{\Sigma, (\mathsf{c}[\overline{i}/\phi] := t : \tau)\ \mathbf{WF}_{\mathcal{S}}}\ \mathcal{S}^{\mathbf{WF}}_{def}$$

CONTEXT WELL-FORMEDNESS

$$\frac{\phi \text{ atomic} \qquad \vDash \phi}{\Sigma \vdash_{\mathcal{S}} \overline{i}/\phi; \varnothing\ \mathbf{WF}_{\mathcal{S}}}\ \mathcal{P}^{\mathbf{WF}}_{\varnothing} \qquad \frac{\Sigma \vdash_{\mathcal{S}} \overline{i}/\phi; \Gamma\ \mathbf{WF}_{\mathcal{S}} \qquad \Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s}{\Sigma \vdash_{\mathcal{S}} \overline{i}/\phi; \Gamma, x : A\ \mathbf{WF}_{\mathcal{S}}}\ \mathcal{P}^{\mathbf{WF}}_{\mathcal{X}}$$

TYPING

$$\frac{\phi \vdash_{\mathcal{S}} \mathcal{A}(s_1, s_2)}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} s_1 : s_2}\ \mathcal{P}_{\mathcal{S}} \qquad \frac{(x : A) \in \Gamma}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} x : A}\ \mathcal{P}_{\mathcal{X}}$$

$$\frac{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s \qquad \Sigma; \overline{i}/\phi; \Gamma, x : A \vdash_{\mathcal{S}} t : B}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} \lambda x : A.\, t : \Pi x : A.\, B}\ \mathcal{P}_{\lambda}$$

$$\frac{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} M : \Pi x : A.\, B \qquad \Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s \qquad \Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} N : A}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} M\ N : B\{x \mapsto N\}}\ \mathcal{P}_{@}$$

$$\frac{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s_1 \qquad \Sigma; \overline{i}/\phi; \Gamma, x : A \vdash_{\mathcal{S}} B : s_2 \qquad \phi \vdash_{\mathcal{S}} \mathcal{R}(s_1, s_2, s_3)}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} \Pi x : A.\, B : s_3}\ \mathcal{P}_{\Pi}$$

$$\frac{\begin{array}{l}\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s_A \\ \Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} B : s_B \end{array} \qquad \Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} M : A \qquad \phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} M : B}\ \mathcal{P}_{\preceq}$$

$$\frac{}{\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} A}\ \preceq_{\mathrm{r}} \qquad \frac{\phi \vdash_{\mathcal{S}} \mathcal{C}(s, s')}{\phi \vdash_{\mathcal{S}} s \preceq_{\Sigma\phi} s'}\ \preceq_{\mathcal{C}} \qquad \frac{\phi \vdash_{\mathcal{S}} B \preceq_{\Sigma\phi} B'}{\phi \vdash_{\mathcal{S}} \Pi x : A.\, B \preceq_{\Sigma\phi} \Pi x : A.\, B'}\ \preceq_{\pi}$$

$$\frac{A \equiv_{\beta\Sigma\phi} A' \qquad \phi \vdash_{\mathcal{S}} A' \preceq_{\Sigma\phi} B' \qquad B' \equiv_{\beta\Sigma\phi} B}{\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B}\ \preceq_{\equiv}$$

$$\frac{(\mathsf{c}[\overline{j}/\psi] : \tau) \in \Sigma \qquad |\overline{u}| = |\overline{j}| \qquad \phi \vDash \psi\{\overline{j} \mapsto \overline{u}\}}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} \mathsf{c}_{\overline{u}} : \tau\{\overline{j} \mapsto \overline{u}\}}\ \mathcal{P}_{decl}$$

$$\frac{(\mathsf{c}[\overline{j}/\psi] := t : \tau) \in \Sigma \qquad |\overline{u}| = |\overline{j}| \qquad \phi \vDash \psi\{\overline{j} \mapsto \overline{u}\}}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} \mathsf{c}_{\overline{u}} : \tau\{\overline{j} \mapsto \overline{u}\}}\ \mathcal{P}_{def}$$

Figure 7.1: Typing rules for $\mathrm{CC}^{\forall}_{\omega}$

$$\frac{\phi \vDash 1 \leq u}{\phi \vdash_{\mathcal{S}} \mathcal{A}(\mathtt{Prop}, \mathtt{Type}_u)}\ \mathcal{A}_{\mathtt{Prop}} \qquad \frac{\phi \vDash u+1 \leq v}{\phi \vdash_{\mathcal{S}} \mathcal{A}(\mathtt{Type}_u, \mathtt{Type}_v)}\ \mathcal{A}_{\mathtt{Type}}$$

$$\frac{}{\phi \vdash_{\mathcal{S}} \mathcal{R}(\mathtt{Type}_u, \mathtt{Prop}, \mathtt{Prop})}\ \mathcal{R}^{\textsc{Cod}}_{\mathtt{Prop}} \qquad \frac{\phi \vDash u \leq v}{\phi \vdash_{\mathcal{S}} \mathcal{R}(\mathtt{Prop}, \mathtt{Type}_u, \mathtt{Type}_v)}\ \mathcal{R}^{\textsc{Dom}}_{\mathtt{Prop}}$$

$$\frac{\phi \vDash (u \leq w) \wedge (v \leq w)}{\phi \vdash_{\mathcal{S}} \mathcal{R}(\mathtt{Type}_u, \mathtt{Type}_v, \mathtt{Type}_w)}\ \mathcal{R}_{\mathtt{Type}}$$

$$\frac{}{\phi \vdash_{\mathcal{S}} \mathcal{C}(\mathtt{Prop}, s)}\ \mathcal{C}_{\mathtt{Prop}} \qquad \frac{\phi \vDash u \leq v}{\phi \vdash_{\mathcal{S}} \mathcal{C}(\mathtt{Type}_u, \mathtt{Type}_v)}\ \mathcal{C}_{\mathtt{Type}}$$

Figure 7.2: CTS derivation rules for $\mathrm{CC}^{\forall}_{\omega}$

Chapter 8

Embedding Universe Polymorphism in the lambda-Pi-calculus modulo

Defining a translation from one system into $\lambda\Pi_{\equiv}$ is naturally done by means of a well-defined computable function mapping the terms and proofs of our target system to terms of the $\lambda\Pi_{\equiv}$.

The two main properties of a translation function are *correctness* and *conservativity*, also called *completeness*. Correctness simply states that the translation of terms well-typed in the original system remains well-typed in $\lambda\Pi_{\equiv}$ in an encoding signature. Conservativity is a bit more subtle since not all terms well-typed in $\lambda\Pi_{\equiv}$ correspond to a well-typed term of the original system. It rather states that if the translation of a well-sorted type is inhabited in $\lambda\Pi_{\equiv}$, then the type is also inhabited in the original system. Since both properties exclusively mention terms well-typed in the original system, the translation function needs only be defined on these well-typed terms. It is worth noting that conservativity does not require inhabitants of $\lambda\Pi_{\equiv}$ translations to be the translation of inhabitants in the original system.

Because the notion of well-typedness is often relative to a context, it is not unusual to actually define the translation as a function parameterized with a context, as done in Chapter 5. When considering type systems without the uniqueness of type property, translations mechanisms can go a step further and define translations that also depend on the expected type for the translated term (written $[t]_{\Gamma \vdash A}$ in [Ass15b] or $[\Gamma \vdash t : A]$ in [Thi20]). Properties of the translation function have to be proven by induction on the term using inversion lemmas and particular properties to link the shape of terms with its possible typing derivations. For instance, to prove the correctness of its CTS translation function, Assaf relies on *principal types* to force a syntax-oriented set of typing rules called *minimal* (or *bi-directional*) typing. Not all specifications allow this, although Lasson showed [Las12] that it does hold for specifications with the so-called *local minimum* property together with well-foundedness of the cumulativity relation. Most useful systems satisfy these properties, including the Calculus of Constructions, even with abstract universes as studied

in Chapter 6.

Relying on these properties of the system hides the fact that the translation of a well-typed term is deeply connected with the way this term can be typed. In this chapter we chose to make this relation explicit by defining the translation of well-typed terms of $\mathrm{CC}^{\forall}_{\omega}$ as a function on typing derivations. This means that a term may be translated several ways, even in the same context and with the same expected type, each corresponding to a different way to check this term's type. Admittedly this choice makes the definition of the translation a bit more technical but we argue that it better illustrates the rather straightforward relation between the encoding and the targeted type system (as opposed to its syntax). Each typing rule corresponds to a simple $\lambda\Pi_{\equiv}$ construction relying on subterms for the derivations of its premises. Finally this representation better reflects the way systems are translated in practice, usually implemented as an mere side effect of the typechecking algorithm.

8.1 The encoding signature

We finally define the main object of study of this dissertation: a signature encoding the Calculus of Constructions with universe polymorphism, $\mathrm{CC}^{\forall}_{\omega}$, defined in Chapter 7. The first two parts of this encoding, relative to the universe expressions, have already been introduced and studied in Chapter 6. The encoding of terms and sorts relies on provable predicates for the sort predicates of CTS as done in Section 5.4 as well as for the general subtyping rule. Finally the last part of the encoding is *private*, as introduced in Section 5.5 and allows to rely on *codes* to guarantee a correct conversion despite the subtyping annotations.

Definition 8.1.1. *We define the encoding of* $\mathrm{CC}^{\forall}_{\omega}$ *as the signature* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$ *made of:*

- Σ_{c} *is the* public *signature of level representation defined in Figure 6.2. It defines 8 symbols and 2 rewrite rules allowing to represent levels in* $\mathbb{N}$ *and constraints.*
- Σ_{p} *is the* public *signature of constraint representation defined in Figure 6.3. It defines 8 symbols and 6 rewrite rules sufficient to inhabit, in the context* $[\phi]$*, the type representation of any constraint entailed by* ϕ*.*
- $\Sigma_{\mathcal{S}}$ *is the* public *signature of sorts and constraints representation defined in Figure 8.2. It defines 2 symbols defining sorts and 7 others defining its* CTS *structure. Note that this sub-signature could easily be adapted to represent universe polymorphism in a different* CTS *structure.*
- $\Sigma_{\mathcal{T}}$ *is the* public *signature of typing derivation representation defined in Figure 8.1. It defines 11 constructors allowing to represent typing derivations.*
- Σ_{pri} *is the* private *signature defined in Figure 8.3. It defines 10 private symbols and 16 rewrite rules.*

The public signature, $\Sigma_{\text{pub}} := \Sigma_{\text{c}}, \Sigma_{\text{p}}, \Sigma_{\mathcal{S}}, \Sigma_{\mathcal{T}}$, *contains the definitions of the* public *symbols. We write* $\mathcal{T}_{\text{pub}}$ *the set of public terms, i.e. built with symbols from the public signature only:* $\mathcal{T}_{\text{pub}} := \{t \mid \forall \mathtt{c} \in \mathcal{F}, \mathtt{c} \lhd t \Rightarrow \mathtt{c} \in \Sigma_{\text{pub}}\}$.

From now on, all typing and well-formedness judgments in the $\lambda\Pi_{\equiv}$ are implicitly considered in the $\mathcal{D}[\text{CC}^{\forall}_{\omega}]$ signature which may be omitted.

$$
\begin{aligned}
\boldsymbol{\mathcal{S}} &: * \\
\mathtt{U}_{\square} &: \boldsymbol{\mathcal{S}} \to * \\
\mathtt{T}_{\square}\ \square &: \Pi s : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_s \to * \\
\\
\boldsymbol{\mathcal{A}}(\square, \square) &: \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to \mathbb{B} \\
\boldsymbol{\mathcal{R}}(\square, \square, \square) &: \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to \mathbb{B} \\
\boldsymbol{\mathcal{C}}(\square, \square) &: \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to \mathbb{B} \\
\square \subseteq^{\square}_{\square} \square &: \Pi s\ s' : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_s \to \mathtt{U}_{s'} \to \mathbb{B} \\
\mathtt{strefl} &: \Pi s : \boldsymbol{\mathcal{S}}.\, \Pi A : \mathtt{U}_s.\, \boldsymbol{\epsilon}(A \subseteq^{s}_{s} A) \\
\\
\underline{\mathbf{u}}^{\square}_{\square,\square} &: \Pi s\ s' : \boldsymbol{\mathcal{S}}.\, \boldsymbol{\epsilon}\ \boldsymbol{\mathcal{A}}(s, s') \to \mathtt{U}_{s'} \\
\underline{\boldsymbol{\pi}}^{\square}_{\square,\square,\square}\ \square\ \square &: \Pi s_1\ s_2\ s_3 : \boldsymbol{\mathcal{S}}.\, \boldsymbol{\epsilon}\ \boldsymbol{\mathcal{R}}(s_1, s_2, s_3) \to \Pi a : \mathtt{U}_{s_1}.\, (\mathtt{T}_{s_1}\ a \to \mathtt{U}_{s_2}) \to \mathtt{U}_{s_3} \\
\underline{\uparrow}^{\square}_{\square}\ \square\ \square &: \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \Pi a : \mathtt{U}_{s_1}.\, \Pi b : \mathtt{U}_{s_2}.\, \boldsymbol{\epsilon}\ (a \subseteq^{s_2}_{s_1} b) \to \mathtt{T}_{s_1}\ a \to \mathtt{T}_{s_2}\ b
\end{aligned}
$$

Figure 8.1: Public encoding signature (terms): $\Sigma_{\mathcal{T}} \subseteq \Sigma_{\text{pub}}$

Lemma 8.1.2. $\mathcal{D}[\text{CC}^{\forall}_{\omega}]$ *is syntactically well-formed.*

Proof. Easily checked. □

Lemma 8.1.3. *We have* $\underset{\mathcal{D}[\text{CC}^{\forall}_{\omega}]}{\longrightarrow} \cup \underset{\beta}{\longrightarrow}$ *locally confluent.*

Proof. All critical pairs and their closing diagrams are recapitulated below:

$$\texttt{Prop} : \boldsymbol{\mathcal{S}}$$
$$\texttt{Type}_{\square} : \mathbb{N} \to \boldsymbol{\mathcal{S}}$$
$$\boldsymbol{\mathcal{A}}(\texttt{Prop}, \texttt{Type}_U) \longrightarrow \texttt{S}\ 0 \leq U \qquad (\boldsymbol{\mathcal{A}}\text{-P})$$
$$\boldsymbol{\mathcal{A}}(\texttt{Type}_U, \texttt{Type}_V) \longrightarrow \texttt{S}\ U \leq V \qquad (\boldsymbol{\mathcal{A}}\text{-T})$$
$$\boldsymbol{\mathcal{R}}(S, \texttt{Prop}, \texttt{Prop}) \longrightarrow \top \qquad (\boldsymbol{\mathcal{R}}\text{-T-P})$$
$$\boldsymbol{\mathcal{R}}(\texttt{Prop}, \texttt{Type}_U, \texttt{Type}_V) \longrightarrow U \leq V \qquad (\boldsymbol{\mathcal{R}}\text{-P-T})$$
$$\boldsymbol{\mathcal{R}}(\texttt{Type}_U, \texttt{Type}_V, \texttt{Type}_W) \longrightarrow (U \leq W) \wedge (V \leq W) \qquad (\boldsymbol{\mathcal{R}}\text{-T-T})$$
$$\boldsymbol{\mathcal{C}}(\texttt{Prop}, S) \longrightarrow \top \qquad (\boldsymbol{\mathcal{C}}\text{-P})$$
$$\boldsymbol{\mathcal{C}}(\texttt{Type}_U, \texttt{Type}_V) \longrightarrow U \leq V \qquad (\boldsymbol{\mathcal{C}}\text{-T})$$

Figure 8.2: Public encoding signature (sorts): $\Sigma_{\mathcal{S}} \subseteq \Sigma_{\text{pub}}$

Critical pair	Closing diagram
$\xleftarrow[0\text{-}\leq]{};\xrightarrow[\leq\text{-S}]{}$	$\xleftarrow[0\text{-}\leq]{}$
$\xleftarrow[\text{S-}\leq]{};\xrightarrow[\leq\text{-S}]{}$	$\xrightarrow[\leq\text{-S}]{};\xleftarrow[\text{S-}\leq]{}$
$\xleftarrow[\text{S-}\leq]{};\xrightarrow[\text{S-}\leq\text{-S}]{}$	$\xrightarrow[\text{S-}\leq\text{-S}]{};\xleftarrow[\text{S-}\leq]{}$
$\xleftarrow[\leq\text{-S}]{};\xrightarrow[\text{S-}\leq\text{-S}]{}$	$\xrightarrow[\text{S-}\leq\text{-S}]{};\xleftarrow[\leq\text{-S}]{}$
$\xleftarrow[\text{max-}\leq]{};\xrightarrow[\text{max-S}]{}$	$\xrightarrow[\text{S-}\leq]{};\xrightarrow[\text{S-}\leq]{};\xleftarrow[\text{max-}\leq]{};\xleftarrow[\text{S-}\leq]{}$

Critical pair	Closing diagram
$\xleftarrow[\leq\text{-S}]{};\xrightarrow[\text{max-}\leq]{}$	$\xrightarrow[\text{max-}\leq]{};\xleftarrow[\leq\text{-S}]{};\xleftarrow[\leq\text{-S}]{}$
$\xleftarrow[\text{S-}\leq\text{-S}]{};\xrightarrow[\text{max-}\leq]{}$	$\xrightarrow[\text{max-}\leq]{};\xleftarrow[\text{S-}\leq\text{-S}]{};\xleftarrow[\text{S-}\leq\text{-S}]{}$
$\xleftarrow[\text{c-u}]{};\xrightarrow[\text{u-}\lambda]{2}$	$\xleftarrow[\text{c-u}]{};\xleftarrow[\text{c-u}]{};\xleftarrow[\text{c-u}]{};\xleftarrow[\beta]{};\xleftarrow[\text{c-}\lambda]{}$
$\xleftarrow[\text{c-a}]{};\xrightarrow[\text{c-u}]{1}$	$\xrightarrow[\text{u-a}]{};\xrightarrow[\text{c-u}]{};\xrightarrow[\text{c-u}]{};\xrightarrow[\text{c-u}]{}$
$\xleftarrow[\text{c-a}]{};\xrightarrow[\text{c-}\lambda]{1}$	$\xrightarrow[\beta]{};\xleftarrow[\beta]{};\xleftarrow[\text{c}\beta]{}$
$\xleftarrow[\text{u-a}]{};\xrightarrow[\text{u-}\lambda]{}$	$\xrightarrow[\text{c}\beta]{};\xrightarrow[\beta]{};\xleftarrow[\beta]{}$

All orthogonal critical pairs are simple. The diagrams are even decreasing if we chose the following order on labels: $\beta, \text{c}\beta < \text{c-}\lambda, \text{u-a} < \text{c-a}, \text{u-}\lambda$ and for rules operating on level constraints: $\text{S-}\leq, \leq\text{-S}, \text{S-}\leq\text{-S} < \text{max-}\leq < \text{max-S}$. □

All type annotations in $\mathcal{D}[\text{CC}^{\forall}_{\omega}]$ are well-typed without using any of the rewrite rules. It is therefore possible to declare the whole signature of symbols first and only then introduce the rewrite rules. The three type-level rewrite rules enc-T, D-u and D-π are necessary to define some of the other rules.

Type preservation of the system ensures that all instances of the rewrite rules in any context and any signature extension preserves the well-typedness of a term as well as its type. Checking this property can usually be done by means of a type checking algorithm, as is the case here.

Lemma 8.1.4 (Type Preservation)**.** *This system satisfies the type preservation property.*

$$
\begin{array}{rlrl}
\mathbb{C}: & * & \mathtt{D}\ \square: & \mathbb{C} \to * \\
\mathbf{u}_{\square}: & \boldsymbol{\mathcal{S}} \to \mathbb{C} & \mathtt{c}(\square,\square): & \Pi c{:}\mathbb{C}.\,\mathtt{D}\ c \to \mathbb{C} \\
\boldsymbol{\pi}\ \square\ \square: & \mathbb{C} \to (\mathbb{C} \to \mathbb{C}) \to \mathbb{C} & \mathtt{u}(\square,\square): & \Pi c{:}\mathbb{C}.\,\mathbb{C} \to \mathtt{D}\ c \\
\boldsymbol{cL}\ \square: & (\mathbb{C} \to \mathbb{C}) \to \mathbb{C} & \forall: & (\mathbb{C} \to \mathbb{B}) \to \mathbb{B} \\
\boldsymbol{cA}\ \square\ \square: & \mathbb{C} \to \mathbb{C} \to \mathbb{C} & \square \preceq \square: & \mathbb{C} \to \mathbb{C} \to \mathbb{B}
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{T}_s\ t & \longrightarrow & \mathtt{D}\ (\mathtt{c}(\mathbf{u}_s, t)) & \text{(enc-T)} \\
\underline{\mathbf{u}}^{P}_{s,s'} & \longrightarrow & \mathtt{u}(\mathbf{u}_{s'}, \mathbf{u}_s) & \text{(enc-u)} \\
\underline{\boldsymbol{\pi}}^{P}_{s_1,s_2,s_3}\ A\ \lambda x.\,B[x] & \longrightarrow & \mathtt{u}(\mathbf{u}_{s_3}, \boldsymbol{\pi}\ \mathtt{c}(\mathbf{u}_{s_1}, A)\ \lambda x{:}\mathbb{C}.\,\mathtt{c}(\mathbf{u}_{s_2}, B[\mathtt{u}(\mathtt{c}(\mathbf{u}_{s_1}, A), x)])) & \text{(enc-}\pi\text{)} \\
{}^{s_2}_{s_1}\!\underline{\uparrow}^{b}_{a}\ P\ t & \longrightarrow & \mathtt{u}(\mathtt{c}(\mathbf{u}_{s_2}, b), \mathtt{c}(\mathtt{c}(\mathbf{u}_{s_1}, a), t)) & \text{(enc-}\uparrow\text{)} \\
A \subseteq^{s_2}_{s_1} B & \longrightarrow & \mathtt{c}(\mathbf{u}_{s_1}, A) \preceq \mathtt{c}(\mathbf{u}_{s_2}, B) & \text{(ST-d)}
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{D}\ \mathbf{u}_s & \longrightarrow & \mathtt{U}_s & \text{(D-u)} \\
\mathtt{D}\ (\boldsymbol{\pi}\ A\ \lambda x.\,B[x]) & \longrightarrow & \Pi x{:}\mathtt{D}\ A.\,\mathtt{D}\ B[\mathtt{c}(A,x)] & \text{(D-}\pi\text{)}
\end{array}
$$

$$
\begin{array}{rcll}
\mathbf{u}_{s_1} \preceq \mathbf{u}_{s_2} & \longrightarrow & \boldsymbol{\mathcal{C}}(s_1, s_2) & \text{(ST-u)} \\
\boldsymbol{\pi}\ A\ \lambda x.\,B[x] \preceq \boldsymbol{\pi}\ A\ \lambda x.\,B'[x] & \longrightarrow & \forall\ \lambda c{:}\mathbb{C}.\,B[c] \preceq B'[c] & \text{(ST-}\pi\text{)} \\
\forall\ \lambda x.\,P & \longrightarrow & P & \text{(ST-}\forall\text{)}
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{c}(C, \mathtt{u}(C,T)) & \longrightarrow & T & \text{(c-u)} \\
\mathtt{c}(\boldsymbol{\pi}\ A\ \lambda x.\,B[x], \lambda x.\,F[x]) & \longrightarrow & \boldsymbol{cL}\ (\lambda x{:}\mathbb{C}.\,\mathtt{c}(B[x], F[\mathtt{u}(A,x)])) & \text{(c-}\lambda\text{)} \\
\mathtt{u}(\boldsymbol{\pi}\ A\ \lambda x.\,B[x], \boldsymbol{cL}\ \lambda x.\,F[x]) & \longrightarrow & \lambda x{:}\mathtt{D}\ A.\,\mathtt{u}(B[\mathtt{c}(A,x)], F[\mathtt{c}(A,x)]) & \text{(u-}\lambda\text{)} \\
\mathtt{u}(\boldsymbol{\pi}\ A\ \lambda x.\,B[x], T)\ U & \longrightarrow & \mathtt{u}(B[\mathtt{c}(A,U)], \boldsymbol{cA}\ T\ \mathtt{c}(A,U)) & \text{(u-a)} \\
\boldsymbol{cA}\ \mathtt{c}(\boldsymbol{\pi}\ A\ \lambda x.\,B[x], T)\ U & \longrightarrow & \mathtt{c}(B[U], T\ \mathtt{u}(A,U)) & \text{(c-a)} \\
\boldsymbol{cA}\ (\boldsymbol{cL}\ \lambda x.\,T[x])\ U & \longrightarrow & T[U] & \text{(c}\beta\text{)}
\end{array}
$$

Figure 8.3: Private encoding signature: Σ_{pri}

Proof. This property is checked by the Dedukti system. □

No theorem from Part I allows to prove the confluence of rewriting with $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$ together with β. Indeed, the two non-linear rewrite rules, ST-π and c-u, forbid to use the results from Chapter 3. The non-linearity of product subtyping provability, ST-π, could be handled by relying on a product subtyping constructor rather than on computation. This would require to build a more complicated witness (see Lemma 8.3.17) and would make the translation more verbose. It would also be possible to use Theorem 4.4.9 by placing the $\boldsymbol{\epsilon}$ symbol at a layer above all others. The associated syntactical restriction would forbid some terms, such as λ-abstractions of $\boldsymbol{\epsilon}$-headed types, which are not in the image of the translation.

The non-linearity of c-u is trickier. This rule would actually be safe to use in its linearized version, $\mathtt{c}(C, \mathtt{u}(C', T)) \longrightarrow T$, since typing ensures that all well-typed instances of the left-hand side satisfy $C \equiv_{\beta\mathcal{R}} C'$. However, using the linearized version creates non-joinable critical pairs. These critical pairs are necessarily ill-typed but this does not mean they can be ignored them since term rewriting is defined in an untyped setting. While results from Part I are not sufficient yet, they were developed as a first step and in hope to be extended so as to eventually prove the confluence of rewriting with this last encoding rule. We can only *conjecture* for now that our system is, or can be slightly adapted to be, well-behaved.

Conjecture 8.1.5. *The signature $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$ satisfies the product compatibility property, is strongly well-formed and has the uniqueness of type and subject reduction properties:* $\mathbf{PC}(\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}])$, $\mathbf{WF}(\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}])$, $\mathbf{UT}(\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}])$ *and* $\mathbf{SR}(\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}])$.

These key properties are critical for the encoding system to be trusted. In theory they are necessary to prove conservativity (completeness) of our encoding and in practice they ensure decidability of type checking. However we can prove the correctness of the encoding, which essentially states that the translation of well-typed terms produces well typed-terms.

This conjecture that the system is well-behaved, at least on the image of the translation defined in Section 8.2, is supported by several practical evidences. This system was used to translate hundreds of lemmas from several libraries of Coq proofs and to successfully check them using the usual typing algorithm.

8.2 Translation functions

We successively define in this section the several syntactical translation functions of natural numbers, sort expressions, predicates, terms as codes, typing derivations and signatures. A translation of contexts is also provided although it is not implemented in practice but rather used in the proofs of most properties of the translation.

8.2.1 Translation domains

Definition 8.2.1. *We define the following six pairwise disjoint sets of types:*

$$\begin{aligned}
\mathcal{F}_1 &:= \{\boldsymbol{\mathcal{S}}\} & \mathcal{F}_4 &:= \{\boldsymbol{\epsilon}\ t \mid t \in \mathcal{T}\} \mid \Pi x : \mathcal{F}_6.\, \mathcal{F}_4 \\
\mathcal{F}_2 &:= \{\mathbb{B}\} & \mathcal{F}_5 &:= \{\mathbb{C}\} \\
\mathcal{F}_3 &:= \{\mathbb{N}\} & \mathcal{F}_6 &:= \mathtt{T}_s\ t \mid \mathtt{U}_s \mid \mathtt{D}\ t \mid \Pi x : \mathcal{F}_6.\, \mathcal{F}_6
\end{aligned}$$

We write $\mathcal{F}_i(\Gamma) \subseteq \mathcal{F}_i$ the set of terms of $\mathcal{F}_i$ well-typed in Γ and in $\beta^\square$ normal form.

Lemma 8.2.2. *For all i and Γ, $\mathcal{F}_i(\Gamma)$ and $\mathcal{F}_i$ are stable by $\underset{\beta\mathcal{R}}{\longrightarrow}$.*

Proof. As none of the $\boldsymbol{\mathcal{S}}$, $\mathbb{B}$, $\mathbb{N}$, $\boldsymbol{\epsilon}$ and $\mathbb{C}$ symbols have rewrite rules, it is easy to see that $\mathcal{F}_i$ is stable for $i \leq 5$. Assume $t \in \mathcal{F}_6$ such that $t \underset{\beta\mathcal{R}}{\longrightarrow} u$. We prove by induction on t that we have $u \in \mathcal{F}_6$. If $t = \mathtt{T}_s\ c$ or $t = \mathtt{U}_s$ or $t = \mathtt{D}\ c$, then either the reduction occurs in s or c, in which case the reduct is immediately in $\mathcal{F}_6$, or t reduces at the head with either the enc-T, D-u or D-π rules and in all three case, the reduct remains in $\mathcal{F}_6$. Otherwise, t is a product and reduction occurs in either the domain or the codomain subterm. Since both are in $\mathcal{F}_6$, we conclude by induction hypothesis.

