Subtyping in Signatures

Georgiana Elena Lungu

A thesis submitted for the degree of

Doctor of Philosophy

Royal Holloway University of London

2017

Abstract

Type theories with canonical objects like Martin Löf 's Type Theory or Luo's UTT have increasingly gained popularity in the last decades due to their usage in proof assistants, formal semantics of natural language and formalization of mathematics. The main purpose of this work is to explore a new way of introducing coercive subtyping in such type theories which facilitates the representation of some practical notions of subtyping.

Introducing subtyping in dependent type theories is not straightforward when the preservation of properties like canonicity and subject reduction is also desired. Previous research already showed how such properties are affected by the usual notion of subsumptive subtyping and offered an alternative in the form of coercive subtyping introduced by enriching the system with a set of coercive subtyping judgements. Here I introduce a new way of adding coercive subtyping to type theory, specifically by annotating certain functions in assumptions, arguing that this is more handy to represent practical cases. This system is also closer to the programming model of proof assistants like Coq where coercions are annotated as such at the assumption level.

Assumptions in Type Theory are represented as either contexts, which are sequences of membership entries for variables that bear abstraction and substitution or signatures, which are sequences of memberships entries for constants for which abstraction and substitution are not available. I shall use signatures as an environment for subtyping assumptions. I will prove that the system thus obtained is well behaved, in that it is only abbreviational to the original system, by considering its relation with the previous version of coercive subtyping which was already proved to be well behaved.

To demonstrate the ability of the system to argue about practical situations, I will present three case studies. The first one studies the relationship between a subsumptive subtyping system and coercive subtyping. The second

case study discusses how Russell-style universe inclusions, as found in Homotopy Type Theory, can be understood as coercions in a system with Tarski style hierarchy. And the last discussion is the need to treat injectivity as an assumption as well in order to capture faithfully some notions of subtyping which are based on or generalize inclusion.

Acknowledgements

I am extremely grateful to my supervisor Zhaohui Luo for the invaluable guidance, support and patience throughout the course of this work. The things I've learnt while doing this research under his supervision changed my perspective over everything.

I would like to thank my colleagues Fjodor Part, Thomas van Binsbergen, Ionut Tutu and Claudia Chirita and to Sergei Soloviev for the insightful discussions about type theory, programming languages and logic.

I am also grateful to the Department of Computer Science at Royal Holloway University of London for the opportunity to pursue the research presented here.

Last but not least many thanks to my parents and my sister for the endless support.

Contents

Chapter 1

Introduction

Considering the increasing popularity of type theories with canonical objects due to usage in proof assistants like Coq [Coq10], Agda [Agd08], Plastic [CL01] and Lego [Pol94], in formalization of mathematics projects like Homotopy Type Theory [Uni13] and in formal semantics of natural language under the paradigm of common nouns as type initiated by Ranta [Ran94] and further developed by Luo [Luo12a], I find it important to formulate the concept of subtyping in a way which does not break the useful properties of such type theories and at the same time reflects the use of subtyping in practice. So the main focus of this thesis is to introduce the notion of subtyping in such a way and analyze some practical situations and how it can be used to argue about them. In the rest of this chapter I informally introduce some concepts and present the overview together with the contributions of this thesis.

1.1 Dependent Type Theory

Type Theory is a formal language developed around the concept of terms being of a certain, defined, uniquely determined type. In contrast to Set Theory, where $a \in A$ is a proposition that can be negated as $a \notin A$, in Type Theory the fact that a term has a type, denoted by $a{:}A$, is a derivable judgement and its negation does not make sense as the term a can only exist as a term of a certain type. In addition, it is its own deduction system, a type A and a term of it $a{:}A$ are introduced or computed via preestablished rules. Church's Simply Typed λ - Calculus [Chu32, Chu40] and Martin Löf's Intuitionistic Type Theory [ML73, ML84] are examples of such languages. The latter uses the propositions themselves as types and a proposition P being true amounts to the ability of constructing a proof for it $p{:}P$. This setting was initially employed for foundation of constructive mathematics but it recently also gained relevance as a programming language.

We can consider the type of natural numbers Nat and the type of pairs $Nat \times P$. The terms of this type have as their first component a natural number and as their second component a proof of the proposition P. Similarly we can consider the type of functions $A \longrightarrow B$. Further, given an integer $n{:}Nat$ we can intuitively consider $n * Nat$ which is the type of multiples of n. This can be seen as the type of natural numbers m, such that there exists another natural number q such that $m = n * q$. Formally, this can also be seen as a pair formed by m in the first component and in the second component a proof that there exists another natural number q such that $m = n * q$. Let us rephrase this by saying that the second component is a proof that n divides m. This time the type of the second component depends on the first component, let us call this type, which is a proposition, $P(n)$. The type of such pairs is called a dependent pairs type which is denoted by $\Sigma(Nat, \lambda m{:}Nat.P(m))$ or $\Sigma_{m:Nat}P(m)$. Similarly if we want to consider a program that takes a natural number n and gives back a vector of length n, we denote the type of this program by $\Pi(Nat, \lambda n{:}Nat.Vect(n))$ or $\Pi_{n:Nat}Vect(n)$ and refer to it as dependent function type.

What these examples have in common is the idea of types that depend on terms. A language with dependent types is more powerful than a simply typed language. Some dependent type systems in which types depend on terms are Martin-Löf's Intuitionistic Type Theory [ML84], Coquand and Mohring's Calculus of Inductive Constructions [CP90, PM93] and Luo's Extended Calculus of Constructions [Luo90] and Unifying Theory of Dependent Types(UTT) [Luo92, Luo94].

When working with dependent type theories one specifies the way well formed terms are built by introduction rules. Such a type theory can be specified in a meta-theory which we call a logical framework. A logical framework is a metalanguage for formalization of deductive systems like natural deduction, categorical logic, axiomatic methods or sequent calculus([Pfe02]). We will call the deductive systems under formalization *object theories*. The benefit of using a logical framework is that it is itself a type theory so it is computer understandable and it can embed multiple object theories in the same type theory.

An example of logical frameworks is Edinburgh Logical Framework from ([HHP93]) which is obtained by adding type dependency to the simply typed λ-calculus. It was used to represent natural deduction based on the correspondence *judgements-as-types*. Here by judgements we mean judgements of the object theory in the style of natural deduction. The types of the logical framework itself will be called kinds for distinction. Another logical framework is Martin-Löf's logical framework introduced for Martin Löf's intensional type theory [NPS90] and which is also based on λ-calculus. Martin-Löf's logical framework is untyped. A typed version of it was developed by Luo [Luo94] and UTT [Luo92, Luo94] is a theory specified in this logical framework.