The stability of the $\mathcal{F}_i(\Gamma)$ follows from subject reduction. □

Lemma 8.2.3. *For all terms $t \in \mathcal{F}_i(\Gamma)$, we have $\mathcal{D}[\mathrm{CC}^\forall_\omega]; \Gamma \vdash_\mathcal{D} t : *$.*

Proof. Immediate since t is either a product or a fully applied type constructor. □

Lemma 8.2.4. *For all i, context Γ and term t in $\beta^\square$ normal form and well-typed in Γ such that $t \underset{\beta\mathcal{R}}{\longrightarrow} u$ for some $u \in \mathcal{F}_i$, we have $t \in \mathcal{F}_i$.*

Proof. By structural induction on u.

If reduction in t occurs below the head, then, by Lemma 2.2.20, t and u have the same head symbol and arity. If the head symbol of u and t is in the signature, then it must be the case that $t \in \mathcal{F}_i$, otherwise $t = \Pi x : A.\, B \longrightarrow \Pi x : A'.\, B' = u$ and we conclude using the induction hypothesis on A or B.

If $i \leq 5$, no instance of the right-hand side of a rule can be a term of $\mathcal{F}_i$ and β steps at the head would be $\beta^\square$ steps, therefore all cases are covered.

Assuming $i = 6$ and that reduction occurs at the head, we need to consider only the rules which right-hand side can be in $\mathcal{F}_6$. The rules ST-$\forall$, c-u and cβ operate at object level, their well-typed instances cannot be of type $*$. The rules D-u, D-π operate at type level and instances of their right-hand sides are of type $*$ and all in $\mathcal{F}_6$. □

Lemma 8.2.5. *Assuming confluence of the system, for all i and well-typed term t in $\beta^\square$ normal form such that $t \equiv_{\beta\mathcal{R}} u \in \mathcal{F}_i$, we have $t \in \mathcal{F}_i$.*

Proof. By confluence, $t \underset{\beta\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow} v \underset{\beta\mathcal{R}}{\longleftarrow\!\!\!\!\!\leftarrow} u$. By Lemma 8.2.2, $v \in \mathcal{F}_i$. By subject reduction, all reducts in $t \underset{\beta\mathcal{R}}{\longrightarrow\!\!\!\!\!\rightarrow} v$ are well-typed and in $\beta^\square$ normal form. We therefore conclude by induction on the length of this well-typed rewrite sequence using Lemma 8.2.4. □

We define in 8.2.2 a translation from objects of $\mathrm{CC}^{\forall}_{\omega}$ to objects of $\lambda\Pi_{\equiv}$ in the $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$ signature. The translation of an object X is written $[X]$ and the $\beta^{\square}$ normal form of its type depends on the nature of X as summed up in the following table:

$\mathrm{CC}^{\forall}_{\omega}$	$\lambda\Pi_{\equiv}$	Type
Variables	Variables	
Symbols	Symbols	
Levels	Closed terms. Peano repr. with `0` and `S`	$\mathcal{F}_3$
Level variables	Variables	$\mathcal{F}_3$
Level expressions	Terms	$\mathcal{F}_3$
Constraints	Terms	$\mathcal{F}_2$
Sorts	Terms	$\mathcal{F}_1$
Signature WF Derivation	Signature	
Context WF Derivation	Context	
Typing Derivation	Term	$\mathcal{F}_6$
Subtyping Judgment	Term of type $\boldsymbol{\epsilon}\ (A \subseteq^{s_2}_{s_1} B)$	$\mathcal{F}_4$
CTS Judgment $\phi \vdash_{\mathcal{S}} \mathcal{E}(\overline{s})$	Term of type $\boldsymbol{\epsilon}\ \mathcal{E}([\overline{s}])$	$\mathcal{F}_4$
Constraint Judgment $\phi \vdash_{\mathcal{S}} \psi$	Term of type $\boldsymbol{\epsilon}\ [\psi]$	$\mathcal{F}_4$
(*Terms*)	Non-unique *code* representations	$\mathcal{F}_5$

Lemma 8.2.5 ensures that two objects of different nature (term, derivation, sort, ...) are translated to terms which types are not convertible.

Terms, signatures and contexts do not directly have representation in this encoding. However whenever they are well-typed (resp. well-formed), then they are, in a way, represented by all terms encoding the derivations of their well-typedness (resp. well-formedness). This representation is not necessarily unique.

8.2.2 Translation functions

Definition 8.2.6. *The translation of level expressions and constraints is the same as in Definition 6.4.5. It is extended to sorts with* $[\mathtt{Prop}] := \mathtt{Prop}$ *and* $[\mathtt{Type}_u] := \mathtt{Type}_{[u]}$.

We extend constraint translation to CTS *judgments:* $[\mathcal{A}(s_1, s_2)] := \boldsymbol{\epsilon}\ \boldsymbol{\mathcal{A}}([s_1], [s_2])$, $[\mathcal{R}(s_1, s_2, s_3)] := \boldsymbol{\epsilon}\ \boldsymbol{\mathcal{R}}([s_1], [s_2], [s_3])$, *and* $[\mathcal{C}(s_1, s_2)] := \boldsymbol{\epsilon}\ \boldsymbol{\mathcal{C}}([s_1], [s_2])$.

Lemma 8.2.7 (Constraint translation)**.** *Assume universe variables* $\overline{i}$*, a stratifiable set of constraints* ϕ *and sorts* $\overline{s}$ *such that* $\mathcal{LV}ar(\phi) \cup \mathcal{LV}ar(\overline{s}) \subseteq \overline{i}$*. If* $\phi \vDash \mathcal{E}(\overline{s})$ *for* $\mathcal{E} \in \{\mathcal{A}, \mathcal{R}, \mathcal{C}\}$*, then* $[\mathcal{E}(\overline{s})]$ *is inhabited in the signature* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$ *and context* $\overline{i} : \mathbb{N}, [\phi]$

Proof. By definition of the translation and using Theorem 6.4.6 since $\Sigma_{\mathrm{c}}, \Sigma_{\mathrm{p}} \subseteq \mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$. □

Definition 8.2.8. *The translation of the derivation of a* CTS *judgment,* $\phi \vdash_{\mathcal{S}} \mathcal{E}(\overline{s})$*, is defined as any inhabitant of the type* $[\mathcal{E}(\overline{s})]$.

Definition 8.2.9. *The translation of a derivable subtyping judgment,* $\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B$ *is defined inductively on the derivation using the rules of Figure 9.11.*

We will see later that it actually needs only be defined as some inhabitant, it is irrelevant which exactly, of the type $\boldsymbol{\epsilon}\ ([\pi_A] \subseteq^{[s_B]}_{[s_A]} [\pi_B])$ where π_A (resp. π_B) is a derivation typing A (resp. B) with the sort s_A (resp. s_B).

The translation of typing judgment derivations is a bit trickier to explicit in a concise way. As opposed to previous presentations, we chose to define a translation function on the well-typedness derivations of a term rather than on the term itself. Because $\lambda\Pi_{\equiv}$ has the uniqueness of type property while $\mathrm{CC}^{\forall}_{\omega}$ does not, then a unique term t well-typed in $\mathrm{CC}^{\forall}_{\omega}$ must necessarily have several representations depending on its expected type. In order to prove that a translation function defined on well typed terms is correct or even well defined, it is usually necessary to rely on particular properties of the target system such as inversion lemmas. In our presentation, the translation is directly defined on the typing derivations themselves and therefore no such property is required for the definition to be well-defined. Besides, the proof of correctness is done directly by induction on the typing derivation which is more straightforward. However in order to be correct we need to justify that all translations of a well-sorted type A, $\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s$, have the same inhabitants.

Definition 8.2.10 (Typing judgment translation)**.** *The translation of typing derivations is inductively defined and depends on the last rule. The complete translation function is described at Figure 9.8. In the recap below, premises subtrees are represented with π_* symbols.*

$$
\begin{aligned}
[\mathcal{P}_{\mathcal{S}}] &:= \underline{\mathbf{u}}^{[\pi_{\mathcal{A}}]}_{[s_1],[s_2]} \\
[\mathcal{P}_{\mathcal{X}}] &:= x \\
[\mathcal{P}_{\Pi}] &:= \underline{\boldsymbol{\pi}}^{[\pi_{\mathcal{R}}]}_{[s_1],[s_2],[s_3]}\ [\pi_A]\ \ \lambda x : \mathtt{T}_{[s_1]}\ \ [\pi_A].\,[\pi_B] \\
[\mathcal{P}_{\lambda}] &:= \lambda x\ :\ \mathtt{T}_{[s]}\ \ [\pi_A]\ .\ [\pi_t] \\
[\mathcal{P}_{@}] &:= [\pi_M]\ \ [\pi_N] \\
[\mathcal{P}_{decl}] &:= \mathtt{c}\ \ [u_1]\ \ \ldots\ \ [u_n]\ \ [\pi_{\phi_1}]\ \ \ldots\ \ [\pi_{\phi_k}] \\
[\mathcal{P}_{def}] &:= \mathtt{c}\ \ [u_1]\ \ \ldots\ \ [u_n]\ \ [\pi_{\phi_1}]\ \ \ldots\ \ [\pi_{\phi_k}] \\
[\mathcal{P}_{\preceq}] &:= {}^{[s_2]}_{[s_1]}\!\uparrow^{[\pi_B]}_{[\pi_A]}\ \ [\pi]\ \ [\pi_M]
\end{aligned}
$$

Note that this translation if *shallow* in the sense that the translation of (a derivation typing) an abstraction (resp. an application, resp. a variable, resp. a symbol instance) is an abstraction (resp. an application, resp. a variable, resp. an applied symbol). Similarly, derivation of context and signature well-formedness are translated to the corresponding object of $\lambda\Pi_{\equiv}$.

Definition 8.2.11 (Context translation)**.** *The translation of a context well-formedness derivation is defined in Figure 9.10 as a $\lambda\Pi_{\equiv}$ context.*

Definition 8.2.12 (Signature translation)**.** *The translation of a signature well-formedness derivation is defined in Figure 9.9 as a $\lambda\Pi_{\equiv}$ signature.*

Note that all translations rely exclusively on the public signature. Rules and symbols of the private signature are however required to ensure well-typedness.

The translation of derivable subtyping judgments is voluntarily left ambiguous. However this ambiguity does not impact the translation of typing derivation, since all constructors relying on judgment translation are *irrelevant* in this argument.

Lemma 8.2.13 (Predicate Irrelevance)**.** *Whenever they are well-typed, the following terms are pairwise convertible and have the same type in* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$ *(*$\square$ *is any term):*

$$\underline{\mathbf{u}}^{\square}_{s,s'} \equiv_{\beta\mathcal{R}} \underline{\mathbf{u}}^{\square}_{s,s'}$$
$$\underline{\boldsymbol{\pi}}^{\square}_{\square,\square,s}\ A\ B \equiv_{\beta\mathcal{R}} \underline{\boldsymbol{\pi}}^{\square}_{\square,\square,s}\ A\ B$$
$$\prescript{\square}{\square}{\underline{\uparrow}}^{A}_{\square}\ \square\ t \equiv_{\beta\mathcal{R}} \prescript{\square}{\square}{\underline{\uparrow}}^{A}_{\square}\ \square\ t$$

Proof. Types are easily checked identical. Conversion was checked using Dedukti. □

8.2.3 Reflection properties

Lemma 8.2.14. *Whenever they are well-typed, the following terms are (pairwise) convertible and have the same type in* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$ *(*$\square$ *is any term):*

$$\prescript{s'}{\square}{\underline{\uparrow}}^{c}_{b}\ \square\ \left(\prescript{\square}{s}{\underline{\uparrow}}^{b}_{a}\ \square\ t\right) \equiv_{\beta\mathcal{R}} \prescript{s'}{s}{\underline{\uparrow}}^{c}_{a}\ \square\ t$$

$$\prescript{\square}{\square}{\underline{\uparrow}}^{\underline{\mathbf{u}}^{\square}_{s',\square}}\ \square\ \underline{\mathbf{u}}^{\square}_{s,\square} \equiv_{\beta\mathcal{R}} \underline{\mathbf{u}}^{\square}_{s,s'}$$

$$\underline{\boldsymbol{\pi}}^{\square}_{\square,s_2,s_3}\ \left(\prescript{\square}{\square}{\underline{\uparrow}}^{\square}_{\underline{\mathbf{u}}_{s_1,\square}}\ \square\ a\right)\ b \equiv_{\beta\mathcal{R}} \underline{\boldsymbol{\pi}}^{\square}_{s_1,s_2,s_3}\ a\ b$$

$$\underline{\boldsymbol{\pi}}^{\square}_{s_1,\square,s_3}\ a\ \left(\lambda x\,.\,\prescript{\square}{\square}{\underline{\uparrow}}^{\square}_{\underline{\mathbf{u}}_{s_2,\square}}\ \square\ b\right) \equiv_{\beta\mathcal{R}} \underline{\boldsymbol{\pi}}^{\square}_{s_1,s_2,s_3}\ a\ (\lambda x.\,b)$$

$$\prescript{\square}{\square}{\underline{\uparrow}}^{\underline{\mathbf{u}}^{\square}_{s_4,\square}}\ \square\ \left(\underline{\boldsymbol{\pi}}^{\square}_{s_1,s_2,\square}\ a\ b\right) \equiv_{\beta\mathcal{R}} \underline{\boldsymbol{\pi}}^{\square}_{s_1,s_2,s_4}\ a\ b$$

$$\prescript{\square}{\square}{\underline{\uparrow}}^{\underline{\boldsymbol{\pi}}^{\square}_{\square,s_3,\square}\ A\ C}_{\underline{\boldsymbol{\pi}}^{\square}_{s_1,s_2,\square}\ A\ B}\ \square\ \lambda x.\,b \equiv_{\beta\mathcal{R}} \lambda x : \mathtt{T}_{s_1}\ A.\left(\prescript{s_3}{s_2}{\underline{\uparrow}}^{C\ x}_{B\ x}\ \square\ b\right)$$

$$\left(\prescript{\square}{\square}{\underline{\uparrow}}^{\boldsymbol{\pi}^{\square}_{\square,s_3,\square}\ A\ C}_{\underline{\boldsymbol{\pi}}^{\square}_{s_1,s_2,\square}\ A\ B}\ \square\ b\right)\ a \equiv_{\beta\mathcal{R}} \prescript{s_3}{s_2}{\underline{\uparrow}}^{C\ a}_{B\ a}\ \square\ (b\ a)$$

$$\prescript{s_2}{\square}{\underline{\uparrow}}^{B}_{\square\ \square\ \underline{\mathbf{u}}^{\square}_{s_1,\square}\ \square\ A}\ \square\ a \equiv_{\beta\mathcal{R}} \prescript{s_2}{s_1}{\underline{\uparrow}}^{B}_{A}\ \square\ a$$

$$\prescript{\square}{s_1}{\underline{\uparrow}}^{A}_{\square\ \square\ \underline{\mathbf{u}}^{\square}_{s_2,\square}\ \square\ B}\ \square\ a \equiv_{\beta\mathcal{R}} \prescript{s_2}{s_1}{\underline{\uparrow}}^{B}_{A}\ \square\ a$$

Proof. Types are easily checked identical. Conversion was checked using Dedukti. □

These crucial properties of the encoding do not actually rely on the structure of the universe hierarchy and would hold for any CTS. Each of them correspond to a pair of

different derivations for the same judgment. For instance the first one states that if two subtyping rules are used in chain, they may be merged together into a single subtyping rule instance. These properties were identified, jointly with Thire, as necessary for the correctness of the encoding. In particular Thire studied the case of Assaf's translation, in an embedding where explicit sort-subtyping (lift) is extended to the explicit CTS-subtyping (cast) [Thi20]. He showed correctness for an abstract $\lambda\Pi_{\equiv}$ conversion and assuming that judgment have a so-called "well-structured" derivation. In the following, we take a slightly different road:

- we directly provide the rewrite system, $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$;
- we translate derivations rather than explicitly subtyped terms, allowing to ignore the well-structured hypothesis;
- we rely on particular properties of our system, such as the uniqueness of code representation to retrieve correctness of conversion.

Note that in sharp contrast with Assaf [Ass15b], the reflection property of identity lifts, $\Box\uparrow^a_a \Box\ t \equiv_{\beta\mathcal{R}} t$, does not always hold in our setting. This reflection property would require an extra non-left-linear rewrite rule.

Lemma 8.2.15 (Identity Reflection)**.** *Whenever they are well-typed in* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$, $\Box\uparrow^a_a \Box\ t$ *and* t *have the same type and are convertible in* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}] \cup \{\mathtt{u}(C, \mathtt{c}(C, T)) \longrightarrow T\}$.

This rule is, however, non-linear and makes it much harder to prove confluence of rewriting together with β. We chose to leave this rule out of our embedding and work without this convenient property which is not required for correctness. Note that the system $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}] \cup \{\mathtt{u}(C, \mathtt{c}(C, T))$ satisfies all reflection properties and can therefore correctly encode any CTS assuming typing derivation are "well-structured". It was used in practice in the Krajono translator of the Matita system.

8.3 Correctness of the translation

Correctness, or *soundness*, of the translation function is a crucial property. It guarantees that $\lambda\Pi_{\equiv}$ considered together with the encoding signature $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$ defines an encoding system *at least as* expressive as the encoded system $\mathrm{CC}^{\forall}_{\omega}$. This means that any term well-typed in, theorem or proof, has a representation in our embedding which is therefore rich enough and not too constrained to encompass the original system.

In the particular case of derivation-to-term translation, correctness could simply be stated as the well-typedness of the translation, $[\pi]$, of all derivations π of $\mathrm{CC}^{\forall}_{\omega}$. However our translation is a shallow embedding of the $\mathrm{CC}^{\forall}_{\omega}$ and provides further guarantees. In particular, if Σ is a well-formed signature, then not only should its translation $\left[\frac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_{\mathcal{S}}}\right]$ be a signature well-formed in the encoding but in that signature, for all type A and derivation $\dfrac{\pi_A}{\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s}$, the translation $\mathtt{T}_{[s]}\ [\pi_A]$ is a type in $\lambda\Pi_{\equiv}$ that represents A in the sense that derivations $\dfrac{\pi_t}{\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : A}$ should be translated into inhabitants of it. Note that

because we have type uniqueness in $\lambda\Pi_{\equiv}$, the translated type $\mathtt{T}_{[s]}\ [\pi_A]$ should not depend on the derivation π_A but rather exclusively on the term A this derivation allows to type. Rather than saying we are translating the type A, we chose to keep the translation of the derivation π_A while guaranteeing the *irrelevance* of the translation in the derivation: $\mathtt{T}_{[s]}\ [\pi_A] \equiv_{\beta\mathcal{R}} \mathtt{T}_{[s]}\ [\pi'_A]$.

8.3.1 Correctness of sort and constraint proof translation

The correctness of level and constraint translation was already covered in Lemma 6.4.6. It extends quite straightforwardly to sorts and CTS judgments.

Since we chose not to reflect any conversion at the universe level in our system, correctness of sort translation is simply its well-typedness:

Lemma 8.3.1. *For all sort s such that $\mathcal{LV}ar(s) \subseteq \overline{i}$, $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]; \overline{i} : \mathbb{N} \vdash_{\mathcal{D}} [s] : \boldsymbol{\mathcal{S}}$.*

Proof. By Lemma 6.4.6 and definition of sort translation. □

Even with algebraic universe expressions, the universe structure of $\mathrm{CC}^{\forall}_{\omega}$ allows any sort or product type to have a sort. It is *full*, as defined for PTS systems. In particular, we always have $\mathcal{A}(s, s+1)$ and $\mathcal{R}(s, s', \max(s, s'))$ which are reflected in the encoding.

Lemma 8.3.2. *For all s, s' of type $\boldsymbol{\mathcal{S}}$, $\boldsymbol{\epsilon}\ \boldsymbol{\mathcal{A}}(s, \mathtt{S}\ s)$ and $\boldsymbol{\epsilon}\ \boldsymbol{\mathcal{R}}(s, s', \mathtt{max}\ s\ s')$ are inhabited.*

Proof. Two inhabitants of these types are for instance $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}] \vdash_{\mathcal{D}} \mathtt{refl}\ (\mathtt{S}\ s)\ \mathtt{0} : \boldsymbol{\epsilon}\ \boldsymbol{\mathcal{A}}(s, \mathtt{S}\ s)$ and $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}] \vdash_{\mathcal{D}} \mathtt{refl}\ (\mathtt{max}\ s\ s')\ \mathtt{0} : \boldsymbol{\epsilon}\ \boldsymbol{\mathcal{R}}(s, s', \mathtt{max}\ s\ s')$ respectively. □

Lemma 8.3.3 (Correctness of constraint translation)**.** *Assume level variables $\overline{i}$ and a stratifiable set of atomic constraints ϕ such that $\mathcal{LV}ar(\phi) \subseteq \overline{i}$. Then we have the following:*

- *For any signature Σ, $\Sigma \vdash_{\mathcal{S}} \overline{i}/\phi; \varnothing\ \mathbf{WF}_{\mathcal{S}}$ holds by the $\mathcal{P}^{\mathbf{WF}}_{\varnothing}$ rule.*
- $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}] \vdash_{\mathcal{D}} [\mathcal{P}^{\mathbf{WF}}_{\varnothing}] := \overline{i} : \mathbb{N}, c_1 : \boldsymbol{\epsilon}\ [\phi_1], \ldots, c_k : \boldsymbol{\epsilon}\ [\phi_k]\ \mathbf{WF}_{\mathcal{D}}$.
- *If $\phi \vDash u \leq v$, $\exists p \in \mathcal{T}_{\mathrm{pub}}$, $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]; \overline{i} : \mathbb{N}, c_1 : \boldsymbol{\epsilon}\ [\phi_1], \ldots, c_k : \boldsymbol{\epsilon}\ [\phi_k] \vdash^{\mathrm{WF}}_{\mathcal{D}} p : \boldsymbol{\epsilon}\ ([u \leq v])$.*
- *If $\phi \vdash_{\mathcal{S}} \mathcal{E}(\overline{s})$, $\exists p \in \mathcal{T}_{\mathrm{pub}}$, $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]; \overline{i} : \mathbb{N}, c_1 : \boldsymbol{\epsilon}\ [\phi_1], \ldots, c_k : \boldsymbol{\epsilon}\ [\phi_k] \vdash^{\mathrm{WF}}_{\mathcal{D}} p : \boldsymbol{\epsilon}\ ([\mathcal{E}(\overline{s})])$.*

Proof. The first point is immediate. The following two are covered in Lemma 6.4.6 and definition. The proof for CTS judgments follows directly by definition of the translation and conversion of types with the rules $\boldsymbol{\mathcal{A}}$-P, $\boldsymbol{\mathcal{A}}$-T, $\boldsymbol{\mathcal{R}}$-T-P, $\boldsymbol{\mathcal{R}}$-P-T, $\boldsymbol{\mathcal{R}}$-T-T, $\boldsymbol{\mathcal{C}}$-P and $\boldsymbol{\mathcal{C}}$-T. □

Lemma 8.3.4. *For all level expression v, we have $[v\{\overline{i} \mapsto \overline{u}\}] = [v]\{\overline{i} \mapsto [\overline{u}]\}$.*

Proof. By induction on v. □

8.3.2 Codes

Codes are the internal representation of translated derivations. They are well-behaved in terms of conversion but provide absolutely no typing guarantees. For instance, the infamous $\Omega := \Delta\Delta := (\lambda x.\, x\ x)\ (\lambda x.\, x\ x)$ has a well-typed code representation:

$$\vdash_{\mathcal{D}} \boldsymbol{cA}\ (\boldsymbol{cL}\ (\lambda x : \mathbb{C}.\, \boldsymbol{cA}\ x\ x))\ (\boldsymbol{cL}\ (\lambda x : \mathbb{C}.\, \boldsymbol{cA}\ x\ x)) : \mathbb{C}$$

This term is non-terminating in the encoding too: it reduces to itself with the $c\beta$ rewrite rule. Note that we chose to ignore the type annotation of λ-abstractions in the code representation. Barras and Grégoire showed [BG05] that in CTS, type annotations may safely be ignored while checking convertibility between two terms which types are either identical or both sorts.

Definition 8.3.5. *We define the three following functions on terms and contexts of* $\mathrm{CC}^{\forall}_{\omega}$.

$$
\begin{aligned}
|x|_{\Sigma;\Gamma} &:= x && \text{if } x \notin \Gamma \\
|x|_{\Sigma;\Gamma} &:= \mathtt{c}(|A|_{\Sigma;\Gamma}, x) && \text{if } x : A \in \Gamma \\
|\lambda x : A.\, t|_{\Sigma;\Gamma} &:= \boldsymbol{cL}\ (\lambda x : \mathbb{C}.\, |t|_{\Sigma;\Gamma}) \\
|t\ u|_{\Sigma;\Gamma} &:= \boldsymbol{cA}\ \ |t|_{\Sigma;\Gamma}\ \ |u|_{\Sigma;\Gamma} \\
|s|_{\Sigma;\Gamma} &:= \mathbf{u}_{[s]} && \text{if } s \in \mathcal{S} \\
|\Pi x : A.\, B|_{\Sigma;\Gamma} &:= \boldsymbol{\pi}\ \ |A|_{\Sigma;\Gamma}\ \ (\lambda x : \mathbb{C}.\, |B|_{\Sigma;\Gamma}) \\
|\mathtt{c}_{\overline{u}}|_{\Sigma;\Gamma} &:= \mathtt{c}'\ \ [u_1]\ \ \ldots\ \ [u_n] && \text{if } \mathtt{c}[\overline{i}/\phi] : \tau \in \Sigma \text{ (with } n = |\overline{i}|) \\
|\mathtt{c}_{\overline{u}}|_{\Sigma;\Gamma} &:= |t|_{\Sigma;\Gamma}\,\{\overline{i} \mapsto [\overline{u}]\} && \text{if } \mathtt{c}[\overline{i}/\phi] := t : \tau \in \Sigma \\
\\
[\![T]\!]_{\Sigma;\Gamma} &:= \mathtt{D}\ \ |T|_{\Sigma;\Gamma} \\
\\
|\varnothing|_{\Sigma} &:= \varnothing \\
|\Gamma, x : A|_{\Sigma} &:= |\Gamma|_{\Sigma}, x : [\![A]\!]_{\Sigma;\Gamma}
\end{aligned}
$$

These translation functions rely on the private signature and are not meant to be implemented. The first one, $|t|_{\Sigma;\Gamma}$ corresponds to the unique private representation of the well-typed term t, independently from its typing derivation. The second $[\![t]\!]_{\Sigma;\Gamma}$ is the representation of t as a type.

Note that this function is only well-defined on terms t and signature Σ such that all symbol occurrences $\mathtt{c}_{\overline{u}}$ in t are defined or declared in Σ with a set of levels variables $\overline{i}$ such that $|\overline{u}| = |\overline{i}|$. This is always the case when considering well-formed signature and subterms $t \lhd u$ of terms u well-typed in $\Sigma; \ldots; \Gamma$.

Lemma 8.3.6. *For all context* Γ *such that* $x \notin \Gamma$*, we have*

$$
\begin{aligned}
|t|_{\Sigma;\Gamma,x:A} &= |t|_{\Sigma;\Gamma}\,\{x \mapsto \mathtt{c}(|A|_{\Sigma;\Gamma}, x)\} \\
[\![t]\!]_{\Sigma;\Gamma,x:A} &= [\![t]\!]_{\Sigma;\Gamma}\,\{x \mapsto \mathtt{c}(|A|_{\Sigma;\Gamma}, x)\} \\
[\![s]\!]_{\Sigma;\Gamma} &\xrightarrow[\Sigma]{} \mathtt{U}_{[s]} \\
[\![\Pi x : A.\, B]\!]_{\Sigma;\Gamma} &\xrightarrow[\Sigma]{} \Pi x : [\![A]\!]_{\Sigma;\Gamma}.\, [\![B]\!]_{\Sigma;\Gamma,x:A}
\end{aligned}
$$

Proof. By induction on t. If $t = x \notin \Gamma$, $|x|_{\Gamma,x:A} = \mathtt{c}(|A|_{\Sigma;\Gamma}, x) = x\{x \mapsto \mathtt{c}(|A|_{\Sigma;\Gamma}, x)\}$. If $t = y \neq x$, $|y|_{\Gamma,x:A} = |y|_{\Gamma}$. All other cases directly follow from induction hypothesis. The second equality follows by definition.

By definition, $[\![s]\!]_{\Sigma;\Gamma} := \mathtt{D}\ \mathbf{u}_{[s]} \xrightarrow[\text{D-u}]{} \mathtt{U}_{[s]}$ and $[\![\Pi x : A.\, B]\!]_{\Sigma;\Gamma} := \mathtt{D}\ (\boldsymbol{\pi}\ |A|_{\Sigma;\Gamma}\ (\lambda x : \mathbb{C}.\, |B|_{\Sigma;\Gamma}))$ which rewrites to $\Pi x : \mathtt{D}\ |A|_{\Sigma;\Gamma}.\, \mathtt{D}\ (|B|_{\Sigma;\Gamma}\ \{x \mapsto \mathtt{c}(|A|_{\Sigma;\Gamma}, x)\}) = \Pi x : [\![A]\!]_{\Sigma;\Gamma}.\, [\![B]\!]_{\Gamma,x:A}$. □

Definition 8.3.7. *A context is* closed *if it is either* $\varnothing$ *or* $\Gamma, x : A$ *with* Γ *closed and* $\mathcal{FV}ar(A) \subseteq \Gamma$.

Lemma 8.3.8. *Assume a signature* Σ.

- *if* Γ *is closed and* $x \notin \mathcal{FV}ar(t)$, *then* $|t|_{\Sigma;\Gamma,x:A} = |t|_{\Sigma;\Gamma}$;
- *if* $\Gamma \subseteq \Gamma'$ *closed and* $\mathcal{FV}ar(t) \subseteq \Gamma$, *then* $|t|_{\Sigma;\Gamma} = |t|_{\Sigma;\Gamma'}$;
- *if* Γ *and* t *closed, then* $|t|_{\Sigma;\Gamma} = |t|_{\Sigma;\varnothing}$;
- *if* Γ *is closed and* $(x : A) \in \Gamma$, *then* $x : [\![A]\!]_{\Sigma;\Gamma} \in |\Gamma|_\Sigma$;

Proof. First is Corollary of Lemma 8.3.6. Second is done by a simple induction on Γ'. Third is a particular case of second. Because if $\Gamma = \Gamma_1, x : A, \Gamma_2$ is closed, then $\mathcal{FV}ar(A) \subseteq \Gamma_1$ and $[\![A]\!]_{\Sigma;\Gamma_1} = [\![A]\!]_{\Sigma;\Gamma}$ which proves fourth. □

Lemma 8.3.9. *If* $\dfrac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_\mathcal{S}}$ *and* $\Sigma \vdash^{\text{WF}}_\mathcal{S} \Gamma\ \mathbf{WF}_\mathcal{S}$ *then* $\mathcal{D}[\text{CC}^\forall_\omega], [\pi_\Sigma] \vdash_\mathcal{D} |\Gamma|_\Sigma\ \mathbf{WF}_\mathcal{D}$.
Besides if $\mathcal{FV}ar(t) \subseteq \Gamma$ *then* $\mathcal{D}[\text{CC}^\forall_\omega]; |\Gamma|_\Sigma \vdash_\mathcal{D} |t|_{\Sigma;\Gamma} : \mathbb{C}$.

Proof. Note that we do not concern ourselves with the well-typedness of $\mathcal{D}[\text{CC}^\forall_\omega], [\pi_\Sigma]$ yet.

By induction on the length of Γ which is closed since $\Sigma \vdash_\mathcal{S} \Gamma\ \mathbf{WF}_\mathcal{S}$. The property holds for empty contexts. Assuming it holds for Γ, then by induction on t such that $\mathcal{FV}ar(t) \subseteq \Gamma$ we have $\mathcal{D}[\text{CC}^\forall_\omega] \vdash_\mathcal{D} |t|_{\Sigma;\Gamma} : \mathbb{C}$. We use Lemma 8.3.8 for the variable case, all other cases are done directly by definition and induction hypothesis. We deduce that $|\Gamma, x : A|_\Sigma := |\Gamma|_\Sigma, x : \mathtt{D}\ |A|_{\Sigma;\Gamma}$ is well-typed if $\Gamma, x : A$ is closed. □

Note that t does not need to be well-typed. In fact it was not even necessary for Γ to be well-typed, the only important property is that each type annotation relies exclusively on free variables that were previously declared.

Lemma 8.3.10. *For all level substitution* $\{\bar{i} \mapsto \overline{u}\}$, $|t\{\bar{i} \mapsto \overline{u}\}|_{\Sigma;\Gamma} = |t|_{\Sigma;\Gamma}\ \{\bar{i} \mapsto [\overline{u}]\}$.

Proof. By induction on t, the only interesting case is when $t = s$ and $t = \mathtt{c}_{\overline{u}}$ for which the property holds by Lemma 8.3.4. □

Lemma 8.3.11. *If* $x \notin \Gamma$, *then* $|t\{x \mapsto u\}|_{\Sigma;\Gamma} = |t|_{\Sigma;\Gamma}\ \{x \mapsto |u|_{\Sigma;\Gamma}\}$.

Proof. By induction on t, the only interesting case is when $t = x$ for which the property holds since $x \notin \Gamma$. □

Lemma 8.3.12. *If* $t \xrightarrow[\beta]{} u$, *hen* $|t|_{\Sigma;\Gamma} \xrightarrow[\beta\mathcal{R}]{} |u|_{\Sigma;\Gamma}$.

Proof. By induction of t. If the β-reduction occurs at the root, then $t = (\lambda x : A.\, v)\ w$ and $u = v\{x \mapsto w\}$. In that case we have by definition and Lemma 8.3.11:

$$|t|_{\Sigma;\Gamma} = \boldsymbol{cA}\ (\boldsymbol{cL}\ \lambda x : \mathbb{C}.\, |v|_{\Sigma;\Gamma})\ |w|_{\Sigma;\Gamma} \xrightarrow[\text{c}\beta]{} |v|_{\Sigma;\Gamma}\ \{x \mapsto |w|_{\Sigma;\Gamma}\} = |v\{x \mapsto w\}|_{\Sigma;\Gamma} = |u|_{\Sigma;\Gamma}$$

Other cases follow by induction hypothesis. □

Lemma 8.3.13. *If* $t \xrightarrow[\delta]{} u$, *then* $|t|_{\Sigma;\Gamma} = |u|_{\Sigma;\Gamma}$.

Proof. Immediate since, by Lemma 8.3.10, $|t|_{\Sigma;\Gamma}$ is defined so that $|t|_{\Sigma;\Gamma} = |t{\downarrow_\delta}|_{\Sigma;\Gamma}$. □

Lemma 8.3.14. *If* $t \equiv_{\beta\Sigma} u$, *then* $|t|_{\Sigma;\Gamma} \equiv_{\beta\mathcal{R}} |u|_{\Sigma;\Gamma}$.

Proof. By definition and confluence of $\beta\delta$, $t \xrightarrow[\beta\delta]{}\!\!\!\!\!\rightarrow \xleftarrow[\beta\delta]{}\!\!\!\!\!\leftarrow u$ and using Lemma 8.3.12 and Lemma 8.3.13, we deduce $|t|_{\Sigma;\Gamma} \xrightarrow[\beta\mathcal{R}]{}\!\!\!\!\!\rightarrow \xleftarrow[\beta\mathcal{R}]{}\!\!\!\!\!\leftarrow |u|_{\Sigma;\Gamma}$. □

Corollary 8.3.14.1. *If* $t \equiv_{\beta\Sigma\phi} u$, $\mathbf{NA}(t)$, $\mathbf{NA}(u)$ *and* ϕ *acyclic, then* $|t|_{\Sigma;\Gamma} \equiv_{\beta\mathcal{R}} |u|_{\Sigma;\Gamma}$.

Proof. By Lemma 8.3.14 and Lemma 7.3.8. □

8.3.3 Correctness of conversion

We are now able to prove that the conversion in $\mathrm{CC}^{\forall}_{\omega}$ is properly reflected in the translation of derivations. We need to assume the three properties justified in Section 7.3: **(H1)**, **(H2)** and **(H3)**.

Lemma 8.3.15. *Assume* $\dfrac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_{\mathcal{S}}}$ *and* $\dfrac{\pi}{\Sigma;\bar{i}/\phi;\Gamma \vdash_{\mathcal{S}} t : A}$ *two derivations satisfying* ***(H1)***, ***(H2)*** *and* ***(H3)***. *Then* $\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi]) \equiv_{\beta\mathcal{R}} |t|_{\Sigma;\Gamma}$ *for* $\mathcal{R} := \Sigma_{\mathrm{pri}}, [\pi_\Sigma]$.

Proof. By induction on the length of Σ, Γ and on the derivation π.

- $\mathcal{P}_{\mathcal{S}}$: $t = s$ and $A = s + 1$.
  $\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi]) = \mathtt{c}(|A|_{\Sigma;\Gamma}, \underline{\mathbf{u}}^{[\pi_A]}_{[s],[s']}) \xrightarrow[\text{enc-u}]{} \mathtt{c}(|A|_{\Sigma;\Gamma}, \mathtt{u}(\mathbf{u}_{[s']}, \mathbf{u}_{[s]})) \xrightarrow[\text{c-u}]{} \mathbf{u}_{[s]} = |s|_{\Sigma;\Gamma}$.
- $\mathcal{P}_{\mathcal{X}}$: $t = x$ and $x : A \in \Gamma$. $\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi]) = \mathtt{c}(|A|_{\Sigma;\Gamma}, x) = |t|_{\Sigma;\Gamma}$ by definition.
- $\mathcal{P}_{\Pi}$: $t = \Pi x : U.\, V$ and $A = s$.

  $$\begin{aligned}\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi]) &= \mathtt{c}(|A|_{\Sigma;\Gamma}, \underline{\boldsymbol{\pi}}^{\square}_{s_1,s_2,s_3}\ [\pi_U]\ \ \lambda x : \mathtt{T}_{[s_1]}\ \ [\pi_U].\, [\pi_V]) \\ &\xrightarrow[\text{enc-}\pi]{} \mathtt{c}(|A|_{\Sigma;\Gamma}, \mathtt{u}(\mathbf{u}_{s_3}, \boldsymbol{\pi}\ \mathtt{c}(\mathbf{u}_{s_1}, [\pi_U])\ \lambda x : \mathbb{C}.\, \mathtt{c}(\mathbf{u}_{s_2}, [\pi_V]\,\{x \mapsto \mathtt{u}(\mathtt{c}(\mathbf{u}_{s_1}, [\pi_U]), x)\}))) \\ &\xrightarrow[\text{c-u}]{} \boldsymbol{\pi}\ \mathtt{c}(\mathbf{u}_{s_1}, [\pi_U])\ \lambda x : \mathbb{C}.\, \mathtt{c}(\mathbf{u}_{s_2}, [\pi_V]\,\{x \mapsto \mathtt{u}(\mathtt{c}(\mathbf{u}_{s_1}, [\pi_U]), x)\})\end{aligned}$$

  By induction hypothesis we have $|\pi_U|_{\Sigma;\Gamma} = \mathtt{c}(\mathbf{u}_{s_1}, [\pi_U])$ and $|\pi_V|_{\Sigma;\Gamma;x:A} = \mathtt{c}(\mathbf{u}_{s_2}, [\pi_V])$. From the latter and Lemma 8.3.6 we deduce

  $$\begin{aligned}\mathtt{c}(\mathbf{u}_{s_2}, [\pi_V])\{x \mapsto \mathtt{u}(|\pi_U|_{\Sigma;\Gamma}, x)\} &= |\pi_V|_{\Sigma;\Gamma;x:A}\,\{x \mapsto \mathtt{u}(|\pi_U|_{\Sigma;\Gamma}, x)\} \\ &= |\pi_V|_{\Sigma;\Gamma}\,\{x \mapsto \mathtt{c}(|A|_{\Sigma;\Gamma}, x)\}\{x \mapsto \mathtt{u}(|\pi_U|_{\Sigma;\Gamma}, x)\} \\ &= |\pi_V|_{\Sigma;\Gamma}\,\{x \mapsto \mathtt{c}(|A|_{\Sigma;\Gamma}, \mathtt{u}(|\pi_U|_{\Sigma;\Gamma}, x))\} \\ &\xrightarrow[\text{c-u}]{}\!\!\!\!\!\rightarrow |\pi_V|_{\Sigma;\Gamma}\,\{x \mapsto x\} = |\pi_V|_{\Sigma;\Gamma}\end{aligned}$$

  This allows to conclude

  $$\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi]) \equiv_{\beta\mathcal{R}} \boldsymbol{\pi}\ \ |U|_{\Sigma;\Gamma}\ \ \lambda x : \mathbb{C}.\, |V|_{\Sigma;\Gamma} = |\Pi x : U.\, V|_{\Sigma;\Gamma} = |t|_{\Sigma;\Gamma}$$

- $\mathcal{P}_\lambda$: $t = \lambda x : U.\, v$ and $A = \Pi x : U.\, V$.

$$\begin{aligned}\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi]) &= \mathtt{c}(\boldsymbol{\pi}\ \ |U|_{\Sigma;\Gamma}\ \ \lambda x : \mathbb{C}.\, |V|_{\Sigma;\Gamma}, \lambda x\ :\ \mathtt{T}_{[s]}\ \ [\pi_U]\ .\ \ [\pi_v]) \\ &\xrightarrow[\text{c-}\lambda]{} \boldsymbol{\mathtt{cL}}\ \ \lambda x : \mathbb{C}.\, \mathtt{c}(|V|_{\Sigma;\Gamma}, [\pi_v]\, \{x \mapsto \mathtt{u}(|U|_{\Sigma;\Gamma}, x)\})\end{aligned}$$

  Again, by induction hypothesis, $|v|_{\Sigma;\Gamma;x:U} = \mathtt{c}(|V|_{\Sigma;\Gamma,x:U}, [\pi_v])$ and

$$\begin{aligned}|v|_{\Sigma;\Gamma;x:U}\, \{x \mapsto \mathtt{u}(|U|_{\Sigma;\Gamma}, x)\} &= |v|_{\Sigma;\Gamma}\, \{x \mapsto \mathtt{c}(|U|_{\Sigma;\Gamma}, x)\}\{x \mapsto \mathtt{u}(|\pi_U|_{\Sigma;\Gamma}, x)\} \\ &= |v|_{\Sigma;\Gamma}\, \{x \mapsto \mathtt{c}(|U|_{\Sigma;\Gamma}, \mathtt{u}(|U|_{\Sigma;\Gamma}, x))\} \\ &\xrightarrow[\text{c-u}]{} |v|_{\Sigma;\Gamma}\, \{x \mapsto x\} = |v|_{\Sigma;\Gamma} \\ |V|_{\Sigma;\Gamma;x:U}\, \{x \mapsto \mathtt{u}(|U|_{\Sigma;\Gamma}, x)\} &\xrightarrow[\text{c-u}]{} |V|_{\Sigma;\Gamma}\end{aligned}$$

  from which we have $|v|_{\Sigma;\Gamma} \xleftarrow[\text{c-u}]{} \xrightarrow[\text{c-u}]{} \mathtt{c}(|V|_{\Sigma;\Gamma}, [\pi_v]\, \{x \mapsto \mathtt{u}(|U|_{\Sigma;\Gamma}, x)\})$ and therefore $\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi]) \equiv_{\beta\mathcal{R}} \boldsymbol{\mathtt{cL}}\ \lambda x : \mathbb{C}.\, |v|_{\Sigma;\Gamma} = |\lambda x : U.\, v|_{\Sigma;\Gamma} = |t|_{\Sigma;\Gamma}$.

- $\mathcal{P}_@$: $t = u\ v$, $\frac{\pi_u}{\vdash_{\mathcal{S}} u : \Pi x : V.\, T}$, $A = T\{x \mapsto v\}$ and π_v's last derivation is a $\mathcal{P}_\preceq$:

$$\frac{\dfrac{\pi'_v}{\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} v : W} \qquad \dfrac{\pi_W}{\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} W : s} \qquad \dfrac{\pi_V}{\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} V : s'} \qquad \phi \vdash_{\mathcal{S}} W \preceq_{\Sigma\phi} V}{\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} v : V}$$

  We have by induction hypothesis, $|u|_{\Sigma;\Gamma} = \mathtt{c}(\boldsymbol{\pi}\ \ |V|_{\Sigma;\Gamma}\ \ \lambda x : \mathbb{C}.\, |T|_{\Sigma;\Gamma}, [\pi_u])$, therefore:

$$\begin{aligned}|t|_{\Sigma;\Gamma} &= \boldsymbol{\mathtt{cA}}\ \mathtt{c}(\boldsymbol{\pi}\ \ |V|_{\Sigma;\Gamma}\ \ \lambda x : \mathbb{C}.\, |T|_{\Sigma;\Gamma}, [\pi_u])\ \ |v|_{\Sigma;\Gamma} \\ &\xrightarrow[\text{c-a}]{} \mathtt{c}(|T|_{\Sigma;\Gamma}\, \{x \mapsto |v|_{\Sigma;\Gamma}\}, [\pi_u]\ \ \mathtt{u}(|V|_{\Sigma;\Gamma}, |v|_{\Sigma;\Gamma}))\end{aligned}$$

  By Lemma 8.3.11, $|T|_{\Sigma;\Gamma}\, \{x \mapsto |v|_{\Sigma;\Gamma}\} = |T\{x \mapsto v\}|_{\Sigma;\Gamma} = |A|_{\Sigma;\Gamma}$, and by definition $[\pi_t] = [\pi_u]\ \ [\pi_v]$, it remains only to show that $[\pi_v] \equiv_{\beta\mathcal{R}} \mathtt{u}(|V|_{\Sigma;\Gamma}, |v|_{\Sigma;\Gamma})$. This would not hold without the **(H3)** property, but in our case

$$\begin{aligned}[\pi_v] &= \begin{smallmatrix}[s'] \\ [s]\end{smallmatrix}\!\!\uparrow\!\!\!\!\!\!\begin{smallmatrix}[\pi_V] \\ [\pi_W]\end{smallmatrix}\ \square\ \ [\pi'_v] \\ &\xrightarrow[\text{enc-}\uparrow]{} \mathtt{u}(\mathtt{c}(\mathbf{u}_{[s']}, [\pi_V]), \mathtt{c}(\mathtt{c}(\mathbf{u}_{[s]}, [\pi_W]), [\pi'_v])) \\ &\equiv_{\beta\mathcal{R}} \mathtt{u}(|V|_{\Sigma;\Gamma}, \mathtt{c}(|W|_{\Sigma;\Gamma}, [\pi'_v])) \equiv_{\beta\mathcal{R}} \mathtt{u}(|V|_{\Sigma;\Gamma}, |v|_{\Sigma;\Gamma})\end{aligned}$$

  By successive induction hypothesis on the π'_v, π_W and π_V subtrees.

- $\mathcal{P}_\preceq$: $t : B \preceq_{\Sigma\phi} A$. The result follows by induction hypothesis on the derivation of $\vdash_{\mathcal{S}} t : B$ and $\vdash_{\mathcal{S}} B : s$.

$$\begin{aligned}\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi_t]) &= \mathtt{c}(|A|_{\Sigma;\Gamma}, \begin{smallmatrix}[s'] \\ [s]\end{smallmatrix}\!\!\uparrow\!\!\!\!\!\!\begin{smallmatrix}[\pi_A] \\ [\pi_B]\end{smallmatrix}\ \square\ \ [\pi'_t]) \\ &\xrightarrow[\text{enc-}\uparrow]{} \mathtt{c}(|A|_{\Sigma;\Gamma}, \mathtt{u}(\mathtt{c}(\mathbf{u}_{[s']}, [\pi_A]), \mathtt{c}(\mathtt{c}(\mathbf{u}_{[s]}, [\pi_B]), [\pi'_t]))) \\ &\xrightarrow[\text{c-u}]{} \mathtt{c}(\mathtt{c}(\mathbf{u}_{[s]}, [\pi_B]), [\pi'_t]) \equiv_{\beta\mathcal{R}} \mathtt{c}(|B|_{\Sigma;\Gamma}, [\pi'_t]) \equiv_{\beta\mathcal{R}} |t|_{\Sigma;\Gamma}\end{aligned}$$

- $\mathcal{P}_{decl}$: $t = \mathtt{c}_{\overline{u}}$ and $A = \tau\{\overline{i} \mapsto \overline{u}\}$ for τ such that $(\mathtt{c}[\overline{i}/\phi] : \tau) \in \Sigma$.

  $\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi]) = \mathtt{c}(|A|_{\Sigma;\Gamma}, \mathtt{c}\ \ [\overline{u}]\ \ [\overline{\pi_\phi}]) \longrightarrow \mathtt{c}(|A|_{\Sigma;\Gamma}, \mathtt{u}(\ldots, \mathtt{c}'\ \ [\overline{u}])) \xrightarrow[\text{c-u}]{} \mathtt{c}'\ \ [\overline{u}] = |\mathtt{c}_{\overline{u}}|_{\Sigma;\Gamma}$

- $\mathcal{P}_{def}$: $t = \mathtt{c}_{\overline{u}}$ and $A = \tau\{\overline{i} \mapsto \overline{u}\}$ for τ such that $(\mathtt{c}[\overline{i}/\phi] := t : \tau) \in \Sigma$. We have $\dfrac{\pi'}{\Sigma'; \overline{i}/\phi; \varnothing \vdash_{\mathcal{S}} t : \tau}$ for some $\Sigma' \subseteq \Sigma$.

  By induction hypothesis, $\mathtt{c}(|\tau|_{\Sigma';\varnothing}, [\pi']) \equiv_{\beta\mathcal{R}} |t|_{\Sigma';\varnothing}$ and since extra variables and symbols in Γ and Σ are not in t, by well-formedness of $\Sigma; \overline{i}/\phi; \Gamma$, we have, by Lemma 8.3.8, $\mathtt{c}(|\tau|_{\Sigma;\Gamma}, [\pi']) \equiv_{\beta\mathcal{R}} |t|_{\Sigma;\Gamma}$. By Lemma 8.3.10, $|A|_{\Sigma;\Gamma} = |\tau|_{\Sigma;\Gamma}\{\overline{i} \mapsto [u]\}$ and finally

  $$\begin{aligned}\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi]) &= \mathtt{c}(|\tau|_{\Sigma;\Gamma}\{\overline{i} \mapsto [u]\}, \mathtt{c}\ [\overline{u}]\ [\overline{\pi_\phi}]) \\ &\longrightarrow \mathtt{c}(|\tau|_{\Sigma;\Gamma}\{\overline{i} \mapsto [u]\}, [\pi']\{\overline{i} \mapsto [u]\}) \\ &= \mathtt{c}(|\tau|_{\Sigma;\Gamma}, [\pi'])\{\overline{i} \mapsto [u]\} = |t|_{\Sigma;\Gamma}\{\overline{i} \mapsto [u]\} = |\mathtt{c}_{\overline{u}}|_{\Sigma;\Gamma}\end{aligned}$$ □

This characterization of derivation allows to easily get the correctness of conversion result we need:

Corollary 8.3.15.1. *Assume* $\dfrac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_{\mathcal{S}}}$, $\dfrac{\pi_t}{\Sigma; \overline{i}/\phi; \Gamma \vdash t : A}$ *and* $\dfrac{\pi_u}{\Sigma; \overline{i}/\phi; \Gamma \vdash u : B}$ *all satisfying* ***(H1)***, ***(H2)*** *and* ***(H3)***, *such that* $t \equiv_{\beta\Sigma\phi} u$.

Then $\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi_t]) \equiv_{\beta\mathcal{R}} \mathtt{c}(|B|_{\Sigma;\Gamma}, [\pi_u])$ *for* $\mathcal{R} = \Sigma_{\text{pri}}, [\pi_\Sigma]$.

Proof. Follows from Lemma 8.3.15 and Lemma 8.3.14 since we have:

$$\mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi_t]) \equiv_{\beta\mathcal{R}} |t|_{\Sigma;\Gamma} \equiv_{\beta\mathcal{R}} |u|_{\Sigma;\Gamma} \equiv_{\beta\mathcal{R}} \mathtt{c}(|B|_{\Sigma;\Gamma}, [\pi_u]).$$ □

Note that we could not have hoped to have $[\pi_t] \equiv_{\beta\mathcal{R}} [\pi_u]$ since these two terms may not have the same type and the embedding is type preserving. However if $B = A$ (or even $B \equiv_{\beta\Sigma} A$), then $[\pi_t]$ and $[\pi_u]$ are two representations of convertible terms and it would be reasonable to expects these representations to be convertible. They might however not be in our encoding because we do not have the elimination of identity cast property. The term x can simply be translated into x if it is typed with the axiom rule but it can also be translated into ${}^{s}_{s}\!\uparrow^{A}_{\perp A}\ p\ x$ if it is typed with a non-required identity subtyping rule. This is where explicit subtyping becomes frustrating to deal with as it breaks the full reflection.

A first solution would be to guarantee these identity casts never occur. It is possible to define the translation such that identity subtyping rules are collapsed and no identity cast is generated. However, because of universe polymorphism, identity casts may originate from the collapsing, by the universe substitution of a δ-reduction, of a cast that was required on the definition's body with abstract universes.

Full reflection using lift elimination

Eliminating identity casts can be done by means of the extra non-linear rewrite rule $\mathtt{u}(C, \mathtt{c}(C,T)) \longrightarrow T$. Assuming we added this rule to our rewrite system, then the previous Lemma 8.3.15 would no longer need the **(H3)** assumption.

Lemma 8.3.16. *Assume* $\dfrac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_\mathcal{S}}$ *and* $\dfrac{\pi_t}{\Sigma;\bar{i}/\phi;\Gamma \vdash t : A}$ *two derivations satisfying* ***(H1)*** *and* ***(H2)****. Then* $[\pi_t] \equiv_{\beta\mathcal{R}} \mathtt{u}(|A|_{\Sigma;\Gamma}, |t|_{\Sigma;\Gamma})$ *for* $\mathcal{R} = \Sigma_{\text{pri}}\ ,\ \{\ \mathtt{u}(C, \mathtt{c}(C,T)) \longrightarrow T\ \}\ ,\ [\pi_\Sigma]$.

Proof. The proof is done by reusing all cases of the proof of Lemma 8.3.15 since we now have $\mathtt{u}(|A|_{\Sigma;\Gamma}, \mathtt{c}(|A|_{\Sigma;\Gamma}, \pi_t)) \equiv_{\beta\mathcal{R}} [\pi_t]$. The only difference is for the application case where **(H3)** was required but is no longer assumed. In that case, we have $t = u\ v$, $\frac{\pi_u}{\vdash_\mathcal{S} u : \Pi x : V.\, T}$, $A = T\{x \mapsto v\}$ and $\frac{\pi_v}{\vdash_\mathcal{S} v : V}$. By induction hypothesis

$$\begin{array}{rcl}
[\pi] = [\pi_u]\ [\pi_v] & = & \mathtt{u}(\boldsymbol{\pi}\ \ |V|_{\Sigma;\Gamma}\ \ \lambda x : \mathbb{C}.\, |T|_{\Sigma;\Gamma}, |u|_{\Sigma;\Gamma})\ \mathtt{u}(|V|_{\Sigma;\Gamma}, |v|_{\Sigma;\Gamma}) \\
& \xrightarrow[\text{u-a}]{} & \mathtt{u}\Big(\ \ |T|_{\Sigma;\Gamma}\{x \mapsto \mathtt{c}(|V|_{\Sigma;\Gamma}, \mathtt{u}(|V|_{\Sigma;\Gamma}, |u|_{\Sigma;\Gamma}))\} \\
& & \qquad ,\quad \boldsymbol{cA}\ \ |u|_{\Sigma;\Gamma}\ \ \mathtt{c}(|V|_{\Sigma;\Gamma}, \mathtt{u}(|V|_{\Sigma;\Gamma}, |v|_{\Sigma;\Gamma}))\Big) \\
& \xrightarrow[\text{c-u}]{}\!\!\!\!\!\!\rightarrow & \mathtt{u}(|T|_{\Sigma;\Gamma}\{x \mapsto |u|_{\Sigma;\Gamma}\}, \boldsymbol{cA}\ \ |u|_{\Sigma;\Gamma}\ \ |v|_{\Sigma;\Gamma})
\end{array}$$

We conclude by Lemma 8.3.11 since $|T|_{\Sigma;\Gamma}\{x \mapsto |v|_{\Sigma;\Gamma}\} = |T\{x \mapsto v\}|_{\Sigma;\Gamma} = |A|_{\Sigma;\Gamma}$. □

Corollary 8.3.15.1 would then provide the corresponding *full reflection* lemma without assuming the **(H3)** property. We chose to remove this rule from our encoding since, unlike its counterpart c-u, this rule cannot be linearized using typing.