Types in such type theories are introduced by type constructors whose introduction rules determine their canonical objects. Some of these systems exhibit some important properties like canonicity (every closed object of a type reduces to a canonical object of that type), subject reduction (if a term M reduces to another term N then, if $M{:}A$ is derivable then $N{:}A$ is derivable as well), strong normalization (starting from a well-typed term, every rewriting sequence terminates) and Church-Rosser (if a term M reduces to two terms P and Q then there exists a term N such that both P and Q reduce to N).

1.2 Signatures and Contexts

Judgements of a type theory are typically of the form $\Gamma \vdash J$, where Γ is called context and represents the assumptions part of the judgements. This is a sequence of membership entries like $x_1{:}A_1, ..., x_n{:}A_n$. $\{x_i\}_{i=\{1..n\}}$ are usually treated as variables, they can be substituted or abstracted over.

Signatures were first introduced in [HHP93], and are used to keep track of constants as opposed to variables. They are also sequences of membership entries like $a_1{:}A_1, ..., a_n{:}A_n$ but these entries don't support substitution or abstraction. When signatures are used, the judgements are of the form $\Gamma \vdash_\Sigma J$, where Γ is a context and Σ is a signature. Signatures have been used in [CL15] to represent situations in natural language under the paradigm of common nouns as types initiated by Ranta [Ran94] and further developed by Luo and colleagues [Luo12a, CL14, LL14].

To understand the difference better let us consider again the example with multiples of an integer. If $n{:}Nat$ is part of a context, say for the judgement $\Gamma, n{:}Nat, p{:}P(n) \vdash_\Sigma J$ we can substitute n with any concrete natural number, say 2 and obtain $\Gamma, p{:}P(2) \vdash [2/n]J$[1]. This cannot happen if we consider it to be part of a signature like $\vdash_{\Sigma, n:Nat, p:P(n)} J$ because n is not a variable here. Similarly, if we had the context $\Gamma, n{:}Nat \vdash_\Sigma p{:}P(n)$ we could abstract over it and obtain the judgement $\Gamma \vdash_\Sigma [n{:}Nat]p{:}\Pi(Nat, \lambda n{:}Nat.P(n))$. This again is not possible if $n{:}Nat$ is an entry in the signature. More details on how substitution and abstraction work follow in Subsection 2.1.1 of the next chapter.

1.3 Subtyping

Subtyping is a very important and widely used concept in Computer Science as well as in mathematics, Natural Language and other domains. The most intuitive and extensively used form of subtyping is subsumption, which states

[1]here we assume that n might occur in J and $[2/n]$ means that we substitute any occurrence of it with 2

that, if A is a subtype of B all terms of type A are also terms of type B. The intuition for this kind of subtyping is given by the very expressive notion of subset. In programming languages employing type assignment systems (well typing discipline), like ML, which is based on Curry's system, the problem of typeability is, for a given term M, to find a context (basis) Γ and a type A such that $\Gamma \vdash M{:}A$. Here, because of concepts like principal types (eg. [vB92]), it is crucial to understand the type hierarchy as a partial order and a term as belonging to multiple types.

If A is a subtype of A', we write $A \leq A'$. A very useful consequence of the subsumptive subtyping concept is that we can use an object of A wherever an object of A' is expected, a property like the Liskov substitution principle [LW94] from object oriented programming. This property is also referred to as subtyping polymorphism. Polymorphism can be achieved through subtyping but there is also a form of ad hock polymorphism, when there is no subsumptive relation, for example + can be applied to any pair of terms belonging to subtypes of real numbers, $\mathbb{R}$ but also to a pair of terms of type $String$([Rey80]), the type of strings or a pair of terms, for example, of type $\mathbb{Z}_8$, the type of integers modulo 8([JG94], none of these types having a subsumptive relation with $\mathbb{R}$. To represent this kind of polymorphism, Reynolds [Rey80] was first to consider the notion of coercion between objects of different unrelated (from subtyping point of view) types in programming languages. Other developments of coercion semantics of subtyping for programming languages include work by Mitchell [Mit84] and Breazu-Tannen et al [BCGS91].

When talking about dependent type theories, subtyping as understood in programming languages simply does not fit. On the one hand the problem of finding a type for an object doesn't make sense as objects can only be introduced as having a type. Further, allowing a term to have multiple types breaks some of the useful properties of such a type theory that I mentioned earlier, for example a term of a type can reduce to a term of a different type[2]. For this, Luo [Luo96, Luo99] introduced the notion of coercion in dependent

[2]more details about this follow in the next chapter in section 2.3

type theory to denote the use of explicitly distinguished conversions to map an object of T into objects of its supertypes. The kind of subtyping he proposed keeps the advantage of being able to use an object of a type A wherever an object of its supertypes is expected but without the burden of an object having multiple types. If A is a subtype of B, we call the application of a function that expects an object of type B to an object of type A coercive application. In order to keep the system consistent, when enhancing it with rules for coercive application, we want such applications to be nothing more than abbreviations of the normal well typed application. The formalization of this correctness was studied later in [SL02, LL01, Luo05] and only finalized in [LSX13, Xue13b]. This formalization is for a system which introduces subtyping through a set of subtyping judgements. More precisely, the authors consider a base type theory specified in Luo's logical framework [Luo94] and they enrich it with a set $\mathcal{C}$ formed of judgements of the form $\Gamma \vdash A <_c B$, where c is the coercion between A and B in context Γ, together with a rule that makes all judgements in $\mathcal{C}$ derivable.

Introducing subtyping through a set of judgements works well in theory, but it does not represent closely practical situations in which subtyping entries are part of assumption, for example the programming model of Coq [Coq10] which is a proof assistant that supports coercive subtyping. The way one can specify a subtyping assumption in Coq is simply by annotating a predefined mapping as coercion. This thesis introduces a system which allows the possibility to add coercive subtyping entries to assumptions which apply to a judgement rather than through a set which applies to the whole system. Previous work in this direction was started by Luo [LP13] but even though that seemed like a powerful system that setting is quite tedious to work with. I argue that the system I introduce here achieves a balance in that it is capable to represent practical situations, it is close to the programming model of proof assistants and at the same time it is reasonably easy to formalize and work with.

Subtyping in dependent type theories has also been studied by Betarte and

Tasistro [BT98] for Martin-Löf's logical framework analyzing subkinding between kinds (called types), Barthe and Frade[BF99] on constructor subtyping and Aspinall and Compagnoni [AC01] on a form of subsumptive subtyping in assumptions for Edinburgh Logical Framework [HHP93] among others. The latter inspires one of the practical situation that I will formally represent in the system I introduce here. I will also consider constructor subtyping when discussing certain properties that subtyping benefits from in practice.

1.4 Overview of the Thesis and Contributions

The second chapter introduces the setting used in later chapters of this thesis in formal details. It will give technical details about dependent type theories, logical frameworks used to specify type theories and their inductive types, signatures and subtyping.

A main objective of this thesis is to introduce a system which achieves a good balance between being powerful enough to represent some practical situations and to have a reasonably easy meta-theory. I argue that the system I introduce in chapter three achieves this objective. This system is an extension of an original dependent type theory with the ability to annotate conversions as coercions at assumption level. Essentially, this system is a system with coercive subtyping entries in signatures. This chapter also discusses the relation of this new system to a system introduced by Luo et al. [LSX13, Xue13b] which also uses the notion of coercive subtyping.

One of the practical situations that I claim this system is capable to represent is subsumptive subtyping. The first part of chapter four considers such a form, more precisely, a system which introduces subtyping through contexts, and shows how it can be represented in the system with coercive subtyping entries in signatures. Later in the chapter, more forms of subtyping are considered. First I consider Russell style universes with their cumulativity and argue that a system with Tarski style universes and coercive subtyping entries in signatures can represent it. Further, when subsumptive subtyping, often perceived as inclusion is considered, an important thing to consider is injectivity and to what extent coercive subtyping can exhibit such a property. This is a discussion I carry out at the end of chapter four by means of constructor subtyping with Leibniz equality.

The fifth chapter concludes the discussion from the previous chapters and presents certain points raised during the research work for this thesis and left open for future work. In particular, an important topic is extension by definition in type theories.

Chapter 2

Type Theory, Subtyping and Signatures

In this chapter I will give the formal context needed to understand the work done in this thesis. I will reiterate the sections of the previous chapter with emphasis on the technical details. In particular, I present the notion of logical framework with emphasise on Luo's Logical Framework (*LF* [Luo94]), and I give examples on how logical frameworks can be used to specify type theories. I also present universes, Russell and Tarski style and give examples of how they can be used. I revisit the notion of signatures in a more technical light and finish with a discussion about subsumptive and coercive subtyping and some of the multiple forms they appeared in related work.

2.1 Dependent Type Theory

2.1.1 Logical Framework

A logical framework is a metalanguage for formalization of object theories which are deductive system [Pfe02]. Such logical frameworks are Edinburgh Logical Framework([HHP93]), where dependently typed λ-calculus was used to represent natural deduction based on the correspondence *judgements-as-types*, the untyped Martin-Löf's logical framework used for Martin-Löf's intensional type theory [NPS90] based on the correspondence *propositions-as-types* and the logical framework which we shall denote by LF developed by Luo [Luo94] and used to specify UTT [Luo92, Luo94].

In what follows I will present LF in more detail. This is also presented in Chapter 9 of [Luo94].

Kinds. As mentioned earlier, the types of the logical framework itself will be called kinds for distinction. The kind of types will be called *Type*. Note that this kind is *Set* in Martin-Löf's logical framework [NPS90]. Other kinds of LF are of the form $El(A)$, where $A{:}Type$, and $(x{:}K)K'$. $El(A)$ is the kind of elements of A. I will often write A for $El(A)$ where there is no confusion. $(x{:}K)K'$ is the dependent product kind, analogous to $\Pi(A, B)$ which, if defined, is the dependent product type. If x does not occur free in K' I will simply write $(K)K'$ instead of $(x{:}K)K'$.

Judgements. Judgements of LF are of the form

1. $\vdash \Gamma$ which states that Γ is a valid context. Γ is of the form $x_1{:}K_1, ..., x_n{:}K_n$. Contexts are used to keep track of such variable which can be abstracted and substituted. Note that LF does not originally use signatures so all the assumption entries represent variables.

2. $\Gamma \vdash K\ \ kind$ which states that the kind K is valid under the assumptions in Γ.

3. $\Gamma \vdash k{:}K$ which states that the term k has kind K under the assumptions in Γ.

4. $\Gamma \vdash k = k'{:}K$ which states that the terms k and k' of kind K are *definitionally equal* under the assumptions in Γ. The notion of definitional equality follows in this section.

Inference Rules. A judgement can be derivable, that is it can be inferred using inference rules from derivable premises. Inference rules are of the form

$$\frac{J_1...J_n}{J}$$

with $\{J_i\}_{i\in\{1..n\}}$ the premises of the rules and J the conclusion of the rule. An instance of a rule is a rule in which $\{J_i\}_{i\in\{1..n\}}$ and J are concrete judgements.

In what follows, I will use the notation $\equiv$ to denote the syntactic equality.

Abstraction. In the signatures discussion from Section 1.2 of the previous chapter I also used the notation $[x{:}K]k$ to denote abstraction. What this really means is that, if we have the judgement $\Gamma, x{:}K \vdash k{:}K'$, we can abstract over the variable x to obtain the judgement $\Gamma \vdash [x{:}K]k{:}(x{:}K)K'$. Note that $[x{:}K]k$ corresponds to the untyped functional operation $(x)k$ from Martin-Löf's logical framework.

Substitution. When discussing about signatures, in Section 1.2 of the previous chapter, I briefly introduced the notation $[k/x]J$ to denote the substitution of the free variable x with k in the judgement J. We have seen that, in our case, J can be Γ, $K\ \mathit{kind}$, $k{:}K$ or $k = k'{:}K$. If $J \equiv k'{:}K$, then $[k/x]J \equiv [k/x]k'{:}[k/x]K$. Of course if x does not occur free in k', then $[k/x]k' \equiv k'$. If $J \equiv \Gamma \equiv x_1{:}K_1, ..., x_n{:}K_n$ then $[k/x]J \equiv x_1{:}[k/x]K_1, ...x_n{:}[k/x]K_n$. However when substituting a variable special care needs to be taken to not capture free variables, for example, let us look at $[y{:}L]k'{:}K$, where x does occur free in k'. If we substitute x with y, we obtain $[y{:}L]([y/x]k'){:}[y/x]K$ and now the bound $y{:}L$ also captures the replaced occurrences of x in k' which were free before the substitution, that is $[y/x]([y{:}L]k') \not\equiv [y{:}L]([y/x]k')$. This can be avoided by using α-conversion which is essentially the procedure of renaming bound variables. In our case, $[y{:}L]([z/x]k'){:}K$ is α-convertible to $[z{:}L]k'{:}K$ and now we can freely substitute x with y without capturing any additional variable. When a term t is α convertible to another term t', we write $t \equiv t'$.