8.3.4 Correctness of subtyping

Lemma 8.3.17. *Assume* $\dfrac{\pi_A}{\Sigma;\bar{i}/\phi;\Gamma \vdash_\mathcal{S} A : s_A}$ *and* $\dfrac{\pi_B}{\Sigma;\bar{i}/\phi;\Gamma \vdash_\mathcal{S} B : s_B}$ *both satisfy the* ***(H1)****,* ***(H2)*** *and* ***(H3)*** *properties and* $\phi \vdash_\mathcal{S} A \preceq_{\Sigma\phi} B$*. Then either* $A \equiv_{\beta\Sigma\phi} B$ *or we have* $A \equiv_{\beta\Sigma\phi} \Pi x_1 : U_1 \ldots \Pi x_n : U_n.\, s$ *and* $B \equiv_{\beta\Sigma\phi} \Pi x_1 : U_1 \ldots \Pi x_n : U_n.\, s'$ *for some sorts* s*,* s' *such that* $\phi \vdash_\mathcal{S} \mathcal{C}(s,s')$*. In both cases, there exists a public term* $p \in \mathcal{T}_{\text{pub}}$ *such that*

$$\mathcal{D}[\mathrm{CC}^\forall_\omega]; \bar{i} : \mathbb{N}, c_1 : [\phi_1], \ldots, c_k : [\phi_k] \vdash_\mathcal{D} p : \boldsymbol{\epsilon}\ ([\pi_A] \subseteq^{[s_B]}_{[s_A]} [\pi_B])$$

Proof. The first part is an usual property of CTS, see Lemma 5.3.2. We prove by a simple induction on the derivation of $\phi \vdash_\mathcal{S} A \preceq_{\Sigma\phi} B$ that it holds for $\mathrm{CC}^\forall_\omega$ too.

$$\begin{array}{rcll}
[\pi_A] \subseteq^{[s_B]}_{[s_A]} [\pi_B] & \xrightarrow[\text{ST-d}]{} & \mathtt{c}(\mathbf{u}_{[s_A]}, [\pi_A]) \preceq \mathtt{c}(\mathbf{u}_{[s_B]}, [\pi_B]) = \mathtt{c}(|s_A|_{\Sigma;\Gamma}, [\pi_A]) \preceq \mathtt{c}(|s_B|_{\Sigma;\Gamma}, [\pi_B]) \\
& \equiv_{\beta\Sigma\phi} & |A|_{\Sigma;\Gamma} \preceq |B|_{\Sigma;\Gamma} & \text{by Lemma 8.3.15}
\end{array}$$

If $A \equiv_{\beta\Sigma\phi} B$, then, by Lemma 8.3.14, $|A|_{\Sigma;\Gamma} \equiv_{\beta\mathcal{R}} |B|_{\Sigma;\Gamma}$ and we conclude by choosing, for instance, $p := \mathtt{strefl}\ \ |s|_{\Sigma;\Gamma}\ \ |A|_{\Sigma;\Gamma}$.

Otherwise, we prove by induction on the number n of products in the normal form of A and B that $|A|_{\Sigma;\Gamma} \preceq |B|_{\Sigma;\Gamma} \equiv_{\beta\mathcal{R}} [\mathcal{C}(s,s')]$. The base case is immediate using the ST-u rule. Assuming $A \equiv_{\beta\Sigma\phi} \Pi x_1 : U_1.\, A'$ and $B \equiv_{\beta\Sigma\phi} \Pi x_1 : U_1.\, B'$ such that $\phi \vdash_{\mathcal{S}} A' \preceq_{\Sigma\phi} B'$.

$$
\begin{array}{lll}
|A|_{\Sigma;\Gamma} \preceq |B|_{\Sigma;\Gamma} & \twoheadrightarrow & |\Pi x_1 : U_1.\, A'|_{\Sigma;\Gamma} \preceq |\Pi x_1 : U_1.\, B'|_{\Sigma;\Gamma} \\
& = & \left(\boldsymbol{\pi}\ |U_1|_{\Sigma;\Gamma}\ \lambda x_1.\, |A'|_{\Sigma;\Gamma}\right) \preceq \left(\boldsymbol{\pi}\ |U_1|_{\Sigma;\Gamma}\ \lambda x_1.\, |B'|_{\Sigma;\Gamma}\right) \\
& \underset{\text{ST-}\pi}{\longrightarrow} & \forall\ \lambda c : \mathbb{C}.\, \left(|A'|_{\Sigma;\Gamma} \preceq |B'|_{\Sigma;\Gamma}\right) \{x_1 \mapsto c\} \\
& \equiv_{\beta\mathcal{R}} & \forall\ \lambda c : \mathbb{C}.\, ([\mathcal{C}(s,s')])\, \{x_1 \mapsto c\} \quad \text{by induction hypothesis} \\
& = & \forall\ \lambda c : \mathbb{C}.\, [\mathcal{C}(s,s')] \quad \text{since sorts only contain level variables} \\
& \underset{\text{ST-}\forall}{\longrightarrow} & [\mathcal{C}(s,s')]
\end{array}
$$

By Lemma 8.3.3 (correctness of constraints translation), there exists $p \in \mathcal{T}_{\text{pub}}$ such that $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]; \bar{i} : \mathbb{N}, c_1 : [\phi_1], \ldots, c_k : [\phi_k] \vdash_{\mathcal{D}} p : \boldsymbol{\epsilon}\ ([\mathcal{C}(s,s')]) \equiv_{\beta\mathcal{R}} \boldsymbol{\epsilon}\ ([\pi_A] \subseteq^{[s_B]}_{[s_A]} [\pi_B])$. □

8.3.5 Correctness of typing

We are now ready for our main theorem which states the correctness of our derivation translation function into the encoding signature $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$.

Theorem 8.3.18. *Assuming all derivations satisfy the **(H1)**, **(H2)** and **(H3)** properties.*

If $\dfrac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_{\mathcal{S}}}$ *, then* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi]\ \ \mathbf{WF}_{\mathcal{D}}$.

Besides, if $\dfrac{\pi_\Gamma}{\Sigma; \bar{i}/\phi; \Gamma\ \mathbf{WF}_{\mathcal{S}}}$ *, then* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_\Sigma] \vdash_{\mathcal{D}} [\pi_\Gamma]\ \ \mathbf{WF}_{\mathcal{D}}$.

Besides, if $\dfrac{\pi_t}{\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : A}$ *then* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_\Sigma]\,; [\pi_\Gamma] \vdash_{\mathcal{D}} [\pi_t] : \mathtt{D}\ \ |A|_{\Sigma;\Gamma}$.

Besides, if $\dfrac{\pi_A}{\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s}$ *, then we have both* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_\Sigma]\,; [\pi_\Gamma] \vdash_{\mathcal{D}} [\pi_A] : \mathtt{U}_{[s]}$ *and* $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_\Sigma]\,; [\pi_\Gamma] \vdash_{\mathcal{D}} [\pi_t] : \mathtt{T}_{[s]}\ \ [\pi_A]$.

Proof. By induction on the length of Σ, Γ and on π_t. For the first property:

- $\Sigma = \varnothing$. Then $[\Sigma] = \varnothing$ is well-formed in $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$.
- $\Sigma = \Sigma', (\mathtt{c}[\bar{i}/\phi] : \tau)$. By induction hypothesis, since $\Sigma'; \bar{i}/\phi; \varnothing\ \mathbf{WF}_{\mathcal{S}}$ we have both $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], \left[\frac{\pi_{\Sigma'}}{\Sigma'\ \mathbf{WF}_{\mathcal{S}}}\right]\ \mathbf{WF}_{\mathcal{D}}$ and $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_{\Sigma'}]\,; \bar{i} : \mathbb{N}, c_1 : \boldsymbol{\epsilon}\,[\phi_1], \ldots, c_k : \boldsymbol{\epsilon}\,[\phi_k] \vdash_{\mathcal{D}} [\pi_\tau] : \mathtt{U}_{[s]}$. Besides, by correctness of constraint translation, $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}] \vdash_{\mathcal{D}} [\phi_j] : \mathbb{B}$ and therefore the translation of the definition of $\mathtt{c}$ is well-typed in $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_{\Sigma'}]$. The definition of $\mathtt{c}'$ is always well-typed. We need to prove type preservation of the rewrite rule for $\mathtt{c}$. Lets assume an instance with σ of its left-hand side is well-typed in a signature extension $\Theta \supseteq \mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_{\Sigma'}]$ and a context Γ, necessarily,

$\Theta;\Gamma \vdash_{\mathcal{D}} \sigma(I_i) : \mathbb{N}$ for all i, and $\Theta;\Gamma \vdash_{\mathcal{D}} \sigma(C_i) : [\phi_i]\,\{\bar{i} \mapsto \sigma(\overline{I})\}$. The translation $[s]$ does not refer to the c_i variables, therefore

$$\vdash_{\mathcal{D}} \left[\frac{\pi_\tau}{\tau:s}\right] \left\{\bar{i} \mapsto \sigma(\overline{I}), \bar{c} \mapsto \sigma(\overline{C})\right\} : \mathtt{U}_{[s]} \left\{\bar{i} \mapsto \sigma(\overline{I}), \bar{c} \mapsto \sigma(\overline{C})\right\} = \mathtt{U}_{[s]} \left\{\bar{i} \mapsto \sigma(\overline{I})\right\}$$

so that $\mathtt{c}(\mathbf{u}_{[s]\{\bar{i}\mapsto\overline{I}\}}, \left[\frac{\pi_\tau}{\tau:s}\right] \left\{\bar{i} \mapsto \overline{I}, \bar{c} \mapsto \overline{C}\right\})\sigma$ is well-typed of type $\mathbb{C}$ and the substituted right-hand side is well-typed of type $\left(\mathtt{D}\ \mathtt{c}(\mathbf{u}_{[s]}, [\pi_\tau])\right) \left\{\bar{i} \mapsto \sigma(\overline{I}), \bar{c} \mapsto \sigma(\overline{C})\right\}$ while the left-hand side has type $\left(\mathtt{T}_{[s]}\ [\pi_\tau]\right) \left\{\bar{i} \mapsto \sigma(\overline{I}), \bar{c} \mapsto \sigma(\overline{C})\right\}$. Both are convertible, therefore the rule is well-typed.

- $\Sigma = \Sigma', (\mathtt{c}[\bar{i}/\phi] := t : \tau)$. Both the symbol declaration and the type preservation of the rewrite rule are checked the same way as above, using induction hypothesis on the translations $[\pi_t]$, $[\pi_\tau]$ of the typing derivations of t and τ.

For the second property:

- $\Gamma = \varnothing$, we only need to check the well-formedness of the context of local level variables and constraints, $\bar{i} : \mathbb{N}, c_1 : \boldsymbol{\epsilon}\,[\phi_1], \ldots, c_k : \boldsymbol{\epsilon}\,[\phi_k]$ which follows from correctness of constraint translation.
- $\Gamma = \Gamma', t : A$ and induction hypothesis proves that $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_\Sigma] \vdash_{\mathcal{D}} \left[\frac{\pi_{\Gamma'}}{\Sigma \vdash_{\mathcal{S}} \bar{i}/\phi;\Gamma'}\right]$ $\mathbf{WF}_{\mathcal{D}}$ and that $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_\Sigma]\,;[\pi_{\Gamma'}] \vdash_{\mathcal{D}} [\pi_A] : \mathtt{U}_{[s]}$ which implies well-formedness of the extended context $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_\Sigma] \vdash_{\mathcal{D}} [\pi_{\Gamma'}], x : \mathtt{T}_{[s]}\ [\pi_A]$ $\mathbf{WF}_{\mathcal{D}}$

For the third property there are 8 cases to consider.

- $t = s$, $A = s'$, $[\pi_t] = \underline{\mathbf{u}}^{[\pi_A]}_{[s],[s']}$ and $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}] \vdash_{\mathcal{D}} [\pi_t] : \mathtt{U}_{[s']}$ and $\mathtt{D}\ |s'|_{\Sigma;\Gamma} = \mathtt{D}\ \mathbf{u}_{[s']} \longrightarrow \mathtt{U}_{[s']}$.
- $t = x$ and $x : A \in \Gamma$. By definition, $x : \mathtt{T}_{[s]}\ [\pi_A] \in [\pi_\Gamma]$ and since $[\pi_t] := x$ we have $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_\Sigma]\,;[\pi_{\Gamma'}] \vdash_{\mathcal{D}} [\pi_t] : \mathtt{T}_{[s]}\ [\pi_A]$. By Lemma 8.3.15, we conclude since we have $\mathtt{T}_{[s]}\ [\pi_A] \longrightarrow \mathtt{D}\ \mathtt{c}(\mathbf{u}_{[s]}, [\pi_A]) = \mathtt{D}\ \mathtt{c}(|s|_{\Sigma;\Gamma}, [\pi_A]) = \mathtt{D}\ |A|_{\Sigma;\Gamma}$.
- $t = \Pi x : U.\, V$, $A = s_3$ and $[\pi_t] := \underline{\boldsymbol{\pi}}^{[\pi_{\mathcal{R}}]}_{[s_1],[s_2],[s_3]}\ [\pi_A]\ \lambda x : \mathtt{T}_{[s_1]}\ [\pi_A].\, [\pi_B]$ so that we have $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}] \vdash_{\mathcal{D}} [\pi_t] : \mathtt{U}_{[s_3]}$ and $\mathtt{D}\ |s_3|_{\Sigma;\Gamma} = \mathtt{D}\ \mathbf{u}_{[s_3]} \longrightarrow \mathtt{U}_{[s_3]}$ like in the first case.
- $t = \lambda x : U.\, v$, $A = \Pi x : U.\, V$ and $[\pi_t] := \lambda x\ :\ \mathtt{T}_{[s]}\ [\pi_U]\ .\ [\pi_v]$. By induction hypothesis, $\vdash_{\mathcal{D}} [\pi_U] : \mathtt{D}\ |s|_{\Sigma;\Gamma} \equiv_{\beta\mathcal{R}} \mathtt{U}_{[s]}$ and $\vdash_{\mathcal{D}} [\pi_v] : \mathtt{D}\ |V|_{\Sigma;\Gamma;x:U}$ so that $[\pi_t]$ is well-typed and has type:

$$\begin{array}{lll}
\Pi x : \mathtt{T}_{[s]}\ [\pi_U].\, \mathtt{D}\ |V|_{\Sigma;\Gamma;x:U} & \equiv_{\beta\mathcal{R}} & \Pi x : \mathtt{D}\ \mathtt{c}(\mathbf{u}_{[s]}, [\pi_U]).\, \mathtt{D}\ |V|_{\Sigma;\Gamma;x:U} \\
& \equiv_{\beta\mathcal{R}} & \Pi x : \mathtt{D}\ \mathtt{c}(|s|_{\Sigma;\Gamma}, [\pi_U]).\, \mathtt{D}\ |V|_{\Sigma;\Gamma;x:U} \\
& \equiv_{\beta\mathcal{R}} & \Pi x : \mathtt{D}\ |U|_{\Sigma;\Gamma}.\, \mathtt{D}\ |V|_{\Sigma;\Gamma;x:U} \qquad \text{by Lemma 8.3.15} \\
& = & \Pi x : [\![U]\!]_{\Sigma;\Gamma}.\, [\![V]\!]_{\Sigma;\Gamma;x:U} \qquad \text{by definition} \\
& = & [\![\Pi x : U.\, V]\!]_{\Sigma;\Gamma} = \mathtt{D}\ |\Pi x : U.\, V|_{\Sigma;\Gamma} \qquad \text{by Lemma 8.3.6}
\end{array}$$

- $t = M\ N$, $A = B\{x \mapsto N\}$ and $[\pi_t] := [\pi_M]\ \ [\pi_N]$. By induction hypothesis, $[\pi_N]$ has type $[\![U]\!]_{\Sigma;\Gamma}$ and $[\pi_M]$ has type $[\![\Pi x : U.\, B]\!]_{\Sigma;\Gamma} \equiv_{\beta\mathcal{R}} \Pi x : [\![U]\!]_{\Sigma;\Gamma}.\, [\![B]\!]_{\Sigma;\Gamma;x:A}$, therefore the application is well-typed and has type

$$\begin{array}{lll} \mathtt{D}\ |B|_{\Sigma;\Gamma;x:A}\{x \mapsto [\pi_N]\} & \equiv_{\beta\mathcal{R}} & \mathtt{D}\ |B|_{\Sigma;\Gamma}\{x \mapsto \mathtt{c}(|A|_{\Sigma;\Gamma}, x)\}\{x \mapsto [\pi_N]\} \\ & = & \mathtt{D}\ |B|_{\Sigma;\Gamma}\{x \mapsto \mathtt{c}(|A|_{\Sigma;\Gamma}, [\pi_N])\} \\ & \equiv_{\beta\mathcal{R}} & \mathtt{D}\ |B|_{\Sigma;\Gamma}\{x \mapsto |N|_{\Sigma;\Gamma}\} \\ & = & \mathtt{D}\ |B\{x \mapsto N\}|_{\Sigma;\Gamma} = \mathtt{D}\ |A|_{\Sigma;\Gamma} \end{array}$$

- $t = \mathtt{c}_{\overline{u}}$, $(\mathtt{c}[\overline{j}/\psi] := t : \tau) \in \Sigma$, $A = \tau\{\overline{u}/\overline{j}\}$ and $t := \mathtt{c}\ [\overline{u}]\ [\overline{\pi_\phi}]$. By correctness of sort translation 8.3.1, $\mathcal{D}[\mathrm{CC}^\forall_\omega] \vdash_{\mathcal{D}} [u_j] : \mathbb{N}$ for all j and by correctness of constraint translation 8.3.3, $\mathcal{D}[\mathrm{CC}^\forall_\omega]; \overline{i} : \mathbb{N}, \overline{c} : \boldsymbol{\epsilon}\ \ [\phi] \vdash_{\mathcal{D}} \left[\pi_{\phi_j}\right] : \boldsymbol{\epsilon}\ \ [\phi_j]$ for all j.

- $t = \mathtt{c}_{\overline{u}}$, $(\mathtt{c}[\overline{j}/\psi] : \tau) \in \Sigma$, $A = \tau\{\overline{u}/\overline{j}\}$ and $t := \mathtt{c}\ [\overline{u}]\ [\overline{\pi_\phi}]$

- $\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : B$ for some B such that $\phi \vdash_{\mathcal{S}} B \preceq A$ and $[\pi_t] := \begin{smallmatrix}[s']\\[s]\end{smallmatrix}\!\!\uparrow\!\!\begin{smallmatrix}[\pi_A]\\[\pi_B]\end{smallmatrix}\ \ [\pi]\ \ \left[\begin{smallmatrix}\pi_M\\ M:A\end{smallmatrix}\right]$.

□

8.4 Future work

8.4.1 Towards a confluent encoding

The $\mathtt{c}(\square, \square)$ operator is the only way to build a code of type $\mathbb{C}$ from a term of the public signature. In many ways it can be seen as a *marking* operator which annotates a term t with its (annotated) type: $\mathtt{ann}(t) := \mathtt{c}(\mathtt{ann}(A), t)$. The, necessary non-linear, elimination of identity lift was avoided (see Lemma 8.3.15) by expecting all derivations to subtype arguments, the **(H3)** property. Similarly, instead of erasing these annotations to collapse to a representation of the term t independently from its typing derivation, we could keep the annotation and consider that we are translating Dowek, Huet and Werner's marked terms [DW93] in which types are explicitly annotated, t^A.

Confluence of the non-linear system could be done using syntactical *levels* on terms. This would require to duplicate and annotate all dependent types representing derivations ($\mathtt{U}$, $\mathtt{T}$, $\mathtt{D}$) with a level ($\mathtt{U}^n$, $\mathtt{T}^n$, $\mathtt{D}^n$) and adapt all subsequent symbols and rewrite rules to retrieve a well-typed (infinite) signature. The fact that this restriction still allows to represent all typing derivations of well-typed terms in $\mathrm{CC}^\forall_\omega$ is linked to the conjecture of the existence of leveled typing derivations, as defined in [Thi20]. Confluence could be obtained, at least on a subset of terms containing the image of the translation, if the relation were proven terminating. Termination criteria sometimes require only local confluence which holds for our encoding.

8.4.2 Conservativity

Conservativity is a crucial property when embedding type systems. It essentially means that the representation of $\mathrm{CC}^\forall_\omega$ types (in particular theorems) as $\lambda\Pi_{\equiv}$ types are only

inhabited in the encoding if it was in the original system. This property guarantees that the embedding does not allow to prove too many theorems. In practice it means that proofs can be developed directly in $\lambda\Pi_{\equiv}$ and checked in the $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$ encoding, for instance using Dedukti. Such proofs can be trusted since it was proven that their existence guarantees the provability in the original system.

It obviously remains to prove conservativity of this encoding. Conservativity on normal form, is relatively straightforward *in confluent systems*. The head $f\ \overline{u}$ of the considered terms must either be a variable from the context or symbols from the public signature. Using confluence we can retrieve enough guarantees on the type of the arguments $\overline{u}$ to link the well-typedness of the term to the well-typedness of its corresponding *back translation* to $\mathrm{CC}^{\forall}_{\omega}$. A first step towards conservativity is therefore to explicit this partial correspondence between well-typed terms in the encoding public signature to terms well-typed in the original systems.

Definition 8.4.1 (Backward Translation)**.** *For Σ a $\mathrm{CC}^{\forall}_{\omega}$ signature, the* backward translation, $|\cdot|^{\Sigma}$*, is a partial function from terms of the public signature of $\lambda\Pi_{\equiv}$ to terms of $\mathrm{CC}^{\forall}_{\omega}$ inductively defined as follows:*

$$
\begin{array}{rcl@{\qquad}rcl}
|x|^{\Sigma} & := & x & |\mathtt{U}_s|^{\Sigma} & := & |s|^{\Sigma} \\
|\lambda x:A.\,t|^{\Sigma} & := & \lambda x:|A|^{\Sigma}.\,|t|^{\Sigma} & |\mathtt{T}_s\ A|^{\Sigma} & := & |A|^{\Sigma} \\
|t\ u|^{\Sigma} & := & |t|^{\Sigma}\ |u|^{\Sigma} & |\mathtt{0}|^{\Sigma} & := & 0 \\
\left|\underline{\mathbf{u}}^{\square}_{s,\square}\right|^{\Sigma} & := & |s|^{\Sigma} & |\mathtt{S}\ s|^{\Sigma} & := & |s|^{\Sigma}+1 \\
\left|\underline{\boldsymbol{\pi}}^{\square}_{\square,\square,\square}\ A\ \lambda x:\square.\,B\right|^{\Sigma} & := & \Pi x:|A|^{\Sigma}.\,|B|^{\Sigma} & |\mathtt{max}\ s\ s'|^{\Sigma} & := & \max(|s|^{\Sigma},|s'|^{\Sigma}) \\
\left|\begin{smallmatrix}\square\\ \square\end{smallmatrix}\!\!\uparrow^{\square}_{\square}\ \square\ t\right|^{\Sigma} & := & |t|^{\Sigma} & |\mathtt{Prop}|^{\Sigma} & := & \mathtt{Prop} \\
 & & & |\mathtt{Type}_u|^{\Sigma} & := & \mathtt{Type}_{|u|^{\Sigma}} \\
\multicolumn{6}{c}{|\mathtt{c}\ u_1\ \ldots\ u_n\ \pi_1\ \ldots\ \pi_k|^{\Sigma} \quad := \quad \mathtt{c}_{|\overline{u}|^{\Sigma}} \qquad \text{if } \mathtt{c}[\bar{i}/\phi]\in\Sigma,\ |\bar{i}|=n \text{ and } |\phi|=k}
\end{array}
$$

It can be extended to symbols of the private signature:

$$
\begin{array}{rcl@{\qquad}rcl}
|\mathbf{u}_s|^{\Sigma} & := & |s|^{\Sigma} & |\mathtt{D}\ A|^{\Sigma} & := & |A|^{\Sigma} \\
|\boldsymbol{\pi}\ A\ \lambda x:\square.\,B|^{\Sigma} & := & \Pi x:|A|^{\Sigma}.\,|B|^{\Sigma} & |\mathtt{c}(A,t)|^{\Sigma} & := & |t|^{\Sigma} \\
|\boldsymbol{cL}\ \lambda x:\square.\,t|^{\Sigma} & := & \lambda x.\,|t|^{\Sigma} & |\mathtt{u}(A,t)|^{\Sigma} & := & |t|^{\Sigma} \\
|\boldsymbol{cA}\ t\ u|^{\Sigma} & := & |t|^{\Sigma}\ |u|^{\Sigma} & & & \\
\multicolumn{6}{c}{|\mathtt{c}'\ u_1\ \ldots\ u_n|^{\Sigma} \quad := \quad \mathtt{c}_{|\overline{u}|^{\Sigma}} \qquad \text{if } \mathtt{c}[\bar{i}/\phi]\in\Sigma,\ |\bar{i}|=n}
\end{array}
$$

Our separation of categories of type is useful here since it forbids overlap of type between terms representing different categories of Coq objects.

Lemma 8.4.2. *Assuming $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}]$; $[\bar{i}]$, $[\phi]$, $[\Gamma] \vdash_{\mathcal{D}} t : A$.*

- *If $A \in \mathcal{F}_1$ then $|t|^{\Sigma}$ is a sort.*
- *If $A \in \mathcal{F}_2$ then $|t|^{\Sigma}$ is a constraint.*
- *If $A \in \mathcal{F}_3$ then $|t|^{\Sigma}$ is a level expression.*
- *If $A \in \mathcal{F}_5$ or $A \in \mathcal{F}_6$ then $|t|^{\Sigma}$ is a term.*

Lemma 8.4.3. *For all level expression u, $|[u]|^{\Sigma} = u$. For all sort s, $|[s]|^{\Sigma} = s$. For all signature Σ, context Γ, typing derivation $\frac{\pi}{\Sigma;\bar{i}/\phi;\Gamma\vdash_{\mathcal{S}} t:A}$, $|[\pi]|^{\Sigma} = t$ and $\left||t|_{\Sigma;\Gamma}\right|^{\Sigma} = \epsilon(t)$.*

We state here our conservativity conjectures which seem to only lack confluence of the encoding system to hold, at least on normal terms.

Conjecture 8.4.4. *Assume $\frac{\pi}{\vdash_{\mathcal{S}} \bar{i}/\phi;\varnothing\ \mathbf{WF}_{\mathcal{S}}}$ and u, v two universe expressions. Assume $p \in \mathcal{T}_{\text{pub}}$ in β-normal form such that $[\pi] \vdash_{\mathcal{D}} p : [u \leq v]$ and $\mathcal{FV}ar(p) \subseteq \bar{i}$. Then $\phi \vdash_{\mathcal{S}} u \leq v$.*

Conjecture 8.4.5 (Conservativity)**.** *Let Σ, Γ and A such that $\frac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_{\mathcal{S}}}$, $\frac{\pi_\Gamma}{\Sigma;\bar{i}/\phi;\Gamma\ \mathbf{WF}_{\mathcal{S}}}$ and $\frac{\pi_A}{\Sigma;\bar{i}/\phi;\Gamma \vdash_{\mathcal{S}} A : s}$. If $\mathcal{D}[\mathrm{CC}^{\forall}_{\omega}], [\pi_\Sigma]\,;[\pi_\Gamma] \vdash_{\mathcal{D}} t : \mathtt{T}_{[s]}\ [\pi_A]$ for some $t \in \mathcal{T}_{\text{pub}}$, then*

- *if $|t|^{\Sigma}$ is defined then $\Sigma;\bar{i}/\phi;\Gamma \vdash_{\mathcal{S}} |t|^{\Sigma} : A$;*
- *if t is in β-normal-form, then $|t|^{\Sigma}$ is defined.*

Chapter 9

Practical embedding of Coq

While its core logic was covered in the previous chapters, Coq implements many other features which makes the underlying logic quite rich and developments in this language user-friendly. For instance, it features a very general class of inductive types with various forms of universe polymorphism, fixpoint definitions, "let-in" constructions as first-class terms, and further extensions of conversion such as η-expansion. In this chapter we describe how the previously defined embedding of Coq can be extended and adapted in practice to translate these features into Dedukti, sometimes partially or approximately. The rewriting and encoding techniques presented here were used in practice in the latest version of the CoqInE tool. Any of these extra feature is a challenge on its own to embed in $\lambda\Pi_{\equiv}$. Yet that challenge must be met since these features have become quite standard and are now used without restraint, including in the standard library.

We will adopt, from now on, a less precise presentation and rather illustrate concepts through examples in the hope to better convey the ideas behind the implemented techniques. Reference to more thorough formal definitions and implementation details will be provided in the documentation of the CoqInE tool.

Finally we mention experimental results in the translation of the standard library of Coq as well as the translation of parts of the `GeoCoq` library.

9.1 The CoqInE translator

CoqInE (**Co**q **In** D**e**dukti) is an open source translator from Coq to Dedukti, freely available at `github.com/Deducteam/CoqInE`. The very first implementation of CoqInE was done by Boespflug and Burel [BB12] and was restricted to the calculus of *inductive* constructions which extends the 2-sorts CoC, with inductive types presented in more details in Section 9.3. Assaf re-implemented this tools as a Coq plugin and added support for the infinitive hierarchy of universe and cumulativity [Ass15b] as well as partial support for other features such as modules, local `let` definitions, fixpoints and floating universes which are all discussed in the following sections. Lately, the encoding techniques discussed in Chapters 5, 6 and 8 allowed to add support for universe polymorphism to CoqInE. In this chapter we discuss other significant improvements of this tool some of them made

possible by our new encoding paradigm.

Since DEDUKTI embeddings are often used for interoperability purposes, the translator was designed to be highly configurable. Several encodings of different subsets of the logic of COQ are provided and the user is given the choice to select the encoding and translation they think is the best fit for their need further down the line and according with the features used in the library they want to translate. For instance, the translation of the `GeoCoq` library, see Section 9.8, was designed so that its image is expressed in the same fragment as the, somewhat weaker, logic of ISABELLE/HOL. Therefore translated terms were expected never to rely on conversion or dependent types, therefore simplifying the translation.