Definitional Equality. I will use the notation $k_1 = k_2{:}K$ to denote that the terms k_1 and k_2 are definitionally equal. This means that they are identical up to $\beta\eta$ - *conversion.* By β **- conversion** we mean $([x{:}K]k')(k) \to_\beta [k/x]k'$ and by η **- conversion** we mean $[x{:}K]f(x) \to_\eta f$ when x does not occur free in f. For a judgement $\Gamma \vdash k = k'{:}K$ to be derivable we require that both

$\Gamma \vdash k{:}K$ and $\Gamma \vdash k'{:}K$ are derivable.

It is worth noting the difference between this kind of equality, also referred to as judgemental equality, and propositional equality. Propositional equality is a proposition and requires a proof, therefore a type which might be inhabited or not. For a type A, for any two elements $x{:}A$ and $y{:}A$, we define the type $Id_A(x, y)$ which is the proposition that x and y are equal. Two terms of type A, a_1 and a_2 can be proven to be equal or are propositionally equal if there exists $p{:}Id_A(a_1, a_2)$. The type $Id_A(a, a)$ is always inhabited.

The inference rules of LF are presented in Figure 2.1 (and Figure A.1 in the Appendix A). The rules in the first section specify how to derive valid contexts, or sequences of assumptions. For the dependent product kind section of this figure, the first rule on the left hand side gives a way to form the kind, the left rule on the second rule gives a way to introduce a term of this kind and the left rule on the third row gives a way to eliminate such a term. The last two rules from this section are for β and η conversions. The rest of the rules specify how to derive definitional equality and the kind of elements of a type.

2.1.2 Type Theories specified in LF

I mentioned Edinburgh Logical Framework ([HHP93]) has been used to represent natural deduction based on the correspondence *judgements-as-types*, the Martin-Löf's intensional type theory [NPS90] is specified in Martin-Löf's logical framework based on the correspondence *propositions-as-types*. How can one use LF as a meta language to specify object theories?

Essentially, to specify an object type theory in LF, one has to specify new constants and set of computation rules for each constant which represents how a term of the newly introduced type can be used. For example, for dependent

Validity of Signature/Contexts, Assumptions

$$\frac{}{\vdash \langle\rangle} \qquad \frac{\Gamma \vdash K\,\mathit{kind} \quad x \notin \mathit{dom}(\Gamma)}{\vdash \Gamma, x{:}K} \qquad \frac{\vdash \Gamma, x{:}K, \Gamma'}{\Gamma, x{:}K, \Gamma' \vdash x{:}K}$$

Equality Rules

$$\frac{\Gamma \vdash K\ \mathit{kind}}{\Gamma \vdash K = K} \qquad \frac{\Gamma \vdash K = K'}{\Gamma \vdash K' = K} \qquad \frac{\Gamma \vdash K = K' \quad \Gamma \vdash K' = K''}{\Gamma \vdash K = K''}$$

$$\frac{\Gamma \vdash k{:}K}{\Gamma \vdash k = k{:}K} \qquad \frac{\Gamma \vdash k = k'{:}K}{\Gamma \vdash k' = k{:}K} \qquad \frac{\Gamma \vdash k = k'{:}K \quad \Gamma \vdash k' = k''{:}K}{\Gamma \vdash k = k''{:}K}$$

$$\frac{\Gamma \vdash k{:}K \quad \Gamma \vdash K = K'}{\Gamma \vdash k{:}K'} \qquad \frac{\Gamma \vdash k = k'{:}K \quad \Gamma \vdash K = K'}{\Gamma \vdash k = k'{:}K'}$$

Substitution Rules

$$\frac{\vdash \Gamma_0, x{:}K, \Gamma_1 \quad \Gamma_0 \vdash k{:}K}{\vdash \Gamma_0, [k/x]\Gamma_1}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash K'\ \mathit{kind} \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K'\ \mathit{kind}} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash L = L' \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]L = [k/x]L'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash k'{:}K' \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash l = l'{:}K' \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]l = [k/x]l'{:}[k/x]K'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash K'\ \mathit{kind} \quad \Gamma_0 \vdash k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K' = [k'/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash l{:}K' \quad \Gamma_0 \vdash k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]l = [k'/x]l{:}[k/x]K'}$$

Dependent Product Kinds

$$\frac{\Gamma \vdash K\,\mathit{kind} \quad \Gamma, x{:}K \vdash K'\,\mathit{kind}}{\Gamma \vdash (x{:}K)K'\,\mathit{kind}} \qquad \frac{\Gamma \vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash K_1' = K_2'}{\Gamma \vdash (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Gamma, x{:}K \vdash y{:}K'}{\Gamma \vdash [x{:}K]y{:}(x{:}K)K'} \qquad \frac{\Gamma \vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash k_1 = k_2{:}K}{\Gamma \vdash [x{:}K_1]k_1 = [x{:}K_2]k_2{:}(x{:}K_1)K}$$

$$\frac{\Gamma \vdash f{:}(x{:}K)K' \quad \Gamma \vdash k{:}K}{\Gamma \vdash f(k){:}[k/x]K'} \qquad \frac{\Gamma \vdash f = f'{:}(x{:}K)K' \quad \Gamma \vdash k_1 = k_2{:}K}{\Gamma \vdash f(k_1) = f'(k_2){:}[k_1/x]K'}$$

$$\frac{\Gamma, x{:}K \vdash k'{:}K' \quad \Gamma \vdash k{:}K}{\Gamma \vdash ([x{:}K]k')(k) = [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma \vdash f{:}(x{:}K)K' \quad x \notin FV(f)}{\Gamma \vdash [x{:}K]f(x) = f{:}(x{:}K)K'}$$

The kind Type

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathit{Type}\ \mathit{kind}} \qquad \frac{\Gamma \vdash A{:}\mathit{Type}}{\Gamma \vdash \mathit{El}(A)\ \mathit{kind}} \qquad \frac{\Gamma \vdash A = B{:}\mathit{Type}}{\Gamma \vdash \mathit{El}(A) = \mathit{El}(B)}$$

Figure 2.1: Inference Rules for LF

product type the constants are

$$\begin{aligned} \Pi &: (A{:}\mathit{Type})(B{:}(A)\,\mathit{Type})\,\mathit{Type} \\ \lambda &: (A{:}\mathit{Type})(B{:}(A)\,\mathit{Type})((x{:}A)B(x))\Pi(A,B) \\ \mathit{app} &: (A{:}\mathit{Type})(B{:}(A)\,\mathit{Type})(\Pi(A,B))(x{:}A)B(x) \end{aligned}$$

and the computation rule is

$$\mathit{app}(A, B, \lambda(A, B, f), a) = f(a) : B(a).$$

It might be worth noting that dependent product kind is part of the logical framework. In contrast, for a type theory specified in LF to have a dependent product type, one needs to specify this type with constants as above.

To declare constant $k{:}K$ typically corresponds to the addition of a new inference rule

$$\frac{\vdash \Gamma}{\Gamma \vdash k{:}K}$$

to the type theory specified by LF. Similarly each computation rule corresponds to the addition of a new rule. For the example with dependent product type, the constants correspond to the addition of the rules in Figure 2.2 (and in Figure A.2 of the Appendix A).

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma, x{:}A \vdash B(x) : \mathit{Type}}{\Gamma \vdash \Pi(A,B) : \mathit{Type}}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma \vdash B : (A)\mathit{Type} \quad \Gamma \vdash f : (x{:}A)B(x)}{\Gamma \vdash \lambda(A,B,f) : \Pi(A,B)}$$

$$\frac{\Gamma \vdash g : \Pi(A,B) \quad \Gamma \vdash a : A}{\Gamma \vdash \mathit{app}(A,B,g,a) : B(a)}$$

$$\frac{\begin{array}{c}\Gamma \vdash A : \mathit{Type} \quad \Gamma \vdash B : (A)\mathit{Type} \\ \Gamma \vdash f : (x{:}A)B(x) \quad \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathit{app}(A,B,\lambda(A,B,f),a) = f(a) : B(a)}$$

Figure 2.2: Inference Rules for Π-type specified in LF

Observe that we do not need to add structural equality rules for Π, λ and *app* as they are constants and the above terms are obtained through the application from LF which already has structural equality rules.

Similarly we can introduce the type of natural numbers which we have already mentioned, *Nat*, with the following constants:

$$\begin{array}{rcl} \mathit{Nat} & : & \mathit{Type} \\ 0 & : & \mathit{Nat} \\ \mathit{Succ} & : & (\mathit{Nat})\mathit{Nat} \\ \mathit{rec} & : & (C{:}(\mathit{Nat})\mathit{Type})(c{:}C(0))(f{:}(x{:}\mathit{Nat})(C(x))C(\mathit{Succ}(x)))(n{:}N)(C(n)) \end{array}$$

and the computation rules:

$$rec(C, c, f, 0) = c : C(0)$$

$$rec(C, c, f, Succ(n)) = f(n, rec(C, c, f, n)) : C(Succ(n))$$

Note that the *rec* constant corresponds to the *induction principle* and in the second computation rule $rec(C, c, f, n)$ can be seen as a proof of the *induction hypothesis.* Just as the computation for Π tells us that, if we have a function, we can use it by applying it to something, the computation rule of *Nat* essentially tells that, if we have a natural number, we can use it by counting to it. For example, one can use *rec* to define the addition of natural numbers $+(m, n) = rec([x{:}Nat]Nat, m, [\_{:}Nat]Succ, n)$ Similarly to the Π type, *Nat* also corresponds to the respective rules from Figure 2.3.

$$\frac{\vdash \Gamma}{\Gamma \vdash Nat : Type} \qquad \frac{\vdash \Gamma}{\Gamma \vdash 0{:}Nat} \qquad \frac{\Gamma \vdash n{:}Nat}{\Gamma \vdash Succ(n){:}Nat}$$

$$\frac{\Gamma \vdash C{:}(Nat)Type \quad \Gamma \vdash n_0{:}C(0) \quad \Gamma \vdash f{:}(x{:}Nat)(C(x))C(Succ(x)) \quad \Gamma \vdash n{:}Nat}{\Gamma \vdash rec(C, n_0, f, n){:}C(n)}$$

$$\frac{\Gamma \vdash C{:}(Nat)Type \quad \Gamma \vdash n_0{:}C(0) \quad \Gamma \vdash f{:}(x{:}Nat)(C(x))C(Succ(x)) \quad \Gamma \vdash n{:}Nat}{\Gamma \vdash rec(C, n_0, f, 0) = n_0{:}C(0)}$$

$$\frac{\Gamma \vdash C{:}(Nat)Type \quad \Gamma \vdash n_0{:}C(0) \quad \Gamma \vdash f{:}(x{:}Nat)(C(x))C(Succ(x)) \quad \Gamma \vdash n{:}Nat}{\Gamma \vdash rec(C, n_0, f, Succ(n)) = f(n, rec(C, n_0, f, n)) : C(Succ(n))}$$

Figure 2.3: Inference Rules for *Nat* specified in *LF*

The Unifying Theory of Dependent Types *UTT*

UTT [Luo92, Luo94] is an important example of type theory specified in *LF*. Martin-Löf's intensional type theory [NPS90] is specified in Martin-Löf's logical framework based on the correspondence *propositions-as-types. UTT* distinguishes between data types and logical propositions and introduces a type of logical propositions *Prop* as follows:

$$\begin{array}{rcl} \mathit{Prop} & : & \mathit{Type} \\ \mathit{Prf} & : & (\mathit{Prop})\mathit{Type} \\ \forall & : & (A{:}\mathit{Type})((A)\mathit{Prop})\mathit{Prop} \\ \Lambda & : & (A{:}\mathit{Type})(P{:}(A)\mathit{Prop})((x{:}A)\mathit{Prf}(P(x)))\mathit{Prf}(\forall(A,P)) \\ E_\forall & : & (A{:}\mathit{Type})(P{:}(A)\mathit{Prop})(R{:}(\mathit{Prf}(\forall(A,P)))\mathit{Prop})((g{:}(x{:}A)\mathit{Prf}(P(x))) \\ & & \mathit{Prf}(R(\Lambda(A,P,g))))(z{:}\mathit{Prf}(\forall(A,P)))\mathit{Prf}(R(z)) \end{array}$$

and the computation rule is

$$E_\forall(A,P,R,f,\Lambda(A,P,g)) = f(g){:}\mathit{Prf}(R(\Lambda(A,P,g))).$$

Inductive types are generated by inductive schemata as in [Dyb91, PM93, Luo92, Luo94]. I mentioned earlier that a type is given by its canonical objects, also called values or β normal forms which are generated by constructors. Constructors of *Nat* for examples are 0 and *Succ* and any term of *Nat* can be obtained with its constructors. Similarly λ is the constructor of Π. UTT uses *schemata* to introduce inductive types. In what follows I will briefly define what this means, more details can be found in [Luo94]. First some definitions are required.