In order to keep the reader closer to the actual implementation, examples in this chapter, will be given in the syntax of DEDUKTI rather than in the concise symbolic notation adopted up until now. The correspondence with previously defined notations is summed up in the following table:

$\lambda\Pi_{\equiv}$	DEDUKTI	$\lambda\Pi_{\equiv}$	DEDUKTI	$\lambda\Pi_{\equiv}$	DEDUKTI
$\Box$	`Kind`	$\mathcal{S}$	`Sort`	$\mathbb{B}$	`Bool`
$*$	`Type`	U	`Univ`	ϵ	`eps`
$\Pi x : A.\, B$	`(x : A) →B`	T	`Term`	$\top$	`true`
$A \to B$	`A →B`	$\underline{\mathbf{u}}$	`univ`	$\mathbf{I}$	`I`
$\lambda x : A.\, B$	`(x : A) ↦B`	$\underline{\pi}$	`pi`	$\wedge$	`and`
		pair	`pair`	$\uparrow$	`cast`
See deducteam.github.io		$\subseteq$	`SubType`	$\mathcal{A}$	`Axiom`
for the full syntax				$\mathcal{R}$	`Rule`
				$\mathcal{C}$	`Cumul`
$\mathbb{C}$	`Code`	$\mathbf{u}$	`cu`		
D	`Decode`	π	`cPi`		
c	`code`	cL	`cLam`	$\preceq$	`ST`
u	`uncode`	cA	`cApp`	$\forall$	`forall`

In DEDUKTI, rewrite rules are declared with an explicit context of meta-variables and meta-applications are represented as simple applications:

$\lambda\Pi_{\equiv}$ $\quad \mathtt{f}\ X\ (\lambda z.\, Y[z]) \longrightarrow Y[X]$

DEDUKTI $\quad$ `[X,Y] f X (z ↦Y z) ↪Y X.`

9.2 Renouncing left-linearity

Relying on non-linear rules is generally regarded as both unsafe and inefficient. For this reason non-linearity in encoding systems is usually avoided or kept, as much as possible, under control, as in Chapter 6 where **(H3)** was introduced to avoid the non-linear identity casts elimination, $\mathtt{u}(C, \mathtt{c}(C, T)) \longrightarrow T$.

Non-linearity is unsafe because it compromises the confluence of higher-order rewriting together with β which is essential to guarantee the subject reduction property of the encoding as well as all subsequent properties, including the conservativity of the translation.

In Chapter 4 we show that under some conditions non-linear rules may actually be safely considered, even in a non-terminating system.

The implementation of non-linearity rewriting is also usually inefficient. In the potential redex $\mathtt{u}(u, \mathtt{c}(v, t))$, for instance, it is necessary to check the convertibility of the subterms u and v to decide whether the non-linear rule applies or not. This check may require to compute the normal form of both terms to conclude that they are not convertible, preventing therefore the rule from being fired.

In practice however non-linear rewrite rules sometimes allow to use rewriting in a more convenient and sometimes even more efficient way. Indeed, the many artifacts put in place to avoid non-linearity make the translation a lot more verbose. This compromises the readability of the generated code which no longer has the same structure as the original development. For this reason, several non-linear rules are considered for the practical embedding of Coq. Examples, of such useful and somewhat safe non-linearities are detailed and discussed in the following.

9.2.1 In universe representation

The universe and constraint representation defined in Chapter 6 is convenient in theory but a bit cumbersome in practice. Inferred universe levels are often quite unnecessary large algebraic expressions. For instance, assume a context where A is a type of sort $\mathtt{Type}_i$. Then the sort inferred for the type $(A \to A \to A) \to A$ would be $\mathtt{Type}_{\max(\max(i,\max(i,i)),i)}$. It seems quite natural to rely on non-linear simplification rules such as $\mathtt{max}\ i\ i \longrightarrow i$ and $\mathtt{max}\ (\mathtt{S}\ i)\ i \longrightarrow \mathtt{S}\ i$ which are admissible and greatly simplify algebraic expressions in some cases.

In the particular case of the current implementation of Coq, the considered universe algebraic expressions are of depth 1, either $\max(u, v)$, i or $i+1$, and constraints are upper-bounded with a level variable. This means that it is actually possible to rely on a universe representation closer to the one introduced in Definition 6.2.16.

Note however that because we rely on a complete algorithm for explicit constraints checking, we do not *need* these rewrite rules, they merely allow smaller representation of universe expression. The cumbersome $\mathtt{Type}_{\max(\max(i,\max(i,i)),i)}$ may be carried over in the rest of the translation and will have the same properties than the minimal version $\mathtt{Type}_i$. In fact it is even possible to compute this minimal version at translation time and type $(A \to A \to A) \to A$ directly with $\mathtt{Type}_i$ although it requires to build less trivial proofs of the rules predicates. While the predicate $\boldsymbol{\mathcal{R}}(u, v, \mathtt{max}\ u\ v)$ is immediately proven with $\mathtt{refl}\ (\mathtt{max}\ u\ v)$, the translation would need to compute a proof of $\boldsymbol{\mathcal{R}}(i, i, i)$ such as $\mathtt{pair}\ (\mathtt{refl}\ i)\ (\mathtt{refl}\ i)$.

In fact, the proof of constraints may also be simplified using non-linear rules on constraint types such as $i \leq i \longrightarrow \top$, $i \leq \mathtt{S}\ i \longrightarrow \top$ or $C \wedge C \longrightarrow C$. These rules provide much simpler normal forms for constraints and allow, for instance, the simple object representation `pi i i i I A (_ ↦A)` for the product type $A \to A$.

9.2.2 In subtyping predicates

Subtyping predicates extend the $\mathcal{C}$ predicate on sort representations to arbitrary types in order to allow the general `cast` operator. While explicitly inhabiting sort predicates remains manageable, inhabiting subtyping predicates can be quite verbose, even with a simple reflexivity constructor, `refl`. For instance, if $A \equiv_\beta B$ and $\vdash_{\mathcal{S}} t : A$, then its translation `cast` $s_A\ s_B\ A\ B$ (`refl` A) t requires to translate A twice.

The non-linear ST-π rule allows a subtyping predicate between two products with convertible domain to directly reduce to the subtyping of their codomains, as expected. This allows the translator to provide a proof of the level constraints on the ultimate codomains only. Assuming $A \underset{\beta}{\longrightarrow\!\!\!\!\!\rightarrow} C \underset{\beta}{\longleftarrow\!\!\!\!\!\leftarrow} B$ then we have:

$$\begin{array}{lll}\left[\Pi x : A.\, \mathtt{Type}_i \leq \Pi x : B.\, \mathtt{Type}_j\right] & := & \left(\underline{\boldsymbol{\pi}}^{\square}_{\square,\square,\square}\ [A]\ \lambda x.\, \underline{\mathbf{u}}_{[i],\square}\right) \subseteq \left(\underline{\boldsymbol{\pi}}^{\square}_{\square,\square,\square}\ [B]\ \lambda x.\, \underline{\mathbf{u}}_{[j],\square}\right) \\ & \longrightarrow\!\!\!\!\!\rightarrow & (\boldsymbol{\pi}\ |C|\ \lambda x.\, \mathbf{u}_i) \preceq (\boldsymbol{\pi}\ |C|\ \lambda x.\, \mathbf{u}_j) \\ & \longrightarrow & \forall\ \lambda x : \mathbb{C}.\, (\mathbf{u}_i \preceq \mathbf{u}_j) \longrightarrow \mathbf{u}_i \preceq \mathbf{u}_j \longrightarrow i \leq j\end{array}$$

This rule is used in practice to the encoding without any issue noticed. It allows to simplify the translation since a subtyping instance is now only required to provide the set of necessary level constraints.

This non-linearity can be extended with the elimination of identity subtyping predicates, $C \preceq C \longrightarrow \mathbf{I}$. Note that this extra rule generates, however, critical pairs with the other rules which requires to successively consider other non-linear rules such as $\mathcal{C}(S, S) \longrightarrow \mathbf{I}$ and $U \leq U \longrightarrow \mathbf{I}$.

9.2.3 In code representations

The two critical rules of the encoding are the elimination of imbricated coding and uncoding operators:

$$\mathtt{c}(C, \mathtt{u}(C, T)) \longrightarrow T \qquad \text{(c-u)}$$

$$\mathtt{u}(C, \mathtt{c}(C, T)) \longrightarrow T \qquad \text{(u-c)}$$

The first rule remains well-typed when linearized, it is therefore safe to use its efficient linearized version rather than the original. The second rule u-c, however, cannot.

The first rule was necessary to collapse several imbricated liftings of a term t to ensure it has a unique representation: ${}^{s'}_{\square}\!\uparrow^{B}_{\square}\ \square\ ({}^{\square}_{s}\!\uparrow^{\square}_{A}\ \square\ t) \equiv_{\beta\mathcal{R}} {}^{s'}_{s}\!\uparrow^{B}_{A}\ \square\ t$. The second rule is required to eliminate identity casts: ${}^{\square}_{\square}\!\uparrow^{A}_{A}\ \square\ t \equiv_{\beta\mathcal{R}} t$. This last property is inherently non-linear but not required for the encoding to be correct as it can be avoided by systematically subtyping arguments, **(H3)**.

However in practice, this rule allows to avoid this costly casting. Even though it can be costly, it makes the translation much more compact and readable. The set of critical pairs of the encoding system with the extra non-linear rule is extended for the four pairs in Lemma 8.1.3 to the following 8 pairs which all remain joinable:

Critical pair	Closing diagram
$\underset{\text{c-u}}{\longleftarrow}; \overset{2}{\underset{\text{u-}\lambda}{\longrightarrow}}$	$\underset{\text{c-u}}{\longleftarrow}; \underset{\text{c-u}}{\longleftarrow}; \underset{\text{c-u}}{\longleftarrow}; \underset{\beta}{\longleftarrow}; \underset{\text{c-}\lambda}{\longleftarrow}$
$\underset{\text{c-a}}{\longleftarrow}; \overset{1}{\underset{\text{c-u}}{\longrightarrow}}$	$\underset{\text{u-a}}{\longrightarrow}; \underset{\text{c-u}}{\longrightarrow}; \underset{\text{c-u}}{\longrightarrow}; \underset{\text{c-u}}{\longrightarrow}$
$\underset{\text{c-a}}{\longleftarrow}; \overset{1}{\underset{\text{c-}\lambda}{\longrightarrow}}$	$\underset{\beta}{\longrightarrow}; \underset{\beta}{\longleftarrow}; \underset{\text{c}\beta}{\longleftarrow}$
$\underset{\text{u-a}}{\longleftarrow}; \underset{\text{u-}\lambda}{\longrightarrow}$	$\underset{\text{c}\beta}{\longrightarrow}; \underset{\beta}{\longrightarrow}; \underset{\beta}{\longleftarrow}$

New critical pair	Closing diagram
$\underset{\text{c-u}}{\longleftarrow}; \overset{2}{\underset{\text{u-c}}{\longrightarrow}}$	$=$
$\underset{\text{u-c}}{\longleftarrow}; \overset{2}{\underset{\text{c-u}}{\longrightarrow}}$	$=$
$\underset{\text{u-c}}{\longleftarrow}; \overset{2}{\underset{\text{c-}\lambda}{\longrightarrow}}$	$\underset{\text{u-c}}{\longleftarrow}; \underset{\text{u-c}}{\longleftarrow}; \underset{\beta}{\longleftarrow}; \underset{\text{u-}\lambda}{\longleftarrow}$
$\underset{\text{u-a}}{\longleftarrow}; \underset{\text{u-c}}{\longrightarrow}$	$\underset{\text{c-a}}{\longrightarrow}; \underset{\text{u-c}}{\longrightarrow}; \underset{\text{u-c}}{\longrightarrow}$

9.3 Inductive constructions

9.3.1 Inductive Types and Constructors

Inductive types have become a rather predominant feature of the Coq system. They are a powerful extension of the Calculus of Constructions, introduced by Coquand and Paulin-Mohring [CP90, PM93] and used extensively since then. Since they quickly become quite complex to reason with, if considered precisely, they are purposely missing from our approximation $\mathrm{CC}^{\forall}_{\omega}$ which focuses instead on the presentation of universe polymorphism.

Inductive types allow to define new data-types in the signature from a context Γ_C of *constructors* that are used to build inhabitants of the defined types, Γ_I. Beside constructors, inductively defined types come with a *destructor* allowing to perform *structural induction*, or *match*, on constructed inhabitants. Inductive types can be mutually defined if they reference one another in the type of their constructors. Mutually defined inductive types have a number p of *parameters* so that all the inductive types and the types of their constructors are all products headed with the same p quantifications, $\forall(\mathtt{c} : T) \in \Gamma_I \cup \Gamma_C, T = \Pi x_1 : A_1 \ldots \Pi x_p : A_p.\, T'$. Finally, in order to be well-formed and accepted in Coq, inductive types must satisfy several other criteria: inductive types must build a type so their type must be a so-called "arity" (product types which "ultimate" codomain is a sort), constructors must build inhabitants of one of the defined inductive types and their type must satisfy the *nested positivity condition*. Using the notations from the Coq documentation [Tea], inductive definitions are written $\mathbf{Ind}[p](\Gamma_I := \Gamma_C)$.

Example 1: Below are the definition of usual inductively defined data structures:

Natural numbers: $\mathbf{Ind}[0]\left(\left[\mathtt{nat} : \mathtt{Type}_0\right] := \begin{bmatrix}\mathtt{O} : \mathtt{nat} \\ \mathtt{S} : \mathtt{nat} \to \mathtt{nat}\end{bmatrix}\right)$

Polymorphic lists:

$$\mathbf{Ind}[1]\left(\left[\mathtt{list} : \Pi A : \mathtt{Type}_0.\, \mathtt{Type}_0\right] := \begin{bmatrix}\mathtt{nil} : \Pi A : \mathtt{Type}_0.\, \mathtt{list}\ A \\ \mathtt{cons} : \Pi A : \mathtt{Type}_0.\, A \to \mathtt{list}\ A \to \mathtt{list}\ A\end{bmatrix}\right)$$

Even and odd predicates on `nat`:

$$\mathbf{Ind}[0]\left(\begin{bmatrix}\mathtt{even} : \mathtt{nat} \to \mathtt{Prop} \\ \mathtt{odd} : \mathtt{nat} \to \mathtt{Prop}\end{bmatrix} := \begin{bmatrix}\mathtt{even0} : \mathtt{even}\ \mathtt{O} \\ \mathtt{evenS} : \Pi n : \mathtt{nat}.\, \mathtt{odd}\ n \to \mathtt{even}\ (\mathtt{S}\ n) \\ \mathtt{oddS} : \Pi n : \mathtt{nat}.\, \mathtt{even}\ n \to \mathtt{odd}\ (\mathtt{S}\ n)\end{bmatrix}\right)$$

Inductive types must all be well-sorted and their constructors must be well-typed in the context extended with the inductive type declarations. Therefore they may simply be translated as successive symbols declarations.

```
nat : Univ (type 0).
O : Term (type 0) nat.
S : Term (type 0) nat →Term (type 0) nat.
list : (A:Univ (type 0)) →Univ (type 0).
nil : (A:Univ (type 0)) →Term (type 0) (list A).
cons : (A:Univ (type 0)) →Term (type 0) A →Term (type 0) (list A)
                                    →Term (type 0) (list A).
even : Term (type 0) nat →Univ prop.
odd : Term (type 0) nat →Univ prop.
even0: Term prop (even O).
evenS: (n:Term (type 0) nat) →Term prop (odd n)
                                    →Term prop (even (S n)).
oddS : (n:Term (type 0) nat) →Term prop (even n)
                                    →Term prop (odd (S n)).
```

Note however that none of the inductive well-formedness conditions performed by Coq are guaranteed by the encoding or the translation. For now it is necessary to rely on an external check in order to trust the consistency of a translated signature with inductive definitions. However the encoding aims at providing a correct (well-typed) representation.

9.3.2 Destructors

The *destructor* of an inductive type is a very general scheme that allows to use inhabitants of an inductive type by assuming they are necessarily built using one of the type's constructors. It encompasses both the usual *destruction rule* of inductively defined propositions such as disjunctions or conjunctions, and *pattern matching* on elements of inductively defined sets such as natural numbers or lists.

$$\begin{array}{l}\texttt{match}\ m\ \texttt{as}\ x\ \texttt{in}\ \mathtt{I}\ \overline{q}\ \overline{a}\ \texttt{return}\ P\ \texttt{with}\\ (\mathtt{c}_1\ x_{11}\ \ldots\ x_{1p_1}) \Longrightarrow f_1\\ \ldots\\ (\mathtt{c}_n\ x_{n1}\ \ldots\ x_{np_n}) \Longrightarrow f_n\\ \texttt{end}\end{array}$$

This construction allows to inhabit the type $P\{a_i \mapsto t_i\}\{x \mapsto m\}$ depending on the term m inhabiting the instance $\mathtt{I}\ \overline{q}\ t_1\ \ldots\ t_n$ of the inductive type $\mathtt{I}$. It requires to provide an exhaustive list of *case* functions: $f_i : P\{a_i \mapsto t_i\}\{x \mapsto \mathtt{c}_i\ x_{i1}\ \ldots\ x_{ip_i}\}$ for all i and terms t_i (depending on the $x_{i1}\ \ldots\ x_{ip_i}$) such that $\mathtt{c}_i\ x_{i1}\ \ldots\ x_{ip_i} : \mathtt{I}\ \overline{q}\ t_1\ \ldots\ t_n$. These terms t_i can simply be inferred from the type declaration of each constructor, assuming that p_i is such that it is fully applied. Note that the *parameters* $\overline{q} = q_1, \ldots, q_p$, are uniformly fixed.

The translation of such constructions requires a dedicated `match` function:

```
match_I :
  q1:P1 →..→qp:Pp →
  s : Sort →
  P : (a1:A1 →..→ak:Ak →Term sI (I q1 ...qp a1 ...ak) →Univ s) →
  case_c1 : (x11:B11 →..→x1p1:B1p1 →
             Term s (P t11 ...t1k (c1 q1 ...qp x11 ...x1p1))) →
  ... →
  case_cn : (xn1:Bn1 →..→xnpn:Bnpn →
             Term s (P tn1 ...tnk (cn q1 ...qp xn1 ...xnpn))) →
  a1:A1 →..→an:An →
  x : I q1 ...qp a1 ...an →Term s (P a1 ...an x).
```

Where the term t_{ij} are such that for all i, $\mathtt{c}_i\ q_1 \dots q_p\ x_1 \dots x_k : \mathtt{I}\ q_1 \dots q_p\ t_{i1} \dots t_{ik}$.

Occurrences of this destructor are then translated to the well-typed

$$\mathtt{match\_I}\ [q_1] \dots [q_p]\ [s]\ (\lambda \overline{a} x.\, [P])\ (\lambda \overline{x_1}.\, [f_1]) \dots (\lambda \overline{x_n}.\, [f_n])\ [t_1] \dots [t_n]\ [m]$$

Conversion in the Calculus of inductive Constructions is extended with the ι-rule that provide destructors with some computational power:

$$\mathtt{match}\ \mathtt{c}_k\ \overline{q}\ \overline{a}\ \mathtt{with}\ (\mathtt{c}_i\ \overline{x_i} \Longrightarrow f_i)_{1 \leq i \leq n}\ \mathtt{end} \equiv_\iota f_k\{\overline{x_k} \mapsto \overline{a}\}$$

This is quite naturally represented with a rewrite rule for each constructor $\mathtt{c}_i$:

```
[ q1 ... qp s P case_c1 ... case_cn a1 ... ak xi1 ... xipi]
match_I q1 ...qp s P case_c1 ...case_cn a1 ...ak (ci xi1 ...xipi)
↪case_ci xi1 ...xipi.
```

9.3.3 Brackets versus S-injectivity

It is not clear that the rule for constructor elimination preserves subject reduction. Indeed, the inferred type for the left-hand side is $\mathtt{Term}\ \mathtt{s}\ (\mathtt{P}\ a_1 \dots a_k\ (\mathtt{c}_i\ x_1 \dots x_{p_i}))$ while the right hand side is of type $\mathtt{Term}\ \mathtt{s}\ (\mathtt{P}\ t_1 \dots t_k\ (\mathtt{c}_i\ x_1 \dots x_{p_i}))$ for the particular terms t_i such that $\mathtt{c}_i\ x_1 \dots x_k : \mathtt{I}\ q_1 \dots q_p\ t_1 \dots t_k$. A sufficient condition for the rule to be type-preserving is therefore that $a_i \equiv_{\beta\mathcal{R}} t_i$ for all i. And indeed, for the left-hand side to be well-typed, it must be the case that (respectively) the inferred and expected type for the last argument of `P` are convertible:

$$\mathtt{Term}\ s_I\ (\mathtt{I}\ q_1 \dots q_p\ t_1 \dots t_k) \equiv_{\beta\mathcal{R}} \mathtt{Term}\ s_I\ (\mathtt{I}\ q_1 \dots q_p\ a_1 \dots a_k)$$

Dedukti's algorithm for type preservation checking is currently not powerful enough to infer, on its own, that this constraint ensures $a_i \equiv_{\beta\mathcal{R}} t_i$ and therefore that the generated rule is type preserving. It is only allowed to destruct constraints that are headed by a so-called *static* symbol: such that no rewrite rule are headed by that symbol. The `I` symbol is static here but the `Term` symbol is not because several rewrite rules are `Term`-headed. In fact `Term` is not even injective since

$$\mathtt{Term}\ (\mathtt{u}\ 0)\ t \equiv_{\beta\mathcal{R}} \mathtt{Term}\ (\mathtt{u}\ 1)\ (\mathtt{cast}\ 1\ 2\ (\mathtt{u}\ 0)\ (\mathtt{u}\ 1)\ t)$$

while we have neither $\texttt{u}\ 0 \equiv_{\beta\mathcal{R}} \texttt{u}\ 1$ nor convertibility of the second arguments.

However our translation guarantees that, despite having rewrite rules, the functional symbol `Term` is $\{2\}$-*injective*. The general definition of S-injectivity for a set S of argument positions as well as more detailed use cases in the context of rewrite rule checking can be found in Jui-Hsuan Wu and Blanqui's work [Wu19, Bla20].

This means that it is "injective in its second argument": whenever $s \equiv_{\beta\mathcal{R}} s'$ and $\texttt{Term}\ s\ t \equiv_{\beta\mathcal{R}} \texttt{Term}\ s'\ t'$ then necessarily $t \equiv_{\beta\mathcal{R}} t'$. This property is sufficient to guarantee that all the `match` elimination rules are in fact type preserving. From the typing constraint of the left-hand side, we can deduce that

$$\texttt{I}\ q_1 \dots q_p\ t_1 \dots t_k \equiv_{\beta\mathcal{R}} \texttt{I}\ q_1 \dots q_p\ a_1 \dots a_k$$

and then by injectivity of `I`, this can only hold when $a_i \equiv_{\beta\mathcal{R}} t_i$ for all i which in turn obviously guarantees that

$$\texttt{Term s}\ (\texttt{P}\ a_1 \dots a_k\ (\texttt{c}_i\ x_1 \dots x_{p_i})) \equiv_{\beta\mathcal{R}} \texttt{Term s}\ (\texttt{P}\ t_1 \dots t_k\ (\texttt{c}_i\ x_1 \dots x_{p_i}))$$

For Dedukti to accept this rule we need to rely on a hint mechanism called *brackets*, or *dot patterns* in previous versions [BB12]. For instance, the translation of destructor elimination rules for `evenS` is:

```
[s P case_even0 case_evenS case_oddS n]
match_I s P case_even0 case_evenS case_oddS {S n} (evenS n)
                                                  ↪case_evenS n.
```

The expression in brackets is

- interpreted as the `S` n expression when checking rule type preservation;
- interpreted as a fresh meta-variable, Z, applied to all available locally bounded variables (which is none in our example), when using the rule for matching;
- whenever the rule is fired, it is systematically checked at runtime that the bracket's match is in fact convertible with the provided expression `S` n.

This solution allows to check type-preservation but at the cost of a "safety net" which is checked at runtime every time the rule is used and even though that check is in theory not necessary because conversion is guaranteed by typing. Note that this mechanism is a bit different from conditional rewriting. In this mechanism, if the rule ever matched a term with a substitution σ such that $Z\sigma \not\equiv_{\beta\mathcal{R}} (\texttt{S}\ n)\sigma$, then, not only would the rule not be fired, but an exception would also be raised, since by assumption, the matched term cannot be well-typed.

These systematic checks explain some of the performance issues in the translation of Coq developments that heavy rely on inductive-based conversion. Implementing the inference or declaration of S-injectivity to be properly used at rule checking could bring dramatic improvement to this way of encoding computational rules which is a major feature of Coq.

9.4 Fixpoints

9.4.1 Definition

Fixpoints are constructions allowing to build proofs or functions that may recursively refer to other instances of themselves. The computation of an applied fixpoint may therefore require further computation of the same fixpoint which compromise, the termination of reduction. Besides, a general fixpoint scheme may allow circular proofs which compromise the soundness of the system. To prevent both of these issues, Coq enforces recursive calls in fixpoints to be on a *structurally smaller* set of arguments.

For instance, addition on natural numbers can be defined as the following fixpoint

```
Definition plus :=
  fix plus (n m:ℕ ) {struct n} : ℕ
      := match n with 0 ⇒ m | S n' ⇒ S (plus n' m) end.
```

Reusing the notations of [Tea], a fixpoint is written

$$\text{Fix } f_i\{f_1/k_1 : A_1 := t_1, \ldots, f_n/k_n : A_n := t_n\}$$

where the k_i are the indices of a structurally decreasing argument of the mutually recursive function symbols f_i. This means that each of the A_i typing the f_i must be headed with at least k_i products: $A_i = \Pi x_1 : A_i^1 . \ldots \Pi x_{k_i} : A_i^{k_i} . T_i$ and occurrences of f_i in t_j must have its k_i-th arguments structurally smaller than x_{k_j}. The above fixpoint is well-typed in the context Γ if each A_i has a sort s_i and each t_i has type A_i *in the extended context* $\Gamma, f_1 : A_1, \ldots, f_n : A_n$, therefore allowing occurrences of $f_1, \ldots, f_n$ in the t_i.

Our example is well-typed since we have $k = 1$ and the first argument of the recursive call to `plus` in t is n' which is bound by a structural decomposition (`match`) of n and

$$\texttt{plus} : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \vdash_{\mathcal{S}} \lambda nm : \mathbb{N}. (\texttt{match } n \texttt{ with } \texttt{0} \Rightarrow \texttt{0}, \texttt{S } n' \Rightarrow \texttt{plus } n' m \texttt{ end}) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$

Fixpoints naturally add computational power to Coq through the ι rule:

$$\text{Fix } f_i\{F\}\ a_1\ \ldots\ a_{k_i} \quad \underset{\iota}{\longrightarrow} \quad t_i\{f_i \mapsto \text{Fix } f_i\{F\}\}\ a_1\ \ldots\ a_{k_i}$$

only if a_{k_i} is headed by a constructor. If the previously stated condition of F are verified, then this reduction can be proven strongly normalizing together with β and the other reduction rules of Coq. Note that the following rule would obviously not be terminating.

$$\text{Fix } f_i\{F\} \quad \underset{\iota}{\longrightarrow} \quad t_i\{f_i \mapsto \text{Fix } f_i\{F\}\}$$

This rewrite rules guarantees that in our example:

$$\begin{array}{rcl} \texttt{plus (S } x\texttt{) } y & := & \text{Fix } (\texttt{plus} := \ldots)\ (\texttt{S } x)\ y \\ & \underset{\iota}{\longrightarrow} & \texttt{match S } x \texttt{ with } \ldots \texttt{ end}\{\texttt{plus} \mapsto \text{Fix } (\ldots)\} \\ & \underset{\iota}{\longrightarrow} & \texttt{S (plus } x\ y\texttt{)}\{\texttt{plus} \mapsto \text{Fix } (\ldots)\} \\ & = & \texttt{S } (\text{Fix } (\ldots)\ x\ y) \ =: \ \texttt{S (plus } x\ y\texttt{)} \end{array}$$

which does not reduce further as the Fix operator is applied to variables which are not headed by a constructor and therefore prevent the ι rule to be further used.

While embedding fixpoints in Dedukti, we need to be very careful to reflect this computational rule while guaranteeing that it can only be fired if the k_i-th argument is headed by a constructor, otherwise instances of fixpoint will be non-terminating, even when restricting to head normalization.

9.4.2 A first implementation

In Boespflug and Burel [BB12], a fixpoint occurring in a context $\Gamma = y_1 : B_1, \ldots, y_p : B_p$ is represented with $2n$ fresh symbols defined *beforehand* and quantifying over the context Γ. Note the anonymous duplication of the last argument in the $\mathtt{f'}_i$:

```
f1 : y1 : B1 →...→yp : Bp →x11 : A1 →...→x1k1 : A1k1 →T1
...
fn : y1 : B1 →...→yp : Bp →xn1 : A1 →...→xnkn : Ankn →Tn

f'1: y1 : B1 →...→yp : Bp →x11 : A1 →...→x1k1 : A1k1 →A1k1 →T1
...
f'n: y1 : B1 →...→yp : Bp →xn1 : A1 →...→xnkn : Ankn →Ankn →Tn
```

We then define duplicating rewrite rules

```
[...] f1 y1 ...yp x11 ...x1k1 →f'1 y1 ...yp x11 x1k1 x1k1
...
[...] fn y1 ...yp xn1 ...xnkn →f'n y1 ...yp xn1 xnkn xnkn
```

Finally, for all $1 \leq i \leq n$, the $A_i^{k_i}$ must be an inductive type `I` $a_1 \ \ldots \ a_{q_i}$ and for all constructors `c` of `I`, the following rewrite rule is added

```
fi y1 ...yp xi1 ...xiki (c a1 ...aqi) ↪ti
```

where `ti` is the translation of the i-th body, t_i, which may freely refer to the previously defined translations `fi` of the f_i. This implementation is correct with respect to the two following properties of fixpoints:

- all well-typed fixpoint occurrence is translated into a well-typed term, although this term must be considered in a well-typed extension of the signature;
- the computation rule is reflected in the encoding but is limited to constructor guarded k_i-th argument.

This implementation has however two major issues that make it very inconvenient to use in practice for Coq development heavily relying on fixpoint computations.

- It lifts local fixpoint definitions out of their local context which needs to be encoded as arguments in the definition of the fixpoints. This duplication does not interfere with the computation of the fixpoint but it makes its translation quite cumbersome and nearly impossible to read by a human.

- Two identical fixpoints defined separately would have nonconvertible translations. For instance two identical definitions of the addition define convertible terms in Coq, yet are translated to distinct symbols with identical sets of rules, but not convertible.

In order to fix these two issues, we need a way to represent fixpoints directly as a Dedukti term inside a local context and with the same correctness of computation power.

9.4.3 First-class fixpoints

We describe here a very technical encoding of fixpoints as first-class constructions. We detail the implementation with as much precision as we think is required for the reader to get an idea why this implementation is correct. This implementation heavily relies on the public/private separation of the encoding to guarantee the conservativity of the representation.

We start by considering a signature defining a private type of unary natural numbers to represent indices and length as well as a guarding function defined on the translation of all well-sorted types.

```
_N : Type.
_0 : _N.
_S : _N → _N.

Guarded? : Type.
guarded : Guarded?.
def guarded? : s : Sort → Ind : Univ s → Term s Ind → Guarded?.

SingleArity : Type.
def SA : _N → s : Sort → Univ s → SingleArity.

MutualArity : _N → Type.
MAnil : MutualArity _0.
MAcons : n : _N → SingleArity → MutualArity n → MutualArity (_S n).

def MutualFixpoint (n : _N) (MA : MutualArity n) : Type.
```

The `SingleArity` type represents pairs of a well-sorted type (which should be an arity even though it is not guaranteed by the encoding) together with an index, the position of the structurally decreasing argument. If $\vdash_{\mathcal{S}} A : s$, then (k, A) is translated as `SA` $[k]$ $[s]$ $[A]$.

The `MutualArity` type represents lengthed lists of `SingleArity`. Instances are built using the usual constructors of lists.

The `MutualFixpoint` type represents a `MutualArity` together with a list of bodies with the expected type. We will assume a single constructor `MF` expecting an integer n and a `MutualArity` of size n such that `MF n [ SA k1 s1 A1 ; ... ; SA kn sn An ]` has type

```
t1 : (Term s1 A1 →..→Term sn An →Term s1 A1) →
... →
tn : (Term s1 A1 →..→Term sn An →Term sn An) →MutualFixpoint
```

This constructor therefore allows to build a representation for the list of bodies $\{f_1/k_1 : A_1 := t_1, \ldots, f_n/k_n : A_n := t_n\}$ of a mutual fixpoint. The declared types enforce that all provided arities have a sort and of courses that there are as many bodies as arities and that the bodies have the expected type: they define an inhabitant of A_i but may recursively refer to all A_j to do so.

Using this constructor, the representation of $\{f_1/k_1 : A_1 := t_1, \ldots, f_n/k_n : A_n := t_n\}$ is therefore:

```
MF (MACons (SA k1 s1 A1) (... MACons (SA kn sn An) MANil )...)
  (f1 : Term s1 A1 ↦...↦fn : Term sn An ↦t1)
  ...
  (fn : Term s1 A1 ↦...↦fn : Term sn An ↦tn)
```

It now only remains only to define the projections, Fix $f_i\{\ldots\}$, of fixpoints encoded this way:

```
def fix_proj :
  n : N →
  MA: MutualArity n →
  MutualFixpoint n MA →
  i : N →
  Ith_Arity n MA i.
```

where `Ith_Arity n MA i` simply computes, using rewrite rules, the i-th arity in the `MA` list of arities. The actual implementation relies on code representation to ensure the irrelevance of the arities typing sorts.

As an example, the usual definition of the `plus` function

```
Definition plus : ℕ → ℕ → ℕ :=
  fix f (n:ℕ ) (m:ℕ ) {struct n} : ℕ := match n with 0 ⇒ m | S n' ⇒ S (plus n' m) end.
```

would be translated, as follow:

```
nat : Univ 0.
Z : Term 0 nat.
S : Term 0 nat →Term 0 nat.
def arr (A:Univ 0) : Univ 0 := prod 0 0 A (x:Term 0 A ↦A).

plus : Term 0 nat →Term 0 nat →Term 0 nat
:= fix_proj (_S _0)
     (MAcons _0
       (SA _0 (univ 0) (arr nat (arr nat nat)))
       MAnil
     )
     (MF ...(plus ↦x ↦y ↦match_nat x Z (x' ↦S (plus x' y)))
     _0.
```

9.4.4 Hacking through the quadratic blowup

To properly define a well-typed constructor `MF` as described before and used in CoqInE is a bit technical. However, since the type-rewriting techniques involved could probably be reused in other embedding in the $\lambda\Pi_{\equiv}$, we rely on simpler examples to provide the main ideas behind its definition.

Assume a quite standard implementation of lengthed lists in Dedukti:

```
A : Type.

N : Type.
0 : N.
S : N →N.

LList : N →Type.
nil : LList 0.
cons : n:N →LList n →A →LList (S n).
```

In order to build a list of length 3, we need to use

```
def mylist_3 := cons (S (S 0)) (cons (S 0) a (cons 0 nil a)) a.
```

which is a bit costly since we need to repeat the successive length in each occurrence of the `cons` constructors. This may seem negligible here but the unary integer representation actually forces to rely on a term of size n^2 to represent a list of length n. Besides, this immediately becomes a problem when considering, for instance, the case of polymorphic lists :

```
T : A →Type.
PLList : A →N →Type.
pnil : a:A →PLList a 0.
pcons : a:A →n:N →PLList a n →T a →PLList a (S n).
```

In order to build a polymorphic list on length n, one needs to repeat the same polymorphic argument in all the $n+1$ successive constructors.

A work around this issue is to define the following well-typed functions:

```
def PLListAux : A →N →N →Type.
[a,n ] PLListAux a n 0    ↪PLList a n.
[a,n,m] PLListAux a n (S m) ↪T a →PLListAux a n m.

def acc : (a:A) →(n:N) →(m:N) →PLListAux a n m
                                  →T a →PLListAux a (S n) m.
[n ,a,PL ] acc a n 0 PL      ↪Pcons a n PL.
[n,m,a,PL,e] acc a n (S m) PL e ↪acc a n m (PL e).

def smart_cons : (a:A) →(n:N) →PLListAux a n n.
[a ] smart_cons a 0    ↪Pnil a.
[a,n] smart_cons a (S n) ↪acc a n n (smart_cons a n).
```

The defined `smart_cons` functional symbol can be successively applied once to the polymorphic parameter a, once to a size integer argument, n, and eventually to n arguments of type `T` a. This application has type `PLList` a n and it rewrites, as expected, to the fully expanded representation of the polymorphic lengthed list formed with the last n arguments.

```
a : A.
e1 : T a. e2 : T a. e3 : T a.
#INFER smart_cons a (S (S (S 0))).
```

```
(; (T a) -> (T a) -> (T a) -> PLList a (S (S (S 0))) ;)
#EVAL smart_cons a (S (S (S 0))) e1 e2 e3.
(; Pcons a (S (S 0)) (Pcons a (S 0) (Pcons a 0 (Pnil a) e1) e2) e3 ;)
```

Finally, we consider a structure of several lists sharing the same size and polymorphic parameter. The parameters to build an object in this structure will have to be duplicated in the structure constructor and each of the lists constructors. A way to avoid this duplication is to use, once again, a smart constructor that provides a list constructor of the expected type:

```
struct : Type.
pair : (a:A) →(n:N) →PLList a n →PLList a n →struct.

def smart_pair : (a:A) →(n:N) →
  (PLListAux a n n →PLList a n) →
  (PLListAux a n n →PLList a n) →
  struct.

[a,n,l1,l2] smart_pair a n l1 l2
↪pair   a n (l1 (smart_cons a n)) (l2 (smart_cons a n)).

#EVAL smart_pair a (S (S 0)) (c ↦c e1 e2) (c ↦c e2 e3).
(; pair   a  (S  (S 0))
       (Pcons a (S 0) (Pcons a 0 (Pnil a) e1) e2)
       (Pcons a (S 0) (Pcons a 0 (Pnil a) e2) e3) ;)
```

While that last optimization may seem superfluous, it becomes critical for instance when the types of the second list may depend on the first one. In the case of fixpoints, for instance, a first list is provided with the indexed arities (A_i/k_i) and then a second list must be provided with the bodies of the fixpoints definitions which expected type depends on the previous list.

The following usual fixpoint definition

```
Definition f : ℕ → ℕ :=
  fix  f (n:ℕ ) {struct n} : ℕ := match n with 0 ⇒ 0 | S n' ⇒ S (g n') end
  with g (n:ℕ ) {struct n} : ℕ := n
  for f.
```

is therefore translated to the (relatively) concise

```
def f : Term set nat →Term set nat :=
  fixproj 2
    (c ↦c (SA 0 set (prod set set set I nat (n : Term set nat ↦nat)))
          (SA 0 set (prod set set set I nat (n : Term set nat ↦nat))) )
    (c ↦c (f0↦ g0↦ (n:Term set nat)↦
             match_nat set (n0 : Term set nat ↦nat)
               0 (n' : Term set nat ↦S (g0 n')) n
           )
           (f0↦ g0↦ (n:Term set nat)↦ n) )
    0.
```

Finally a similar technique can be used to chain the construction of conjunctions proofs or usage of transitivity in the proof of cumulativity. As an example, the encoding offers the following constructors:

```
c1 : eps A. c2 : eps B. c3 : eps C.
(; Duplication of B and C ;)
pair A (and B C) c1 (pair B C c2 c3) : eps (and A (and B C)).
(; No duplication ;)
smart_pair (Some A) c1 (Some B) c2 (Some C) c3 None : eps (and A (and B C)).

c1 : eps (Cumul s0 s1).
c2 : eps (Cumul s1 s2).
c3 : eps (Cumul s2 s3).
(; Duplication of s0 and s2 ;)
trans s0 s2 s3 (trans s0 s1 s2 c1 c2) c3 : eps (Cumul s0 s2).
(; No duplication ;)
smart_trans s0 (Some s1) c1 (Some s2) c2 (Some s3) c3 None : eps (Cumul s0 s2).
```

The technique to properly define `smart_pair` and `smart_trans` as well-typed symbols is not detailed here but is similar to the ones previously described. Typing ensures that the smart constructors may only be used to build proofs that could be without them. In fact fully applied smart constructors actually rewrite to an expression using exclusively the safe constructors. This representation is therefore redundant but identical parts were all duplicated from a single term and are therefore shared in memory. In the context of encodings, these proofs are discarded anyway to ensure their non-relevance and their expanded form does not even need to be computed in the process of typechecking translations.

9.5 Universe polymorphisms

9.5.1 Template universe polymorphism

Template universe poymorphism was introduced by Herbelin as an implicit form of universe polymorphism for inductive types in their *uniform parameters*: those that specified in all occurrences of the constructors types. It allows inductive types to be universe polymorphic in some of their parameters which type is an arity while keeping monomorphic constructors.

For instance, the previously defined `list` is actually implicitly template polymorphic when defined in Coq. This means that for all universe level i, if $\vdash_{\mathcal{S}} A$: `Type`$_i$ then we have $\vdash_{\mathcal{S}}$ `list` A : `Type`$_i$:

```
Check list ℕ : Set.      Check list Type : Type.
```

Note that this polymorphism is unannotated. The sort of `list` A is deduced from the inferred type of A. In particular, it is not necessarily stable and may decrease by substitution or along reduction. Constructors, however, are monomorphic and any type parameter A_i has a fixed (floating) sort s_i enforced to be above the sort of the corresponding type argument in all later occurrences of the constructor.

This allows to build lists from other lists but forbids for instance to apply `cons` to `list` or to itself:

```
Check cons (cons a nil) nil : list (list A).
Fail Check cons list.
Fail Check cons (@cons).
(* Fails indeed: cannot ensure that "Type@{max(Set,list.u0+1)}"
   is a subtype of "Type@{list.u0}"). *)
```

In previous implementations of CoqInE, this form of polymorphism was simply ignored in the translation and static global universe levels were used:

```
option: Univ 2 →Univ 2.
Some : A : Univ 2 →Term 2 A →Term 2 (option A).
```

Surprisingly this approximation actually works in practice and allows to type-check a couple of files from Coq's prelude, showing that this feature is not necessary to start using type-parameterized inductive types. However the translation eventually becomes ill-typed since, for instance, $\vdash_{\mathcal{S}}$ `option nat` : $\texttt{Type}_0$ in Coq but $\vdash_{\mathcal{D}}$ `option (lift 0 2 nat)` : `Univ 2` in Dedukti.

A first optimization consists in using a sort parameter

```
option : s  : Sort →Univ s →Univ s
Some :   s  : Sort →A : Univ s →Term s A →Term s (option s A)
```

This solution works better since now $\vdash_{\mathcal{D}}$ `option 0 nat` : `Univ 0`.

However we have: $(\lambda A : \texttt{Type}_2.\ \texttt{option}\ A)\ \texttt{nat} \equiv_\beta \texttt{option nat}$ and yet

$$\texttt{(A ↦option 2 A) (lift 0 2 nat)} \xrightarrow[\beta]{} \texttt{option 2 (lift 0 2 nat)} \not\equiv_{\beta\mathcal{R}} \texttt{option 0 nat}$$

Besides constructors should be monomorphic to avoid building inhabitants of the inductive type for higher levels than itself.

A solution is to provide a cast-erasing rewrite rule on the sort-polymorphic inductive type while keeping a static monomorphic constructor:

```
def option : s : Sort →Univ s →Univ s.
[s,s',A] option s (lift s' _ A) ↪lift s' s (option s' A).
Some : A:Univ 2 →Term 2 A →Term 2 (option 2 A).
```

The choice of `option`'s sort argument is now "irrelevant" in some sense. The type obtained by lifting `nat`, `Term (option 2 (lift 0 2 nat))` reduces to `Term 0 (option 0 nat)`.

Finally, using code representation we actually rely on a private version of the inductive type just like we do for universe polymorphic declarations:

```
def option : s : Sort →Univ s →Univ s
option' : Code →Code
[s,A] code _ (option s A) ↪option' (code s A).
Some : A : Univ 2 →Term 2 A →Term 2 (option 2 A)
```

This ensures `Term s' (option s' (lift s s' A))` is convertible with `Term s (option s A)` since both reduce to `Decode (option' (code (univ s) A))`.

9.5.2 Cumulative universe polymorphism

In the particular case of universe polymorphic inductive definitions, Coq recently introduced a new form of universe polymorphism for inductive definitions which allows a less constraining convertibility criteria in some cases. The sort parameters of *cumulative universe polymorphic inductive types* in Coq are either:

- **Invariant**: this is the default kind of universe polymorphic parameters and the default in non-cumulative polymorphic declarations. All invariant sort parameters must be equal for instances to be convertibles. These parameters are translated as simple sort parameters in Dedukti without difficulty.
- **Irrelevant**: as the sort parameter of template polymorphic lists discussed before. They are ignored when deciding whether two instances are convertible and can there be rewritten into private representations erasing them so that they are irrelevant in the translation.
- **Covariant**: when they occur in the type of the constructors rather than in the parameters of the defined inductive type. In that case, subtyping is extended covariantly, meaning that a type instance is a subtype from an other if all of its covariant sort parameters are pairwise smaller. This was translated by extending the subtyping predicate in the encoding system with rewrite rules reducing the subtyping between instances to the corresponding covariant parameters subtyping constraints.

9.6 Local let-in

Coq allows not only variables declarations to be added to a context but also *variable definitions*: $(x := t : T)$. Context well-formedness is checked with

$$\frac{\Sigma; \Gamma \ \mathbf{WF}_{\mathcal{S}} \qquad \Sigma; \Gamma \vdash_{\mathcal{S}} t : T \qquad x \notin \Gamma}{\Sigma; \Gamma, (x := t : T) \ \mathbf{WF}_{\mathcal{S}}} \ \text{W-Local-Def}$$

and such definitions are introduced with a dedicated term construction $\texttt{let}\ x := t\ \texttt{in}\ u$:

$$\frac{\Sigma; \Gamma \vdash_{\mathcal{S}} t : T \qquad \Sigma; \Gamma, (x := t : T) \vdash_{\mathcal{S}} u : U}{\Sigma; \Gamma \vdash_{\mathcal{S}} \texttt{let}\ x := t\ \texttt{in}\ u : U\{x \mapsto t\}} \ \text{Let}$$

This construction enrich the conversion with the so-called ζ-reduction:

$$\texttt{let}\ x := t : T\ \texttt{in}\ u \underset{\zeta}{\longrightarrow} u\{x \mapsto t\}$$

Dedukti does not have such a construction and the only way, that we know of, to encode this construction is to rely on so called `let`-lifting. If $t|_p = (\texttt{let}\ x := u : U\ \texttt{in}\ v)$ is a subterm of t at position p in a derivable judgment $\Sigma; \bar{i}/\phi; \varnothing \vdash_{\mathcal{S}} t : s$ or $\Sigma; \bar{i}/\phi; \varnothing \vdash_{\mathcal{S}} t : \tau$. Then we write $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ the typed local context at this instance, i.e. the

list of typed binding between the root of t and the occurrence of the `let—in`. We define the *lifting* of that occurrence *outside* of t as the definition of an extra symbol:

$$\texttt{let\_x} : \Pi x_1 : A_1 . \ldots \Pi x_n : A_n . U := u$$

well-typed in $\Sigma; \bar{i}/\phi; \varnothing$, so that $\Sigma' := \Sigma, (\texttt{let\_x}[\bar{i}/\phi] : \Pi x_1 : A_1 . \ldots \Pi x_n : A_n . U := u)$ is well-formed. We then translate the following judgment

$$\Sigma'; \bar{i}/\phi; \varnothing \vdash_{\mathcal{S}} t[\ v\{x \mapsto \texttt{let\_x}_{\bar{i}}\ x_1\ \ldots\ x_n\}\ ]_p : \tau$$

where the `let—in` occurrence has been replaced with its body, v, in which all occurrences of x have been replaced with the newly defined symbol with the same universe variables and applied to the same arguments in the same order. This is well-typed since

$$\Sigma'; \bar{i}/\phi; x_1 : A_1, \ldots, x_n : A_n \vdash_{\mathcal{S}} \texttt{let\_x}_{\bar{i}}\ x_1\ \ldots\ x_n : U$$

and $\texttt{let\_x}_{\bar{i}}\ x_1\ \ldots\ x_n \xrightarrow[\delta]{} u$ to match $x \xrightarrow[\zeta]{} u$ in $t|_p$.

As an example, the following well-typed Coq definition

```
Definition t : eq ℕ 0 0 := let x := ℕ : Type in let y := 0 : ℕ in  eq_refl x y.
```

is translated into

```
def let_x (i : Lvl) : Univ i := lift (type 0) (type i) I nat.
def let_y (i : Lvl) : Term (type 0) nat := 0.
def t (i    : Lvl) : Term (type 0) (eq nat 0 0)
                                  := eq_refl (let_x i) (let_y i).
```

This convenient `let—in` construction is first-class in Coq and often used extensively in developments, in particular to name intermediate goals in interactively defined proofs. Therefore many occurrences of `let—in` do not actually require the convertibility relation associated with the bound variable. For instance, we have

$$\dfrac{\dfrac{\ldots}{\vdash_{\mathcal{S}} 2 : \mathbb{N}} \qquad \dfrac{\ldots}{x := 2 : \mathbb{N} \vdash_{\mathcal{S}} x + x : \mathbb{N}}}{\vdash_{\mathcal{S}} \texttt{let}\ x := 2 : \mathbb{N}\ \texttt{in}\ x + x : \mathbb{N}}$$

However the $x := 2 : \mathbb{N}$ definition was not required to derive $\vdash_{\mathcal{S}} x + x : \mathbb{N}$. A simple $x : \mathbb{N}$ declaration would have sufficed. In that particular case, the `let—in` could have therefore been replaced with a simple cut:

$$\dfrac{\dfrac{\dfrac{\ldots}{x : \mathbb{N} \vdash_{\mathcal{S}} x + x : \mathbb{N}}}{\vdash_{\mathcal{S}} \lambda x : \mathbb{N} . x + x : \Pi x : \mathbb{N} . \mathbb{N}} \qquad \dfrac{\ldots}{x := 2 : \mathbb{N} \vdash_{\mathcal{S}} x + x : \mathbb{N}}}{\vdash_{\mathcal{S}} (\lambda x : \mathbb{N} . x + x)\ 2 : \mathbb{N}}$$

When translating Coq developments that are meant to be interoperable with other logics that do not support the ζ-reduction, all occurrences of `let—in` must be of this kind and it is therefore possible to configure CoqInE to systematically translate them as β-redexes rather than the previously described elaborate (and verbose) `let`-lifting. If the rest of the translation fails, the translator is however able to backtrack and eventually translate it as a well-typed `let`-lifting.

9.7 Minimal universe constraints

In Coq proofs, universe levels are not usually declared beforehand. Instead the user may rely on typical ambiguity to omit the level in `Type` occurrences which are then replaced with anonymous or universe polymorphic (locally bound) levels at type-checking. The universe constraints required to type check a term are not declared either. Coq must work backwards as it tries to type a term t and infer on the fly a set of constraints ϕ large enough so that t is indeed typable but guaranteeing that this set remains *stratifiable*.

Then Coq performs a minimization pass to keep only a minimal set of constraints large enough to infer all the constraints that were previously inferred when expanding the typical ambiguity in the user-defined terms. For performance, but also user readability, it removes duplicates or trivial constraints as well as constraints that can be inferred from others using transitivity. For instance, a set of inferred constraints $\{i \leq j, j \leq k, i \leq k\}$ will be minimized to $\{i \leq j, j \leq k\}$ the last constraints being inferrable from the first two, it is dropped.

The translator, CoqInE, on the other hand relies on Coq's type checking and must therefore work with the minimized set of constraints. It is required to rebuild the forgotten constraints with the transitivity symbol, `tr`, every time the translation requires a constraint missing from the minimized set. This is done with a simple breadth-first search where the absence of cycle guarantees that constraints can only be used once and therefore runs in linear time.

9.8 In practice

The previous iterations of the CoqInE tool already allowed to translate Coq's standard library. However only the first few generated files, all from the `Init` folder in Coq's prelude, were well-typed. The main reason for this is that the encoding used to rely on an approximation of template polymorphism and first-class fixpoints, both features implicitly occurring quite often, even in the standard library. Yet, the issues were usually complicated corner cases and most standard examples passed, therefore allowing to showcase the tool. Most of these problematic features were in fact absent from the Matita system, a simpler variant of Coq, which is why Assaf chose to target this system first.

Our work on universe embedding as well as the practical techniques described in this section allowed to extend CoqInE to correctly represent and translate a much large subset of the Coq system. In particular, the current implementation now offers limited support for universe polymorphism, both from template polymorphic inductive types and "true" polymorphism. The covered subset is now large enough to allow the translation of Coq's prelude as well as numerous files from the standard library.

In particular we were able to translate the following selections of Coq libraries into Dedukti, each showing off a different use case of the tool.

- The `GeoCoq` project (geocoq.github.io/GeoCoq) is a Coq implementation of several geometry formalisms and developments. Our first target was the formalization

of the original proofs from the first book of Euclid's Elements. The corresponding library is sizable but not too complicated, relying exclusively on the simplest features of Coq. In collaboration with Boutry, Narboux and Thiré, we were able to fully translate it, including the Pythagorean theorem, using a simple version of the encoding. It is at the moment one of the largest proof library encoded in Dedukti. In particular this development allowed some post-processing treatment of the files in Dedukti files.

- Our second target, from the same project, was the implementation of the first part of Tarski's "Metamathematische Methoden in der Geometrie". In collaboration with Boutry, we were able to fully translate it but the generated files could only be checked up to Chapter 14 mostly because of performance issues in the translation of fixpoints. The Coq development heavily relies on co-linearity and co-planarity algorithms which were implemented directly in Coq rather than in the meta language of tactics. The corresponding terms therefore required a lot of (Coq) computation to be reduced or checked. In order to check the translated Dedukti terms, it was required to perform the same computations in the correct but slower encoding. It even required more computation since Dedukti type checking algorithm is not as clever as Coq's. Removing this particular tactic from the development allowed to go as far as Chapter 16. This development does not features universe polymorphism, apart from some requirements from the standard library, but it showcases the correct representation of inductive types and fixpoints algorithms which are heavily used in this geometrical development.

- Finally a selection of 30 files from the standard library and 18 short hand-written examples was translated and checked. This last batch is quite small but heavily relies on the previously unavailable advanced features of Coq such as universe polymorphism. This selection is used as unit tests for the tool.

A benchmark of sizes, translation and checking times indicators can be found in Figure 9.1. These numbers are a mere order of magnitude of the tool's performance and should not be taken for an accurate benchmark for several reasons. Among the several bias we point two out. In both `GeoCoq` benchmarks, only the project's files are checked with Coq, the standard library is omitted, even though it is then translated and rechecked in Dedukti. In the case of `GeoCoq`'s Tarski benchmark, several files from Coq's standard library are included even though sometimes only a single definition is required from them. In order to minimize the translation, a non negligible part of these Coq files is "pruned" out after translation and only a minimalized version is checked in Dedukti. All developments are open source and referenced on Deducteam's web page https://github.com/Deducteam/.

It is worth mentioning that experiments to use the new encoding techniques to encode the arithmetic library of Matita were not conclusive yet because of performance issues since the new encoding relies on more reduction steps for convertibility. These issues raise interesting questions about the implementation of Dedukti and ways to have a tighter control over the reduction strategy.

	GeoCoq Euclid	GeoCoq Tarski	Coq std
# Files	237	124	48
LoC (w/o comments/empty)	20200	81000	6040
Size	1.4 Mb	2.46 Mb	420 Kb
Compressed size	112 Kb	391 Kb	68.6 Kb
Translation size	497 Mb	1.0 Gb	24.3 Mb
Compressed translation	14.5 Mb	33 Mb	520 Kb
Translation time	24m26s	10m58s	5.7s
Coq checking time	7m17s	7m06s	17.6s (Byte.v alone: 7.4s)
Dedukti checking time	10m38s	1h52 (CoincR.dk: 56m) (CongR.dk: 46m)	10.9s (Byte.dk: 4.9s)

Figure 9.1: Translation benchmark of several Coq developments

Finally the new features of CoqInE, together with some other tools, are expected to allow some interoperability between Coq developments and other systems. For instance, two models of hyperbolic geometry were expressed, one within the Isabelle/HOL proof assistant, based on Poincaré's disc model [SMB20] and the other within Coq based on Klein's disc model. An isomorphism between these two models was devised by the authors of these developments and could be formalized. Instead of manually replicating one development into the other proof system, the Dedukti representation could be used, either as an intermediary representation to translate from Coq to Isabelle, or as a middle ground in which both developments could be expressed in a common logical subset of the logics of both proof assistants.

Conclusion

Conclusion

In the first part of this manuscript, we relied on van Oostrom's decreasing diagrams to provide several confluence criteria for higher-order rewrite systems, both left-linear and non-left-linear, considered together with the usual functional reduction.

In the linear case, our approach consisted in using the multi-step *orthogonal extension* of the reduction $\beta \cup \mathcal{R}$ in order to show the existence of a decreasing diagram for any local peak. This technique is standard to prove the confluence of left-linear systems without critical pair. We show that it extends, in particular, to the case where *orthogonal critical pairs* have decreasing diagrams.

We also introduced a new criteria for the confluence of non-left-linear systems restricted to a subset of well-layered terms featuring a confined first-order layer. The syntactical layering of the considered terms is designed to forbid problematic interactions between non-left-linear rules and higher-order rules. Such interactions were exploited to fabricate confluence counter-examples but the terms of the pure λ-calculus that they rely on are not usually well-typed. However, because confluence on untyped terms must first be proven in order to consider a well behaved typing system, our syntactical restriction plays the role of an intermediate step providing just enough guarantees for confluence to hold while allowing as many as possible, if not all, well-typed terms.