Definition 1. *K is called a Γ* ***- kind*** *if $\Gamma \vdash K$ kind is derivable. K is called a* ***small*** *Γ* ***- kind*** *if $K \equiv (x_1{:}El(A_1))...(x_n{:}El(A_n))El(A_{n+1})$ such that $\Gamma, x_1{:}El(A_1), ..., x_{i-1}{:}El(A_{i-1}) \vdash A_i{:}Type$ for any $i = 1,..,n$.*

For example $El(A)$ is a small Γ - kind for any A such that $\Gamma \vdash A{:}Type$ is derivable but *Type* or $(x{:}El(A))Type$ are not small kinds. In what follows I will write simply A for $El(A)$.

Definition 2 (Kind Schemata)**.** *Let Γ be a context and X a variable that does not occur free in Γ ($X \notin FV(\Gamma)$). We say Φ is a* ***strictly positive operator in*** *Γ and we write $Pos_\Gamma(\Phi)$ if Φ is of the form $(x_1{:}K_1)...(x_n{:}K_n)X$ where, for* *any* $0 \leq i \leq n$, K_i *is a small* $\Gamma, x_1{:}K_1, ...x_{i-1}{:}K_{i-1}$ *- kind. We say* Θ *is a* Γ ***- schema*** *and we write* $Sch_\Gamma(\Theta)$ *if*

1. $\Theta \equiv X$ *or*

2. $\Theta \equiv (x{:}K)\Theta_0$ *where* K *is a small* Γ *- kind and* $Sch_{\Gamma,x:K}(\Theta_0)$ *or*

3. $\Theta \equiv \Phi\Theta_0$ *where* $Pos_\Gamma(\Phi)$ *and* $Sch_\Gamma(\Theta_0)$

For example any strictly positive operator in Γ is a Γ - schema.

With this we can form inductive types and introduce their terms. Let $\overline{\Theta} \equiv < \Theta_1, ..., \Theta_m >$ be a finite sequence of Γ - schema. Θ generates a type constructor denoted by $\mathcal{M}[\overline{\Theta}]$ such that the judgement $\Gamma \vdash \mathcal{M}[\overline{\Theta}]{:}\mathit{Type}$ is derivable. The logical framework constant will be

$$\mathcal{M}[\overline{\Theta}]{:}\mathit{Type}$$

Further, for any $0 \leq i \leq n$, we have a constructor represented by the logical framework constant

$$\iota_i[\overline{\Theta}]{:}[\mathcal{M}[\overline{\Theta}]/X]\Theta_i$$

When I introduced *Nat* and Π-types, apart from the type formation and constructor constants, I mentioned there are some computation rules which use an additional constant. That constant, when generated through schemata is called the eliminator. To generate that constant and the computation rules we need the following definitions.

Definition 3. *Let* $Pos_\Gamma(\Phi)$ *we define*

- *For* $A{:}\mathit{Type}$, $C{:}(A)\mathit{Type}$ *and* $z{:}[A/X]\Phi$, $\Phi^o(A, C, z)$ *is*

  1. $C(z)$ *if* $\Phi \equiv X$ *or*

  2. $(x{:}K)\Phi_0^o(A, C, z(x))$ *if* $\Phi \equiv (x{:}K)\Phi_0$

- *For* $\Phi^\natural{:}(C{:}(A)\mathit{Type})(f{:}(x{:}A)C(x))(z{:}[A/X]\Phi)\Phi^o(A, C, z)$ *is defined as follows*

1. $\Phi^{\natural}(A)(C,f) = f$ *if* $\Phi \equiv X$ *or*

2. $\Phi^{\natural}(A)(C,f,z) = [x{:}K]\Phi_0^{\natural}(A)(C,f,z(x))$ *if* $\Phi \equiv (x{:}K)\Phi_0$

Definition 4. *Let* $Sch_\Gamma(\Theta)$*, with* $\Theta \equiv (x_1{:}M_1),...,(x_n{:}M_n)X$ *we define* ***the arity of*** Θ *to be the subsequence* $\langle M_{i_1},...,M_{i_k}\rangle$ *of* $\langle M_1,...,M_n\rangle$ *that consists of all strictly positive operators (obtained by induction on the structure of* Θ*). We denote this arity by* $Ari(\Theta)$*. For* $A{:}Type$*,* $C{:}(A)\,Type$ *and* $z{:}[A/X]\Theta$*, we define*

$$\begin{aligned}\Theta^o(A,C,z) \quad = \quad & (x_1{:}[A/X]M_1)...(x_n{:}[A/X]M_n)(M^o_{i_1}(A,C,x_{i_1}))...\\ & (M^o_{i_k}(A,C,x_{i_k}))C(z(x_1,...,x_n))\end{aligned}$$

The eliminator for the type $\mathcal{M}[\overline{\Theta}]$ is

$$\begin{aligned}E[\overline{\Theta}] \quad : \quad & (C{:}(\mathcal{M}[\overline{\Theta}])\,Type)(f_1{:}\Theta^o_1(\mathcal{M}[\overline{\Theta}],C,\iota_i[\overline{\Theta}]))...(f_n{:}\Theta^o_n(\mathcal{M}[\overline{\Theta}],C,\iota_n[\overline{\Theta}]))\\ & (z{:}\mathcal{M}[\overline{\Theta}])C(z)\end{aligned}$$

Let $Ari(\Theta_i) = \langle \Phi_{i_1},...,\Phi_{i_k}\rangle$ for each Θ_i from the composition of $\overline{\Theta}$. For every constructor we have a computation rule

$$\begin{aligned}E[\overline{\Theta}](C,f,\iota_i(x)) = f_i(\quad & x, \Phi^{\natural}_{i_1}(\mathcal{M}[\overline{\Theta}])(C,E[\overline{\Theta}](C,f),x_{i_1}),...,\\ & \Phi^{\natural}_{i_k}(\mathcal{M}[\overline{\Theta}])(C,E[\overline{\Theta}](C,f),x_{i_k})\\ & \qquad){:}C(\iota_i(x))\end{aligned}$$

for $f = f_1,...,f_n$ and $x = x_1,...,x_n$.

The eliminator of

$$Nat = \mathcal{M}[X,(X)X]$$

is indeed the *rec* constant, however, for

$$\Pi = [A{:}Type][B{:}(A)\,Type]\mathcal{M}[((x{:}A)B(x))X]$$

observe that

$$app{:}(A{:}\,Type)(B{:}(A)\,Type)(\Pi(A,B))(x{:}A)B(x)$$

does not satisfy the definition of the eliminator given above. To introduce Π-type through schemata we need to give instead the eliminator

$$\begin{aligned} E_\Pi \quad &: \quad (A{:}\,Type)(B{:}(A)\,Type)(C{:}(\Pi(A,B))\,Type) \\ & \quad\ ((f : (x{:}A)B(x))C(\lambda(A,B,f)))(z{:}\Pi(A,B))C(z) \end{aligned}$$

and the computation rule will then become

$$E_\Pi(A,B,C,f,\lambda(A,B,g)) = f(g){:}C(\lambda(A,B,g))$$

This is a stronger way of introducing Π type and we can recover the *app* operator by defining it as

$$app(A,B,F,a) = E_\Pi(A,B,C,[G{:}\Pi(A,B)]B(a),[g{:}(x{:}A)B(x)]g(a),F)$$

in which case we obtain the computation rule used earlier.

Similarly some other types are

- The empty type: $\emptyset = \mathcal{M}[]$,
- The type of lists: $List = [A{:}\,Type]\mathcal{M}[X,(A)(X)X]$,
- The type of dependent pairs:
  $\Sigma = [A{:}\,Type][B{:}(A)\,Type]\mathcal{M}[(x{:}A)(B(x))X]$.

I mentioned before that Π is the dependent function type. The function type is just

$$\longrightarrow = [A{:}\,Type][B{:}\,Type]\mathcal{M}[((A)B)X]$$

Likewise Σ is the dependent pairs type and the constant pairs type is just

$$\times = [A{:}\,Type][B{:}\,Type]\mathcal{M}[(A)(B)X]$$

2.1.3 Meta-theoretic properties of LF

LF has certain properties which are highly desirable due to decidability of type checking. In this subsection, I list them and explain what they mean. I will use the notion of *reduction.* For a detailed presentation of reduction see Goguen [Gog94] but just for an intuition, an example of reduction in LF is given by the β-conversion rule.

Canonicity Property. This property states that every object of a type reduces to a canonical object of that type. The canonical objects of a type, also called values, are generated by the constructors of that type. For example, for natural numbers, $0, 1, 2...$ are all canonical objects. $2 + 2$ reduces to 4 which is a canonical object of Nat. For Π type, the canonical objects are λ terms. For instance for $\Pi(Nat, \lambda n{:}Nat.Vect(n))$, the type of programs that for a natural number n give a vector of length n, will be terms of the form $\lambda(Nat, \lambda n{:}Nat.Vect(n), f)$ where $f{:}(n{:}Nat)Vect(n)$.

Subject Reduction This property states that if a reduces to b and $a{:}A$, then $b{:}A$ as well. For example if we consider $f{:}(Nat)Nat$ with $f \equiv \lambda n{:}Nat.2 + n$, then $f(2)$ is of type Nat, and through β conversion it reduces to $2 + 2$ which also has to be of type Nat.

Strong normalization Every sequence of rewriting terminates with a canonical object. Again, if we consider the example above, $f(2) =_\beta [2/n]f(n) = 2 + 2 = 4$ is a sequence of rewriting and it terminates with a normal form.

Formally, for the rewriting $2+2 = 4$, we define the sum of natural numbers inductively as follows

1. $m + 0 = m$
2. $m + Succ(n) = Succ(m + n)$

In this case $2 + 2 = 2 + Succ(1) = Succ(2 + 1) = Succ(2 + Succ(0)) = Succ(Succ(2 + 0)) = Succ(Succ(2)) = Succ(3) = 4$ is a reduction sequence.

2.1.4 Universes

Girard [Gir72] showed that a type theory with *type of all types* becomes inconsistent in that all its formulas become provable. However, when adding inductive types to a type theory like Martin-Löf's Type Theory or Luo's UTT, one needs a type of types which satisfies reflection principle, namely it is closed to formation of these inductive types. Martin-Löf [ML98, ML75, ML82, ML84] introduced two kinds of universes for his intuitionistic type theory to satisfy these needs without adding the paradox. More precisely, a universe is a type whose objects are types closed to formation of inductive types and it is not an object of itself. Universes can be used to prove that the constructors of a type are distinct. Smith [Smi88] showed this property cannot be proved without universes. Practical applications of universes include constructive formalization of category theory, structured specifications of programs and abstract mathematical structures. I mentioned in the previous subsection that UTT adds a type $Prop\!:\!Type$. This is itself a universe, an impredicative one, in that we can always form $\forall(Prop, P)$.

The two kinds of universes are Russell style and Tarski style. In what follows I shall present both types of universes.

Russell style universes

Russell style is easy to use and expressive. It has been adopted in projects like Homotopy Type Theory [Uni13]. Russell style can be found in Extended Calculus of Constructions [Luo90] and Coq [Coq10] which implements Calculus of Inductive Constructions [CP90, PM93].

The rules for Russell style universes are the ones in Figure 2.4 (and in Figure B.1 of the Appendix B).

for $i \in \omega$

$$\frac{\Gamma\ \mathit{valid}}{\Gamma \vdash U_i : \mathit{Type}} \qquad \frac{\Gamma \vdash A : U_i}{\Gamma \vdash A : \mathit{Type}} \qquad \frac{\Gamma\ \mathit{valid}}{\Gamma \vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i}{\Gamma \vdash A : U_{i+1}}$$

Figure 2.4: Inference Rules for Russell Style Universes

Observe that we have the hierarchy

$$U_0{:}U_1{:}...{:}U_n$$

If we add Π type to the system, the rule corresponding to it will be

$$\frac{\Gamma \vdash A : U_i \quad \Gamma \vdash B : (A)U_i}{\Gamma \vdash \Pi(A, B) : U_i}$$

If we add Σ type, the corresponding rule is

$$\frac{\Gamma \vdash A : U_i \quad \Gamma \vdash B : (A)U_i}{\Gamma \vdash \Sigma(A, B) : U_i}$$

Similarly we can add other inductive types.

The last rule of Figure 2.4 represents cumulativity. This allows one to form a type with $A{:}U_i$ and $U_i{:}U_{i+1}$, such as pairs $A \times U_i$ or $\Sigma_{X:U_i}X$ or functions, $A \longrightarrow U$ or $\Pi(U_i, \lambda X{:}U_i.X)$ as a term of U_{i+1}. We do this by simply regarding A and X as a term of U_{i+1}. This way, for example $\Sigma_{X:U_i}X$ arises directly from the rule for Σ type with premise $\Gamma \vdash U_i{:}U_{i+1}$ and $\Gamma \vdash \lambda X{:}U_i.X : (U_i)U_{i+1}$

The cumulativity rule induces a subsumptive relation between U_i and U_j for any $0 \leq i \leq j \leq n$.

Tarski style universes

Tarski style, on the other hand, is richer from a semantic point of view. An example of Tarski style hierarchy is the universes of *UTT*. Plastic [CL01] is an implementation of *LF* with universes.

The rules for Tarski style universes are given in Figure 2.5 (and Figure B.2 of he Appendix B).