While non-left-linear rewrite rules are extremely useful, they have been avoided so far in higher-order typed settings to which confluence is the keystone most other properties rely on. Our work now allows non-left-linearity to be used and its confluence to be addressed by considering syntactical restrictions on terms.

In the second part of this manuscript, we introduced several techniques, both theoretical and practical, to correctly encode universes and type systems relying on them in $\lambda\Pi_{\equiv}$. We especially focused on the translation of an extension of the Calculus of Constructions featuring some form of universe polymorphism. The encoding system is type-preserving assuming its confluence and the translation mechanism relying on derivation rather than terms was proven correct. These techniques were implemented in practice, along with practical clever tricks that were permitted by them, in order to improve the CoqInE translator, targeting the Coq system. Finally we succeeded in translating the first several sizable Coq libraries into a Dedukti encoding.

Universe polymorphism has long been a roadblock in the integration of Coq numerous

developments into Dedukti. Indeed complex features such as inductive types, fixpoints and universe polymorphism occur even in the standard library, as early as in the prelude subset which could therefore not even be translated using a simple encoding of the calculus of constructions. Our contributions, both theoretical and practical now allow to associate Coq's formalism and developments to current work on interoperability. There even has been some effort in the minimization of universes in Coq development using the work of Thire [Thi20].

Related and future work

Regarding confluence of term-rewriting

We believe this work still requires some improvement to be easily applied to higher-order rewrite systems used in practical applications such as logical system encoding. Some ways to further extend our confluence result in the non-linear case were already mentioned in Section 4.6. In particular the layering condition can most likely be relaxed so as to include much more terms than those encompassed by our, quite strict, layering over $\mathbb{N}$. While most difficulties regarding layering should already be addressed in this manuscript, extending Theorem 3.5.9 to orthogonal sub-rewriting is bound to be a challenging but rewarding task. The resulting theorem would indeed be quite useful in practical non-terminating embedding.

The introduced syntactical restrictions may appear cumbersome at first. For some systems, however, they turn out to be quite natural. Sorts, for instance, are inherently different from λ-terms and therefore their representations in an encoding may be syntactically separated without any risk to threaten neither the correctness nor the conservativity of said encoding. In fact, such systems already rely on typing to ensure they are no more expressive than the system they represent. The syntactical restriction is but an extra constraint to consider in order to offer this guarantee.

Restricting the set of considered terms is hard to avoid when considering non-linear and higher-order rules together. We believe that our syntactical restriction can be extended to encompass a much wider class of terms. For instance, the attentive reader probably noticed that our main confluence theorem does not directly apply to our encoding of Coq's universe polymorphism since the single non-left-linear rule considered is such that terms cannot easily be layered.

Indeed, well-typed instances of the non-left-linear rule may very well substitute the non-linear meta-variable with other redexes of that same rule, breaking the critical hypothesis **(H4)**. In fact there are even such instances that are in the image of the translation. Until that gap is bridged, confluence and conservativity of our embedding is unlikely to be proven and therefore it cannot be trusted as a type checker of Coq's logic.

It does however allow to prove the confluence, see Lemma 6.1.1, of the simpler system in Figure 9.6 embedding the infinitely sorted $\mathrm{PTS}^{\preceq}$. While restricting subtyping to sorts only conveniently allows to put them in a distinct, lower layer, this does not work for the more general subtyping of CTS unless all terms can be layered. It was, in fact, conjectured

[Thi20] that CTS derivation trees can be annotated with a layering satisfying conditions quite close to ours. If that conjecture turned out to be true, then it could be used to prove the confluence of our embedding using duplicated versions of the coding and uncoding operators (`c` and `u`) for each level.

It finally remains to see how the introduced syntactical conditions can be used in practice. Their simplicity allows both well-layeredness of terms and layer preservation of rules to be easily checked before type checking without any significant extra cost. An implementation of these checks as well as simple criteria for hypothesis **(H0)** through **(H5)** and a unification algorithm for the computation of critical pairs could all be added to type checkers such as DEDUKTI for a more convenient practical use of safe non-left-linear systems. The stratification could probably even allow a more efficient computation since, going down one level, some rules become unavailable and can be disregarded for faster computed reduction steps.

Regarding logical system encoding

In order to achieve a completely satisfying theoretical encoding of COQ in DEDUKTI, the following issues still need to be addressed.

- The remaining non-linearities in the rewrite system should either be eliminated or better controlled so that the encoding defines a proven confluent rewrite system on a subset of terms large enough to encompass the image of the translation.
- Conservativity of our derivation tree translation should then be proven. The challenge here is that we cannot use COQ's normalization property to restrict ourselves to normal forms in $\lambda\Pi_{\equiv}$ since these normal forms would not necessarily be in the public encoding. Indeed, private symbols were precisely introduced to provide canonical but unsafe normal forms. A more promising lead would be to adapt Assaf's proof of conservativity of PTS which did not require normalization and relied instead on a back translation which was made explicit in our case in Definition 8.4.1.
- The polymorphism embeddings mentioned in 6.4.2 were not considered in detail in this manuscript as they did not seem to fit our particular needs. It is however likely that these techniques would shine in different contexts and should therefore be further investigated and tested out.

There still remain several features of COQ that we did not manage to represent in DEDUKTI in a completely satisfactory way. We mention a couple of them here.

- COQ admits the η conversion rule: if $\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} t : \Pi x : A.\, B$ then $t \equiv_{\eta} \lambda x : A.\, t\ x$. This conversion rule is only defined on well-typed terms and therefore has to be considered in a given signature and context. Considering this conversion as a rewriting relation in an untyped setting yields either a non-terminating η-expansion or the η-reduction which is not confluent together with β. To reflect this conversion in our encoding, it is possible to reflect it only on the code representation of terms $|t|_{\Sigma;\Gamma}$ which can only represent well-typed terms.

- Modules and functors are a key component is the organization of Coq code yet they are not easily translated in Dedukti. Rather than representing functors using Dedukti terms and rewrite rules, it seems more reasonable to extend Dedukti with support for similar feature.
- Co-inductive types, co-recursive functions, and positivity criteria for inductive types were, for the most part, ignored in our development. Including these features would allow to support a larger part of the available practical Coq developments. It is however likely that their implementation would require to introduce even more complex encoding, possibly non-terminating or relying on further extensions of rewriting such as conditional rewriting.
- The more features supported by the encoding, the more complex and specialized the encoding is bound to become. Instead, it should be possible to focus on developing techniques to transform Coq proofs so that they rely on a minimal set of features before translating them. This would allow the translation to be more easily reused in different contexts, in particular where the avoided features are not available.

Finally, there have been many recent developments in the encoding of other complex type systems into Dedukti which could probably reuse some of the several encoding techniques introduced in this manuscript. For instance, a translation of a subset of PVS was introduced by Gilbert [Gil17, Gil18] and could probably be extended to rely on an adapted form of proof irrelevance in the expression of PVS's predicate subtyping. The use of predicates rather than functional symbols was particularly useful to support universe algebraic expression but it could still be adopted in the encoding of PTS and CTS since they allow to separate the type system encoding from the sort structure. This modular way of encoding may even simplify the encoding of complex $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{C}$ relations and is necessary if one of them is not functional.

Appendix

Bibliography

Bibliography

[Abe13] Andreas Abel. *Normalization by Evaluation: Dependent Types and Impredicativity*. habilitation, Ludwig-Maximilians-University Munich, 2013.

[ADJL16] Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud, and Jiaxiang Liu. Untyped Confluence in Dependent Type Theories. In *Proceedings Higher-Order Rewriting Workshop*, Proc. Higher-Order rewriting Workshop, Porto, Portugal, June 2016. Easy-Chair.

[Ass14] Ali Assaf. A calculus of constructions with explicit subtyping. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *20th International Conference on Types for Proofs and Programs, TYPES 2014, May 12-15, 2014, Paris, France*, volume 39 of *LIPIcs*, pages 27–46. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[Ass15a] Ali Assaf. Conservativity of embeddings in the lambda pi calculus modulo rewriting (long version). *CoRR*, abs/1504.05038, 2015.

[Ass15b] Ali Assaf. *A framework for defining computational higher-order logics*. PhD thesis, École polytechnique, Paris, 2015.

[AYT09] Takahito Aoto, Junichi Yoshida, and Yoshihito Toyama. Proving confluence of term rewriting systems automatically. In Ralf Treinen, editor, *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings*, volume 5595 of *Lecture Notes in Computer Science*, pages 93–102. Springer, 2009.

[Bar81] Hendrik Pieter Barendregt. *The lambda calculus : its syntax and semantics*. Studies in logic and the foundations of mathematics. North-Holland, Amsterdam, New-York, Oxford, 1981.

[Bar93] H. P. Barendregt. *Lambda Calculi with Types*, page 117–309. Oxford University Press, Inc., USA, 1993.

[Bar99] Bruno Barras. *Auto-validation d'un système de preuves avec familles inductives*. PhD thesis, Université de Paris VII, Paris, France, 1999.

[BB12] Mathieu Boespflug and Guillaume Burel. Coqine: Translating the calculus of inductive constructions into the $\lambda\pi$-calculus modulo. *CEUR Workshop Proceedings*, 878, 06 2012.

[BFG94] F. Barbanera, M. Fernandez, and J.H. Geuvers. Modularity of strong normalization and confluence in the algebraic lambda-cube. In *Proceedings 9th Annual IEEE Symposium on Logic in Computer Science (Paris, France, July 4-7, 1994)*, pages 406–415, United States, 1994. IEEE Computer Society.

[BG05] Bruno Barras and Benjamin Grégoire. On the role of type decorations in the calculus of inductive constructions. In C.-H. Luke Ong, editor, *Computer Science Logic, 19th International Workshop, CSL 2005, 14th Annual Conference of the EACSL, Oxford, UK, August 22-25, 2005, Proceedings*, volume 3634 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 2005.

[Bla01] Frédéric Blanqui. *Type theory and rewriting.* PhD thesis, Université de Paris XI, Orsay, France, 2001.

[Bla06] Frédéric Blanqui. Termination and confluence of higher-order rewrite systems. *CoRR*, abs/cs/0610064, 2006.

[Bla20] Frédéric Blanqui. Type Safety of Rewrite Rules in Dependent Types. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, volume 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[BPT17] Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. The next 700 syntactical models of type theory. In *Certified Programs and Proofs (CPP 2017)*, pages 182 – 194, Paris, France, January 2017.

[CD07] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2007.

[CH85] Thierry Coquand and Gérard Huet. Constructions: A higher order proof system for mechanizing mathematics. In Bruno Buchberger, editor, *EUROCAL '85*, pages 151–184, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[CH86] T. Coquand and Gérard Huet. The calculus of constructions. Technical Report RR-0530, INRIA, May 1986.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5(2):56–68, 06 1940.

[Con04] Evelyne Contejean. A certified ac matching algorithm. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications*, pages 70–84, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Coq85] Thierry Coquand. *Une théorie des constructions*. PhD thesis, Université de Paris VII, Paris, France, 1985. Thèse de doctorat dirigée par Huet, Gérard Mathématiques. Informatique Paris 7 1985.

[Coq86] T. Coquand. An analysis of Girard's paradox. Technical Report RR-0531, INRIA, May 1986.

[Cou02] Judicaël Courant. Explicit universes for the calculus of constructions. In Victor A. Carreño, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics*, pages 115–130, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[CP90] Thierry Coquand and Christine Paulin. Inductively defined types. In Per Martin-Löf and Grigori Mints, editors, *COLOG-88*, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[CTW20] Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. The Taming of the Rew: A Type Theory with Computational Assumptions. *Proceedings of the ACM on Programming Languages*, 2020.

[Cur34] Haskell B. Curry. Functionality in combinatory logic. In *Proceedings of the National Academy of Sciences of the United States of America*, pages 584–590, 1934.

[dB83] N. G. de Bruijn. *AUTOMATH, a Language for Mathematics*, pages 159–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[DHKP98] Gilles Dowek, Thérèse Hardin, Claude Kirchner, and Frank Pfenning. Unification via Explicit Substitutions: The Case of Higher-Order Patterns. Research Report RR-3591, INRIA, 1998. Projet COQ, Projet PARA, PROTHEO.

[Dow19] Dowek, Gilles and Jouannaud, Jean-Pierre and Liu, Jiaxiang. Confluence in (un)typed higher-order type theories ii. draft. hal-, INRIA, march 2019. available from http://dedukti.gforge.inria.fr/.

[DW93] Gilles Dowek and Benjamin Werner. On the definition of the eta-long normal form in type systems of the cube. In *Informal Proceedings of the Workshop on Types for Proofs and Programs*, 1993.

[Dyb98] Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65:2000, 1998.

[EKO19] Jörg Endrullis, Jan Willem Klop, and Roy Overbeek. Decreasing diagrams for confluence and commutation, 2019.

[FcT19] Gaspard Férey and François Thiré. Proof irrelevance in lambdapi modulo theory, 2019. Submitted to TYPES 2019.

[Fel13] Bertram Felgenhauer. Rule labeling for conuence of left-linear term rewrite systems. In *International Workshop on Confluence*, pages 23–27, 2013.

[FJ19a] Gaspard Férey and Jean-Pierre Jouannaud. Confluence in (un)typed higher-order theories i, 2019. Submitted to TOCL.

[FJ19b] Gaspard Férey and Jean-Pierre Jouannaud. Confluence in (un)typed higher-order theories i, 2019. Submitted to MFPS 35.

[GAA+13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A Machine-Checked Proof of the Odd Order Theorem. In Sandrine Blazy, Christine Paulin, and David Pichardie, editors, *ITP 2013, 4th Conference on Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 163–179, Rennes, France, July 2013. Springer.

[Gen20a] Guillaume Genestier. Encoding Agda Programs using Rewriting. *5th International Conference on Formal Structures for Computation and Deduction*, 2020.

[Gen20b] Guillaume Genestier. *Termination Criteria for Higher-Order Rewriting with Dependent Types*. Phd thesis, LSV, ENS Paris-Saclay, Université Paris-Saclay, France, 2020.

[Gil17] Frédéric Gilbert. Proof certificates in PVS. In *ITP 2017 - 8th International Conference on Interactive Theorem Proving*, volume 10499 of *ITP 2017: Interactive Theorem Proving*, pages 262–268, Brasilia, Brazil, September 2017. Springer.

[Gil18] Frédéric Gilbert. *Extending higher-order logic with predicate subtyping: application to PVS*. PhD thesis, Université Sorbonne Paris Cité, France, 2018.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, These dÉtat, Paris VII, 1972.

[GN91] Herman Geuvers and Mark-Jan Nederhof. Modular proof of strong normalization for the calculus of constructions. *Journal of Functional Programming*, 1(2):155–189, 1991.

[Gog94] Healfdene Goguen. The metatheory of UTT. In Peter Dybjer, Bengt Nordström, and Jan M. Smith, editors, *Types for Proofs and Programs, International Workshop TYPES'94, Båstad, Sweden, June 6-10, 1994, Selected Papers*, volume 996 of *Lecture Notes in Computer Science*, pages 60–82. Springer, 1994.

[Gon05] Georges Gonthier. A computer-checked proof of the four colour theorem. Technical report, Microsoft Research, Cambridge, 2005.

[Gon08] Georges Gonthier. The four colour theorem: Engineering of a formal proof. *Notices of the AMS*, 55(11):1382–1393, 2008.

[Hal05] Thomas C. Hales. A proof of the kepler conjecture. *Annals of Mathematics*, 162(3):1065–1185, 2005.

[Her05] Hugo Herbelin. Type inference with algebraic universes in the calculus of inductive constructions. Manuscript, 2005.

[HHP93a] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of Association for Computing Machinery (JACM)*, pages 194–204, 1993.

[HHP93b] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, January 1993.

[Hin64] J. Roger Hindley. *The Church-Rosser Property and a Result in Combinatory Logic*. PhD thesis, University of Newcastle upon Tyne, 1964.

[Hin69] J. R. Hindley. An abstract form of the Church-Rosser theorem. i. *J. Symb. Log.*, 34(4):545–560, 1969.

[How80] William Alvin Howard. The formulae-as-types notion of construction. In Haskell Curry, Hindley B., Seldin J. Roger, and P. Jonathan, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism.* Academic Press, 1980.

[HP89] Robert Harper and Robert Pollack. Type checking, universe polymorphism, and typical ambiguity in the calculus of constructions draft. In J. Díaz and F. Orejas, editors, *TAPSOFT '89*, pages 241–256, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[HP91] Robert Harper and Robert Pollack. Type checking with universes. *Theoretical Computer Science*, 89(1):107 – 136, 1991.

[Hue72] Gérard Huet. *Constrained Resolution: A Complete Method for Higher-Order Logic*. PhD thesis, Jennings Computer Center, 1972.

[Hue80] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, October 1980.

[JL12a] Jean-Pierre Jouannaud and Jianqi Li. Church-Rosser properties of normal rewriting. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 21st Annual Conference of the EACSL, CSL 2012, September*

3-6, 2012, Fontainebleau, France, volume 16 of *LIPIcs*, pages 350–365. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[JL12b] Jean-Pierre Jouannaud and Jiaxiang Liu. From diagrammatic confluence to modularity. *Theor. Comput. Sci.*, 464:20–34, 2012.

[JvO09] Jean-Pierre Jouannaud and Vincent van Oostrom. Diagrammatic confluence and completion. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming*, pages 212–222, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Kd89] J.W. Klop and R.C. de Vrijer. Unique normal forms for lambda calculus with surjective pairing. *Information and Computation*, 80(2):97 – 113, 1989.

[Klo80] Jan Willem Klop. *Combinatory reduction systems.* PhD thesis, CWI tracts, 1980.

[KvOvR93] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121(1):279 – 308, 1993.

[Las12] Marc Lasson. *Realizability and parametricity in Pure Type Systems.* Theses, Ecole normale supérieure de lyon - ENS LYON, November 2012. Thèse de doctorat dirigée par Patrick Baillot.

[LDJ14] Jiaxiang Liu, Nachum Dershowitz, and Jean-Pierre Jouannaud. Confluence by critical pair analysis. In *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 287–302, 2014.

[LJO15] Jiaxiang Liu, Jean-Pierre Jouannaud, and Mizuhito Ogawa. Confluence of layered rewrite systems. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 423–440. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[Luo89] Z. Luo. Ecc, an extended calculus of constructions. In *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, pages 386–395, 1989.

[Luo90] Zhaohui Luo. An extended calculus of constructions. Technical report, University of Edinburgh, 1990.

[Mü92] Fritz Müller. Confluence of the lambda calculus with left-linear algebraic rewriting. *Information Processing Letters*, 41(6):293 – 299, 1992.

[Mar08] Luc Maranget. Compiling pattern matching to good decision trees. In *Proceedings of the 2008 ACM SIGPLAN Workshop on ML*, ML '08, page 35–46, New York, NY, USA, 2008. Association for Computing Machinery.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1:497–536, 1991.

[ML75] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73 – 118. Elsevier, 1975.

[ML84] Per Martin-Löf. *Intuitionistic type theory.* Studies in Proof Theory. Napoli: Bibliopolis, 1984.

[MN98] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3 – 29, 1998.

[New42] M. H. A. Newman. On theories with a combinatorial definition of "equivalence.". *Journal of Symbolic Logic*, 7(3):123–123, 1942.

[Oka89] Mitsuhiro Okada. Strong normalizability for the combined system of the typed lambda calculus and an arbitrary convergent term rewrite system. In Gaston H. Gonnet, editor, *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, ISSAC '89, Portland, Oregon, USA, July 17-19, 1989*, pages 357–363. ACM, 1989.

[PM93] Christine Paulin-Mohring. Inductive definitions in the system coq rules and properties. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[Ros73] Barry K. Rosen. Tree-manipulating systems and church-rosser theorems. *J. ACM*, 20(1):160–187, January 1973.

[Rus08] Bertrand Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30(3):222–262, 1908.

[SAB+20] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq Project. *Journal of Automated Reasoning*, February 2020.

[Sai15a] Ronan Saillard. Rewriting modulo beta in the lambda-pi-calculus modulo. *Electronic Proceedings in Theoretical Computer Science*, 185:87–101, Jul 2015.

[Sai15b] Ronan Saillard. *Typechecking in the lambda-Pi-Calculus Modulo : Theory and Practice. (Vérification de typage pour le lambda-Pi-Calcul Modulo : théorie et pratique).* PhD thesis, Mines ParisTech, France, 2015.

[SBF+20] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq. *Proceedings of the ACM on Programming Languages*, pages 1–28, January 2020.

[SMB20] Danijela Simić, Filip Marić, and Pierre Boutry. Formalization of the poincaré disc model of hyperbolic geometry. *Journal of Automated Reasoning*, 2020.

[ST14] Matthieu Sozeau and Nicolas Tabareau. Universe Polymorphism in Coq. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving*, pages 499–514, Vienna, Austria, July 2014.

[Tea] The Coq Development Team. *The Coq Proof Assistant Reference Manual*. INRIA. Available at coq.inria.fr/refman/index.html.

[Thi20] François Thiré. *Meta-theory of Cumulative Types Systems and their embeddings to the lambda-Pi-calculus modulo theory*. PhD thesis, LSV, ENS Paris-Saclay, Université Paris-Saclay, France, 2020.

[Toy81] Yoshihito Toyama. On the church-rosser property of term rewriting systems. In *NTT ECL Technical Report*, page 12, 1981. in Japanese.

[vO94] Vincent van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit Amsterdam, 1994.

[vO97] Vincent van Oostrom. Developing developments. *Theor. Comput. Sci.*, 175(1):159–181, 1997.

[vO08] Vincent van Oostrom. Confluence by decreasing diagrams converted. In Voronkov A., editor, *RTA*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2008.

[vOvR94] Vincent van Oostrom and Femke van Raamsdonk. Weak orthogonality implies confluence: The higher-order case. In Anil Nerode and Yu. V. Matiyasevich, editors, *Logical Foundations of Computer Science*, pages 379–392, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[WR27] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1925–1927.

[Wu19] Jui-Hsuan Wu. Checking the type safety of rewrite rules in the lambda-Pi-Calculus modulo rewriting. Master's thesis, Ecole Normale Supérieure, September 2019.

Figures

$$\frac{\Sigma \vdash_{\mathcal{D}} \Gamma \ \mathbf{WF}_{\mathcal{D}} \qquad (x : A) \in \Sigma}{\Sigma; \Gamma \vdash_{\mathcal{D}} x : A}\ \mathcal{P}_{var}$$

$$\frac{\Sigma; \Gamma \vdash_{\mathcal{D}} M : A \qquad \Sigma; \Gamma \vdash_{\mathcal{D}} B : s \qquad A \equiv_{\beta\mathcal{R}} B}{\Sigma; \Gamma \vdash_{\mathcal{D}} M : B}\ \mathcal{P}_{\equiv_{\beta\mathcal{R}}}$$

Figure 9.2: Extra typing rules for $\lambda\Pi_{\equiv}$

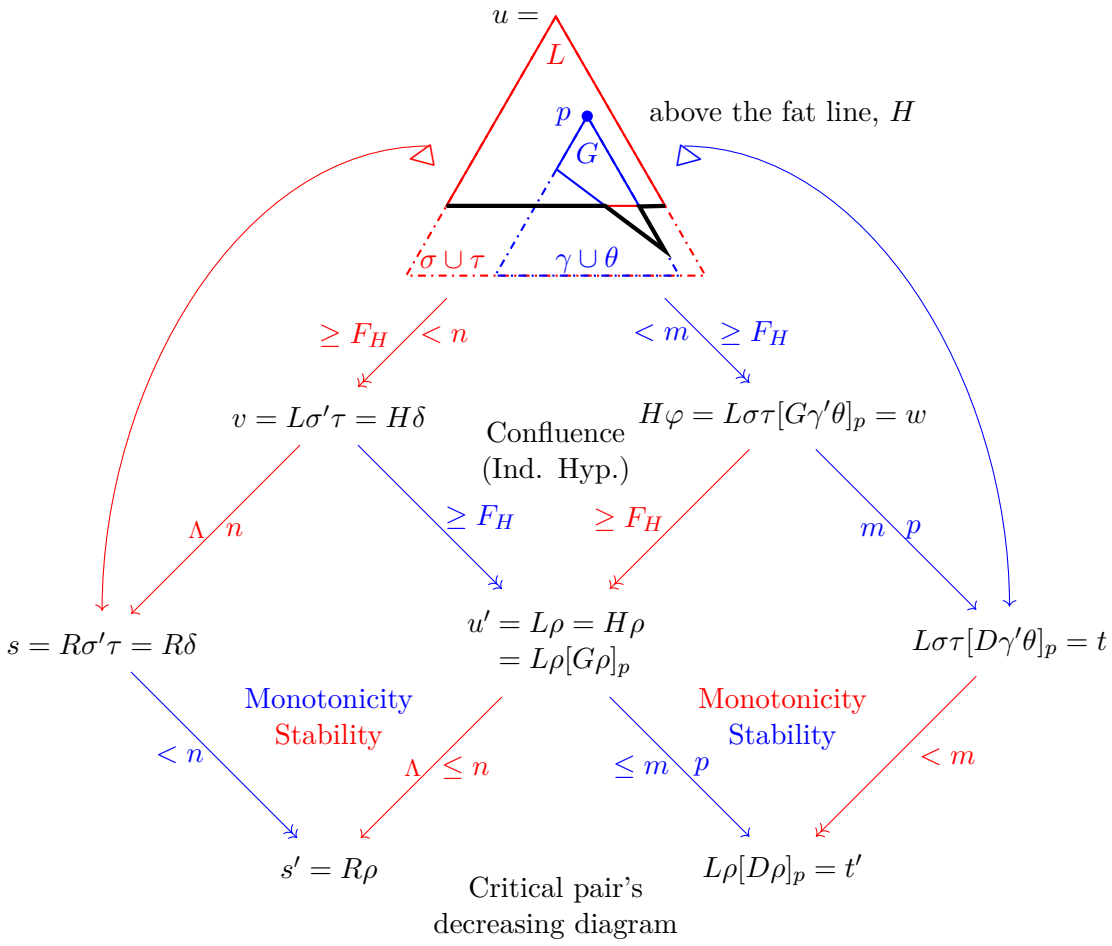


Figure 9.3: Critical pair lemma: Lemma 4.3.8

$$\frac{}{\varnothing\ \mathbf{WF}_{\mathcal{P}}}\ \mathcal{P}^{\mathbf{WF}}_{\varnothing} \qquad \frac{\Gamma \vdash_{\mathcal{P}} A : s_1 \qquad \Gamma, x : A \vdash_{\mathcal{P}} B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash_{\mathcal{P}} \Pi x : A.\, B : s_3}\ \mathcal{P}_{\Pi}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} A : s \qquad x \notin \Gamma}{\Gamma, x : A\ \mathbf{WF}_{\mathcal{P}}}\ \mathcal{P}^{\mathbf{WF}}_{\mathcal{X}} \qquad \frac{\Gamma, x : A \vdash_{\mathcal{P}} M : B \qquad \Gamma \vdash_{\mathcal{P}} \Pi x : A.\, B : s}{\Gamma \vdash_{\mathcal{P}} \lambda x : A.\, M : \Pi x : A.\, B}\ \mathcal{P}_{\lambda}$$

$$\frac{\Gamma\ \mathbf{WF}_{\mathcal{P}} \qquad (x : A) \in \Gamma}{\Gamma \vdash_{\mathcal{P}} x : A}\ \mathcal{P}_{\mathcal{X}} \qquad \frac{\Gamma \vdash_{\mathcal{P}} M : \Pi x : A.\, B \qquad \Gamma \vdash_{\mathcal{P}} N : A}{\Gamma \vdash_{\mathcal{P}} M\ N : B\{x \mapsto N\}}\ \mathcal{P}_{@}$$

$$\frac{\Gamma\ \mathbf{WF}_{\mathcal{P}} \qquad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash_{\mathcal{P}} s_1 : s_2}\ \mathcal{P}_{\mathcal{S}} \qquad \frac{\Gamma \vdash_{\mathcal{P}} M : A \qquad \Gamma \vdash_{\mathcal{P}} B : s \qquad A \equiv_{\beta} B}{\Gamma \vdash_{\mathcal{P}} M : B}\ \mathcal{P}_{\equiv}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} A : s_1 \qquad (s_1, s_2) \in \mathcal{C}}{\Gamma \vdash_{\mathcal{P}} A : s_2}\ \mathcal{P}_{\mathcal{C}}$$

Figure 9.4: Typing rules for $\mathrm{PTS}^{\preceq}(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$

$$\frac{}{\varnothing\ \mathbf{WF}_{\mathcal{C}}}\ \mathcal{P}^{\mathbf{WF}}_{\varnothing} \qquad \frac{\Gamma \vdash_{\mathcal{C}} A : s_1 \qquad \Gamma, x : A \vdash_{\mathcal{C}} B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash_{\mathcal{C}} \Pi x : A.\, B : s_3}\ \mathcal{P}_{\Pi}$$

$$\frac{\Gamma \vdash_{\mathcal{C}} A : s \qquad x \notin \Gamma}{\Gamma, x : A\ \mathbf{WF}_{\mathcal{C}}}\ \mathcal{P}^{\mathbf{WF}}_{\mathcal{X}} \qquad \frac{\Gamma, x : A \vdash_{\mathcal{C}} t : B \qquad \Gamma \vdash_{\mathcal{C}} \Pi x : A.\, B : s}{\Gamma \vdash_{\mathcal{C}} \lambda x : A.\, t : \Pi x : A.\, B}\ \mathcal{P}_{\lambda}$$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash_{\mathcal{C}} x : A}\ \mathcal{P}_{\mathcal{X}} \qquad \frac{(s_1, s_2) \in \mathcal{A}}{\Gamma \vdash_{\mathcal{C}} s_1 : s_2}\ \mathcal{P}_{\mathcal{S}} \qquad \frac{\Gamma \vdash_{\mathcal{C}} u : \Pi x : A.\, B \qquad \Gamma \vdash_{\mathcal{C}} t : A}{\Gamma \vdash_{\mathcal{C}} u\ t : B\{x \mapsto t\}}\ \mathcal{P}_{@}$$

$$\frac{\Gamma \vdash_{\mathcal{C}} t : s \qquad s \preceq_{\mathcal{C}} s'}{\Gamma \vdash_{\mathcal{C}} t : s'}\ \mathcal{P}_{\preceq_{\mathcal{C}}} \qquad \frac{\Gamma \vdash_{\mathcal{C}} B : s \qquad \Gamma \vdash_{\mathcal{C}} t : A \qquad A \preceq_{\mathcal{C}} B}{\Gamma \vdash_{\mathcal{C}} t : B}\ \mathcal{P}_{\preceq}$$

$$\frac{}{A \preceq_{\mathcal{C}} A}\ \preceq_{\mathcal{C}} i \qquad \frac{(s, s') \in \mathcal{C}^*}{s \preceq_{\mathcal{C}} s'}\ \preceq_{\mathcal{C}} c \qquad \frac{B \preceq_{\mathcal{C}} B'}{\Pi x : A.\, B \preceq_{\mathcal{C}} \Pi x : A.\, B'}\ \preceq_{\mathcal{C}} \pi$$

$$\frac{A \equiv_{\beta} A' \qquad A' \preceq_{\mathcal{C}} B' \qquad B' \equiv_{\beta} B}{A \preceq_{\mathcal{C}} B}\ \preceq_{\mathcal{C}} \equiv$$

Figure 9.5: Typing rules for $\mathrm{CTS}(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{C})$

$$
\begin{array}{rcl}
\mathbb{N} & : & * \\
\mathtt{0} & : & \mathbb{N} \\
\mathtt{S} & : & \mathbb{N} \to \mathbb{N} \\
\mathtt{max} & : & \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
 & & \\
\boldsymbol{\mathcal{S}} & : & * \\
\mathtt{Prop} & : & \boldsymbol{\mathcal{S}} \\
\mathtt{Type}_{\square} & : & \mathbb{N} \to \boldsymbol{\mathcal{S}} \\
\boldsymbol{\mathcal{A}}(\square) & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \\
\boldsymbol{\mathcal{R}}(\square,\square) & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \\
\boldsymbol{\mathcal{C}}(\square,\square) & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to * \\
 & & \\
\top & : & * \\
\mathbf{I} & : & \top
\end{array}
\qquad
\begin{array}{rcl}
\mathtt{max}\ (\mathtt{S}\ M)\ (\mathtt{S}\ N) & \longrightarrow & \mathtt{S}\ (\mathtt{max}\ M\ N) \\
\mathtt{max}\ M\ \mathtt{0} & \longrightarrow & M \\
\mathtt{max}\ \mathtt{0}\ N & \longrightarrow & N \\
\boldsymbol{\mathcal{A}}(\mathtt{Prop}) & \longrightarrow & \mathtt{Type}_0 \\
\boldsymbol{\mathcal{A}}(\mathtt{Type}_N) & \longrightarrow & \mathtt{Type}_{(\mathtt{S}\,N)} \\
\boldsymbol{\mathcal{R}}(S,\mathtt{Prop}) & \longrightarrow & \mathtt{Prop} \\
\boldsymbol{\mathcal{R}}(\mathtt{Prop},\mathtt{Type}_N) & \longrightarrow & \mathtt{Type}_N \\
\boldsymbol{\mathcal{R}}(\mathtt{Type}_M,\mathtt{Type}_N) & \longrightarrow & \mathtt{Type}_{(\mathtt{max}\,M\,N)} \\
\boldsymbol{\mathcal{C}}(\mathtt{Prop},N) & \longrightarrow & \top \\
\boldsymbol{\mathcal{C}}(\mathtt{Type}_0,\mathtt{Type}_N) & \longrightarrow & \top \\
\boldsymbol{\mathcal{C}}(\mathtt{Type}_{(\mathtt{S}\,M)},\mathtt{Type}_{(\mathtt{S}\,N)}) & \longrightarrow & \boldsymbol{\mathcal{C}}(\mathtt{Type}_M,\mathtt{Type}_N)
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{U}_{\square} & : & \boldsymbol{\mathcal{S}} \to * \\
\mathtt{T}_{\square}\ \square & : & \Pi s : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_s \to * \\
\mathbf{u}_{\square} & : & \Pi s : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_{\boldsymbol{\mathcal{A}}(s)} \\
\boldsymbol{\pi}^{\square}_{\square}\ \square\ \square & : & \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \Pi a : \mathtt{U}_{s_1}.\, (\mathtt{T}_{s_1}\ a \to \mathtt{U}_{s_2}) \to \mathtt{U}_{\boldsymbol{\mathcal{R}}(s_1,s_2)} \\
\uparrow^{\square}_{\square}\ \square\ \square & : & \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \boldsymbol{\mathcal{C}}(s_1,s_2) \to \mathtt{U}_{s_1} \to \mathtt{U}_{s_1} \\
\uparrow^{\square}_{\square}\ \square & : & \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_{s_1} \to \mathtt{U}_{s_1}
\end{array}
$$

$$
\begin{array}{rcl}
\uparrow^{S'}_{S}\ \square\ A & \longrightarrow & \uparrow^{S'}_{S}\ A \\
\mathtt{T}_{\square}\ \mathbf{u}_S & \longrightarrow & \mathtt{U}_S \\
\mathtt{T}_{\square}\ \left(\boldsymbol{\pi}^{S'}_{S}\ A\ \lambda x.\, B[x]\right) & \longrightarrow & \Pi x : \mathtt{T}_S\ A.\, \mathtt{T}_{S'}\ B[x] \\
\mathtt{T}_{S'}\ \left(\uparrow^{S'}_{S}\ A\right) & \longrightarrow & \mathtt{T}_S\ A
\end{array}
$$

$$
\begin{array}{rcl}
\uparrow^{N}_{N}\ A & \longrightarrow & A \\
\uparrow^{M}_{K}\ \left(\uparrow^{K}_{N}\ A\right) & \longrightarrow & \uparrow^{M}_{N}\ A \\
\boldsymbol{\pi}^{N}_{K}\ \left(\uparrow^{K}_{M}\ A\right)\ \lambda x.\, B[x] & \longrightarrow & \uparrow^{\boldsymbol{\mathcal{R}}(K,N)}_{\boldsymbol{\mathcal{R}}(M,N)}\ \left(\boldsymbol{\pi}^{N}_{M}\ A\ \lambda x : \mathtt{T}_M\ A.\, B[x]\right) \\
\boldsymbol{\pi}^{K}_{M}\ A\ \lambda x.\, \left(\uparrow^{K}_{N}\ B[x]\right) & \longrightarrow & \uparrow^{\boldsymbol{\mathcal{R}}(M,K)}_{\boldsymbol{\mathcal{R}}(M,N)}\ \left(\boldsymbol{\pi}^{N}_{M}\ A\ \lambda x : \mathtt{T}_M\ A.\, B[x]\right)
\end{array}
$$

Figure 9.6: Finite encoding signature for the infinitely sorted CC_ω: $\mathcal{D}[\mathrm{CC}_\omega]$

$$
\begin{array}{rcl}
\mathbb{N} & : & * \\
\mathtt{0} & : & \mathbb{N} \\
\mathtt{S} & : & \mathbb{N} \to \mathbb{N} \\
\mathtt{max} & : & \mathbb{N} \to \mathbb{N} \to \mathbb{N} \\
\mathtt{max}\ (\mathtt{S}\ M)\ (\mathtt{S}\ N) & \longrightarrow & \mathtt{S}\ (\mathtt{max}\ M\ N) \\
\mathtt{max}\ M\ \mathtt{0} & \longrightarrow & M \\
\mathtt{max}\ \mathtt{0}\ N & \longrightarrow & N
\end{array}
$$

$$
\begin{array}{rcl}
\boldsymbol{\mathcal{S}} & : & * \\
\mathtt{Prop} & : & \boldsymbol{\mathcal{S}} \\
\mathtt{Type}_{\square} & : & \mathbb{N} \to \boldsymbol{\mathcal{S}} \\
\boldsymbol{\mathcal{A}}(\square) & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \\
\boldsymbol{\mathcal{R}}(\square, \square) & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \\
\boldsymbol{\mathcal{C}}(\square, \square) & : & \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{S}}
\end{array}
$$

$$
\begin{array}{rcl}
\boldsymbol{\mathcal{A}}(\mathtt{Prop}) & \longrightarrow & \mathtt{Type}_0 \\
\boldsymbol{\mathcal{A}}(\mathtt{Type}_N) & \longrightarrow & \mathtt{Type}_{(\mathtt{S}\ N)} \\
\boldsymbol{\mathcal{R}}(S, \mathtt{Prop}) & \longrightarrow & \mathtt{Prop} \\
\boldsymbol{\mathcal{R}}(\mathtt{Prop}, \mathtt{Type}_N) & \longrightarrow & \mathtt{Type}_N \\
\boldsymbol{\mathcal{R}}(\mathtt{Type}_M, \mathtt{Type}_N) & \longrightarrow & \mathtt{Type}_{(\mathtt{max}\ M\ N)} \\
\boldsymbol{\mathcal{C}}(S, \mathtt{Prop}) & \longrightarrow & \mathtt{Prop} \\
\boldsymbol{\mathcal{C}}(\mathtt{Prop}, \mathtt{Type}_M) & \longrightarrow & \mathtt{Type}_M \\
\boldsymbol{\mathcal{C}}(\mathtt{Type}_M, \mathtt{Type}_N) & \longrightarrow & \mathtt{Type}_{(\mathtt{max}\ M\ N)}
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{U}_{\square} & : & \boldsymbol{\mathcal{S}} \to * \\
\mathtt{T}_{\square}\ \square & : & \Pi s : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_s \to * \\
\mathbf{u}_{\square} & : & \Pi s : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_{\boldsymbol{\mathcal{A}}(s)} \\
\boldsymbol{\pi}^{\square}_{\square}\ \square\ \square & : & \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \Pi a : \mathtt{U}_{s_1}.\, (\mathtt{T}_{s_1}\ a \to \mathtt{U}_{s_2}) \to \mathtt{U}_{\boldsymbol{\mathcal{R}}(s_1, s_2)} \\
\uparrow^{\square}_{\square}\ \square & : & \Pi s_1\ s_2 : \boldsymbol{\mathcal{S}}.\, \mathtt{U}_{s_1} \to \mathtt{U}_{\boldsymbol{\mathcal{C}}(s_1, s_2)}
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{T}_{\square}\ \mathbf{u}_S & \longrightarrow & \mathtt{U}_S \\
\mathtt{T}_{\square}\ \left(\boldsymbol{\pi}^{S'}_{S}\ A\ (\lambda x.\, B[x])\right) & \longrightarrow & \Pi x : \mathtt{T}_S\ A.\, \mathtt{T}_{S'}\ B[x] \\
\mathtt{T}_{\square}\ (\uparrow^{\square}_{S} A) & \longrightarrow & \mathtt{T}_S\ A
\end{array}
$$

Figure 9.7: [Assaf] A finite encoding signature for the infinitely sorted CC_{ω}

$$\mathcal{P}_{\mathcal{S}} \quad \left[\frac{\dfrac{\pi_{\mathcal{A}}}{\phi \vdash_{\mathcal{S}} \mathcal{A}(s_1, s_2)}}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} s_1 : s_2}\right] := \underline{\mathbf{u}}^{[\pi_{\mathcal{A}}]}_{[s_1],[s_2]}$$

$$\mathcal{P}_{\mathcal{X}} \quad \left[\frac{(x : A) \in \Gamma}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} x : A}\right] := x$$

$$\mathcal{P}_{\lambda} \quad \left[\frac{\dfrac{\pi_A}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s} \qquad \dfrac{\pi_t}{\Sigma; \overline{i}/\phi; \Gamma, x : A \vdash_{\mathcal{S}} t : B}}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} \lambda x : A.\, t : \Pi x : A.\, B}\right] := \lambda x \ : \ \mathtt{T}_{[s]} \ [\pi_A] \ . \ [\pi_t]$$

$$\mathcal{P}_{@} \quad \left[\frac{\dfrac{\pi_M}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} M : \Pi x : A.\, B} \qquad \dfrac{\pi_N}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} N : A}}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} M\ N : B\{x \mapsto N\}}\right] := [\pi_M] \ \ [\pi_N]$$

$$\mathcal{P}_{\Pi} \quad \left[\frac{\dfrac{\pi_A}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s_1} \qquad \dfrac{\pi_B}{\Sigma; \overline{i}/\phi; \Gamma, x : A \vdash_{\mathcal{S}} B : s_2} \qquad \dfrac{\pi_{\mathcal{R}}}{\phi \vdash_{\mathcal{S}} \mathcal{R}(s_1, s_2, s_3)}}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} \Pi x : A.\, B : s_3}\right]$$
$$:= \underline{\boldsymbol{\pi}}^{[\pi_{\mathcal{R}}]}_{[s_1],[s_2],[s_3]} \ [\pi_A] \ \lambda x : \mathtt{T}_{[s_1]} \ [\pi_A] . \ [\pi_B]$$

$$\mathcal{P}_{\preceq} \quad \left[\frac{\dfrac{\pi_M}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} M : A} \qquad \dfrac{\pi_A}{\ldots \vdash_{\mathcal{S}} A : s} \qquad \dfrac{\pi_B}{\ldots \vdash_{\mathcal{S}} B : s'} \qquad \dfrac{\pi}{\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B}}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} M : B}\right]$$
$$:= \ {}^{[s']}_{[s]}\!\uparrow^{[\pi_B]}_{[\pi_A]} \ p \ \begin{bmatrix} \pi_M \\ M : A \end{bmatrix} \text{ with } p := \begin{cases} \mathtt{strefl} \ [s] \ [\pi_A] & \text{if } A \equiv_{\beta\Sigma\phi} B \\ [\pi] & \text{otherwise} \end{cases}$$

$$\mathcal{P}_{decl} \quad \left[\frac{(\mathtt{c}[\overline{j}/\psi] : \tau) \in \Sigma \qquad |\overline{u}| = |\overline{j}| \qquad \forall i, \ \dfrac{\pi_i}{\phi \vDash \psi_i\{\overline{j} \mapsto \overline{u}\}}}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} \mathtt{c}_{\overline{u}} : \tau\{\overline{u}/\overline{j}\}}\right] := \mathtt{c} \ [\overline{u}] \ [\overline{\pi}]$$

$$\mathcal{P}_{def} \quad \left[\frac{(\mathtt{c}[\overline{j}/\psi] := t : \tau) \in \Sigma \qquad |\overline{u}| = |\overline{j}| \qquad \forall i, \ \dfrac{\pi_i}{\phi \vDash \psi_i\{\overline{j} \mapsto \overline{u}\}}}{\Sigma; \overline{i}/\phi; \Gamma \vdash_{\mathcal{S}} \mathtt{c}_{\overline{u}} : \tau\{\overline{u}/\overline{j}\}}\right] := \mathtt{c} \ [\overline{u}] \ [\overline{\pi}]$$

Figure 9.8: Typing judgment translation

$$\mathcal{S}^{\mathbf{WF}}_{\varnothing} \quad \left[\frac{}{\varnothing\ \mathbf{WF}_{\mathcal{S}}}\right] := \varnothing$$

$$\mathcal{S}^{\mathbf{WF}}_{decl} \quad \left[\frac{\frac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_{\mathcal{S}}} \qquad \frac{\pi_\tau}{\Sigma; \bar{i}/\phi; \varnothing \vdash_{\mathcal{S}} \tau : s} \qquad \ldots}{\Sigma, (\mathsf{c}[\bar{i}/\phi] : \tau)\ \mathbf{WF}_{\mathcal{S}}}\right]$$
$$:= \left[\frac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_{\mathcal{S}}}\right] \quad , \quad \mathsf{c} : \Pi \bar{i} : \boldsymbol{\mathcal{S}}. \Pi c_1 : \boldsymbol{\epsilon}\,[\phi_1]. \ldots \Pi c_k : \boldsymbol{\epsilon}\,[\phi_k]. \mathtt{T}_{[s]}\ \left[\frac{\pi_\tau}{\tau : s}\right]$$
$$, \quad \mathsf{c}' : \Pi \bar{i} : \boldsymbol{\mathcal{S}}. \mathbb{C}$$
$$, \quad \mathsf{c}\ I_1\ \ldots\ I_n\ C_1\ \ldots\ C_k \longrightarrow \mathsf{u}(\mathsf{c}(\mathbf{u}_{[s]\{\bar{i} \mapsto \bar{I}\}}, \left[\frac{\pi_\tau}{\tau : s}\right] \left\{\bar{i} \mapsto \bar{I}, \bar{c} \mapsto \bar{C}\right\}), \mathsf{c}'\ I_1\ \ldots\ I_n)$$

$$\mathcal{S}^{\mathbf{WF}}_{def} \quad \left[\frac{\frac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_{\mathcal{S}}} \qquad \frac{\pi_\tau}{\Sigma; \bar{i}/\phi; \varnothing \vdash_{\mathcal{S}} \tau : s} \qquad \frac{\pi_t}{\Sigma; \bar{i}/\phi; \varnothing \vdash_{\mathcal{S}} t : \tau} \qquad \ldots}{\Sigma, (\mathsf{c}[\bar{i}/\phi] := t : \tau)\ \mathbf{WF}_{\mathcal{S}}}\right]$$
$$:= \left[\frac{\pi_\Sigma}{\Sigma\ \mathbf{WF}_{\mathcal{S}}}\right] \quad , \quad \mathsf{c} : \Pi \bar{i} : \boldsymbol{\mathcal{S}}. \Pi c_1 : \boldsymbol{\epsilon}\,[\phi_1]. \ldots \Pi c_k : \boldsymbol{\epsilon}\,[\phi_k]. \mathtt{T}_{[s]}\ \left[\frac{\pi_\tau}{\tau : s}\right]$$
$$, \quad \mathsf{c}\ I_1\ \ldots\ I_n\ C_1\ \ldots\ C_k \longrightarrow \left[\frac{\pi_t}{t : \tau}\right] \left\{\bar{i} \mapsto \bar{I}, \bar{c} \mapsto \bar{C}\right\}$$

Figure 9.9: Signature well-formedness judgment translation

$$\mathcal{P}^{\mathbf{WF}}_{\varnothing} \quad \left[\frac{\phi\ \text{atomic} \qquad \vDash \phi}{\Sigma \vdash_{\mathcal{S}} \bar{i}/\phi; \varnothing\ \mathbf{WF}_{\mathcal{S}}}\right] := \bar{i} : \boldsymbol{\mathcal{S}}, c_1 : \boldsymbol{\epsilon}\,[\phi_1], \ldots, c_k : \boldsymbol{\epsilon}\,[\phi_k]$$

$$\mathcal{P}^{\mathbf{WF}}_{\mathcal{X}} \quad \left[\frac{\frac{\pi_\Gamma}{\Sigma \vdash_{\mathcal{S}} \bar{i}/\phi; \Gamma\ \mathbf{WF}_{\mathcal{S}}} \qquad \frac{\pi_A}{\Sigma; \bar{i}/\phi; \Gamma \vdash_{\mathcal{S}} A : s}}{\Sigma; \bar{i}/\phi; \Gamma, x : A\ \mathbf{WF}_{\mathcal{S}}}\right] := [\pi_\Gamma], x : \mathtt{T}_{[s]}\ [\pi_A]$$

Figure 9.10: Context well-formedness judgment translation

$$\preceq_{\mathrm{r}} \quad \left[\frac{}{\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} A}\right] \qquad := \mathbf{I}$$

$$\preceq_{\mathcal{C}} \quad \left[\frac{\dfrac{\pi}{\phi \vdash_{\mathcal{S}} \mathcal{C}(s, s')}}{\phi \vdash_{\mathcal{S}} s \preceq_{\Sigma\phi} s'}\right] \qquad := \left[\frac{\pi}{\phi \vdash_{\mathcal{S}} \mathcal{C}(s, s')}\right]$$

$$\preceq_{\pi} \quad \left[\frac{\dfrac{\pi}{\phi \vdash_{\mathcal{S}} B \preceq_{\Sigma\phi} B'}}{\phi \vdash_{\mathcal{S}} \Pi x : A.\, B \preceq_{\Sigma\phi} \Pi x : A.\, B'}\right] \qquad := \left[\frac{\pi}{\phi \vdash_{\mathcal{S}} B \preceq_{\Sigma\phi} B'}\right]$$

$$\preceq_{\equiv} \quad \left[\frac{A \equiv_{\beta\Sigma\phi} A' \qquad \dfrac{\pi}{\phi \vdash_{\mathcal{S}} A' \preceq_{\Sigma\phi} B'} \qquad B' \equiv_{\beta\Sigma\phi} B}{\phi \vdash_{\mathcal{S}} A \preceq_{\Sigma\phi} B}\right] \qquad := \left[\frac{\pi}{\phi \vdash_{\mathcal{S}} A' \preceq_{\Sigma\phi} B'}\right]$$

Figure 9.11: Subtyping judgment translation

$$\dfrac{\dfrac{\overline{x : \triangle \vdash_{\mathcal{C}} x : \triangle}}{\vdash_{\mathcal{C}} \lambda x.\, x : \triangle \to \triangle} \qquad \dfrac{\overline{\vdash_{\mathcal{C}} * : \square} \qquad \overline{\square \preceq_{\mathcal{C}} \triangle}}{\vdash_{\mathcal{C}} * : \triangle}}{\vdash_{\mathcal{C}} (\lambda x.\, x)\ * : \triangle}$$

or

$$\dfrac{\dfrac{\dfrac{\overline{x : \square \vdash_{\mathcal{C}} x : \square}}{\vdash_{\mathcal{C}} \lambda x.\, x : \square \to \square} \qquad \dfrac{}{\vdash_{\mathcal{C}} * : \square}}{\vdash_{\mathcal{C}} (\lambda x.\, x)\ * : \square} \qquad \dfrac{}{\square \preceq_{\mathcal{C}} \triangle}}{\vdash_{\mathcal{C}} (\lambda x.\, x)\ * : \triangle}$$

or

$$\dfrac{\dfrac{\dfrac{\overline{x : \square \vdash_{\mathcal{C}} x : \square} \qquad \dfrac{}{\square \preceq_{\mathcal{C}} \triangle}}{x : \square \vdash_{\mathcal{C}} x : \triangle}}{\vdash_{\mathcal{C}} \lambda x.\, x : \square \to \triangle} \qquad \dfrac{}{\vdash_{\mathcal{C}} * : \square}}{\vdash_{\mathcal{C}} (\lambda x.\, x)\ * : \triangle}$$

or

$$\dfrac{\dfrac{\overline{x : \square \vdash_{\mathcal{C}} x : \square}}{\vdash_{\mathcal{C}} \lambda x.\, x : \square \to \square} \qquad \dfrac{\dfrac{}{\square \preceq_{\mathcal{C}} \triangle}}{\square \to \square \preceq_{\mathcal{C}} \square \to \triangle}}{\vdash_{\mathcal{C}} \lambda x.\, x : \square \to \triangle} \qquad \dfrac{}{\vdash_{\mathcal{C}} * : \square}$$

$$\vdash_{\mathcal{C}} (\lambda x.\, x)\ * : \triangle$$

Figure 9.12: Multiple derivations of $\vdash_{\mathcal{C}} (\lambda x.\, x)\ * : \triangle$

$$\begin{aligned}
\mathtt{c}(C, \mathtt{u}(C, T)) &\hookrightarrow T && \text{(c-u)}\\
\mathtt{u}(C, \mathtt{c}(C, T)) &\hookrightarrow T && \text{(u-c)}\\
\mathtt{c}(\boldsymbol{\pi}\ A\ \lambda x.\, B[x], \lambda x.\, F[x]) &\hookrightarrow \boldsymbol{cL}\ (\lambda x : \mathbb{C}.\, \mathtt{c}(B[x], F[\mathtt{u}(A, x)])) && \text{(c-}\lambda\text{)}\\
\mathtt{u}(\boldsymbol{\pi}\ A\ \lambda x.\, B[x], \boldsymbol{cL}\ \lambda x.\, F[x]) &\hookrightarrow \lambda x : \mathtt{D}\ A.\, \mathtt{u}(B[\mathtt{c}(A, x)], F[\mathtt{c}(A, x)]) && \text{(u-}\lambda\text{)}\\
\mathtt{u}(\boldsymbol{\pi}\ A\ \lambda x.\, B[x], T)\ U &\hookrightarrow \mathtt{u}(B[\mathtt{c}(A, U)], \boldsymbol{cA}\ T\ \mathtt{c}(A, U)) && \text{(u-a)}\\
\boldsymbol{cA}\ \mathtt{c}(\boldsymbol{\pi}\ A\ \lambda x.\, B[x], T)\ U &\hookrightarrow \mathtt{c}(B[U], T\ \mathtt{u}(A, U)) && \text{(c-a)}\\
\boldsymbol{cA}\ (\boldsymbol{cL}\ \lambda x.\, T[x])\ U &\hookrightarrow T[U] && \text{(c}\beta\text{)}
\end{aligned}$$

Figure 9.13: Rewrite rules

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Confluence d'ordre supérieur et encodage d'univers dans le Logical Framework

Mots clés: Lambda calcul, types dépendants, réécriture de terme, confluence, déduction modulo, polymorphisme d'univers

Résumé: La multiplicité des systèmes formels a mis en évidence la nécessité d'un socle logique commun dans lequel les formalismes logiques pourraient être exprimés. L'enjeu principal de ce manuscrit est la définition de techniques d'encodages reposant sur la réécriture de termes et capables de réprésenter les fonctionnalités avancées des systèmes de types modernes. Nos encodages s'appuieront sur le lambda-Pi calcul modulo, un système de types dépendants, communément utilisé comme cadre logique, étendu ici par de la réécriture d'ordre supérieur. On s'intéresse, dans une première partie, aux critères de confluence de systèmes de réécriture avec la bêta réduction. La confluence d'un système linéaire à gauche se déduit de l'étude de ses paires critiques pour lesquelles il faut exhiber un diagramme décroissant vis-à-vis d'un certain étiquetage des règles. Le cas non-linéaire nécessite, lui, une compartimentalisation des termes considérés. On considère, dans un second temps, l'encodage de systèmes de types complexes. Sont étudiés successivement, la cumulativité qui nécessite de considérer des symboles privés pour encoder une forme de "proof irrelevance", les expressions algébriques d'univers sous contraintes d'univers et enfin le polymorphisme d'univers dont on prouve la correction d'une fonction de traduction depuis un sous-ensemble de Coq. L'implantation de ces résultats a permis de traduire en Dedukti plusieurs développements Coq de taille significative.

Title: Higher-Order Confluence and Universe Embedding in the Logical Framework

Keywords: Lambda calculus, dependent types, term rewriting, confluence, deduction modulo, universe polymorphism

Abstract: In the context of the multiplicity of formal systems, it has become a growing need to express formal proofs into a common logical framework. This thesis focuses on the use of higher-order term rewriting to embed complex formal systems in the simple and well-studied lambda-Pi calculus modulo. This system, commonly used as a logical framework, features dependent types and is extended with higher-order term rewriting. We study, in a first part, criteria for the confluence properties of higher-order rewrite systems considered together with the usual beta reduction. In the case of left-linear systems, confluence can be reduced to the study of critical pairs which must be provided a decreasing diagram with relation to some rule labeling. We show that in the presence of non-linear rules, it is still possible to achieve confluence if the set of considered terms is layered. We then focus, in a second part, on the encoding of higher-order logics based on complex universe structures. The of cumulativity, a limited form of subtyping, is handled with new rewriting techniques relying on private symbols and allowing some form of proof irrelevance. We then describe how algebraic universe expressions containing level variables can be represented, even in presence of universe constraints. Eventually we introduce an embedding of universe polymorphism as defined in the core logic of the Coq system and prove the correctness of the defined translation mechanism. These results, along with other more practical techniques, allowed the implementation of a translator to Dedukti which was used to translate several sizable Coq developments.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France