In addition to these rules we also require the equation

$$T_{i+1}(t_{i+1}(a)) = T_i(a){:}Type$$

for $i \in \omega$

$$\frac{\vdash \Gamma}{\Gamma \vdash U_i : \mathit{Type}} \qquad \frac{\Gamma \vdash a : U_i}{\Gamma \vdash T_i(a) : \mathit{Type}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash t_{i+1} : (U_i)U_{i+1}}$$

where t_{i+1} are the lifting operators,

$$\frac{\vdash \Gamma}{\Gamma \vdash u_i : U_{i+1}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash T_{i+1}(u_i) = U_i : \mathit{Type}}$$

where u_i is the name of U_i in U_{i+1}

Figure 2.5: Inference Rules for Tarski Style Universes

Observe that we no longer simply regard U_i as a term of U_{i+1}, instead we say that the name of U_i is a term of U_{i+1}. Further, the terms of U_i are not simply terms of U_{i+1} through cumulativity, instead we say that they can be converted to terms of U_{i+1} via the lifting operator. Likewise, we no longer regard its elements as types but just as names of types.

The rule for the names of Π-types is

$$\frac{\Gamma \vdash a : U_i \quad \Gamma,\ x : T_i(a) \vdash b(x) : U_i}{\Gamma \vdash \pi_i(a, b) : U_i}$$

together with the following:

1. And equation stating the type that the name refers to:

$$\Gamma \vdash T_i(\pi_i(a, b)) = \Pi(T_i(a), [x{:}T_i(a)]T_i(b(x))) : \mathit{Type}$$

2. The fact that the name is unique, more precisely the names obtained via lifting and via dependent product rule are equal:

$$\Gamma \vdash t_{i+1}(\pi_i(a, b)) = \pi_{i+1}(t_{i+1}(a), [x{:}T_i(a)]t_{i+1}(b(x))) : U_{i+1}$$

If we want to add Σ types we add the rule

$$\frac{\Gamma \vdash a : U_i \quad \Gamma,\ x : T_i(a) \vdash b(x) : U_i}{\Gamma \vdash \sigma_i(a, b) : U_i}$$

together with the following equations:

1. $\Gamma \vdash T_{i+1}(\sigma_i(a,b)) = \Sigma(T_{i+1}(a), [x{:}T_i(a)]T_{i+1}(b(x))) : \mathit{Type}$

2. $\Gamma \vdash t_{i+1}(\sigma_i(a,b)) = \sigma_{i+1}(t_{i+1}(a), [x{:}T_i(a)]t_{i+1}(b(x))) : U_{i+1}$

Let us now look at what motivated the cumulativity for Russell style universes, namely the possibility to form a type with U_i and its own terms. So what is now the analogous for, say $\Sigma_{X:U_i}X$? Here the name of the type we want to form comes simply from the rule for dependent sum, specifically we have $\sigma_i(u_i, \lambda x{:}T_{i+1}(u_i).t_{i+1}(x))$. Observe that here instead of simply regarding $x{:}U_i$ as a term of U_{i+1}, as we did for Russell style universes, now we convert it via the lifting operator t_{i+1} to such a term. The type we are looking for is simply given by the first equation and it is $\Sigma(T_{i+1}(u_i), [x{:}T_{i+1}(u_i)]T_{i+1}(t_{i+1}(x)))$ which is $\Sigma(U_i, [x{:}T_{i+1}(u_i)]T_{i+1}(t_{i+1}(x)))$, where u_i is the name of U_i in U_{i+1}. The second equation essentially says that if we know how to convert the name of U_i in U_{i+1} to a name in U_{i+2} and $t_{i+1}(x)$, which is a name in U_{i+1} for any $x{:}U_i$ to a name in U_{i+2}, then we can convert the name $\sigma_i(u_i, \lambda x{:}T_{i+1}(u_i).t_{i+1}(x))$ from U_{i+1} to a name in U_{i+2}.

Of course, now it is obvious what I mean by the fact that Russell style universes are simpler to use than Tarski style. However if we follow the subtleties of the discussion above, we notice that it is much more rigorous from the semantic point of view. To begin with, in Russell style we silently interchange using something as type with using the same thing as a term of a universe. This is avoided in Tarski style by using something as a type and a name of it as a term of a universe. Then in Russell style we are forced to employ a cumulative behaviour in order to form certain types. This also makes us silently interchange using something as a term of a universe with using it as a term of a superior universe. In Tarski style we don't have to do this as we can use the lifting operators for such conversions.

Employing Russell style universes with its cumulative feature can lead to some subtle inconsistencies. These problems were discussed by Luo [Luo12b] and are related to the properties of canonicity or subject reduction. On the one hand, if one adopts the standard notation of terms with full type information, for instance, the term $\lambda X{:}U_1.Nat$, where $Nat : U_0$, would be represented as $\lambda(U_1, [\_{:}U_1]U_0, [\_{:}U_1]Nat)$. This term, which is of type $U_0 \to U_0$ (by subsumption, since $U_1 \to U_0 \leq U_0 \to U_0$ by contravariance), is not definitionally equal to a canonical term which is of the form $\lambda(U_0, ...)$. On the other hand, if we employed terms with less typing information like using (a, b) instead of $pair(A, B, a, b)$ to represent pairs, as in HoTT (see Appendix 2 of [Uni13]) not only the property of type uniqueness fails, but we end up with a situation in which a proof term may have incompatible types. For example, for $a : A$ and $A : U$, where U is a type universe, the pair (A, a) has both types $U \times A$ and $\Sigma X{:}U.X$, which are incompatible in the sense that none of them is a subtype of the other. This would lead to undecidability of type checking in a presentation where type checking depends on type inference.

This justifies why Tarski style universes, although more tedious to work with, are preferred to Russell style. We will see in Section 4.2 of Chapter 4 how we can put some of the syntactic difficulties of using Tarski style universes in a black box without having to carry them along in all computations.

2.2 Signatures

We have seen that a judgement in a type theory specified in LF is of the form $\Gamma \vdash J$ where Γ is a sequence of assumptions which keeps track of the kinds assigned to variables. In Figure A.1, we can see that LF provides rules such as

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash k'{:}K' \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]k'{:}[k/x]K'}$$

and

$$\frac{\Gamma, x{:}K \vdash k{:}K'}{\Gamma \vdash [x{:}K]k{:}(x{:}K)K'}$$

for substitution and abstraction of the variables in context. Because of rules like this, if $\Gamma, n{:}Nat, m{:}n * Nat \vdash_\Sigma J$ and $\Gamma \vdash 2{:}Nat$ are derivable judgements we can substitute n with 2 and derive $\Gamma, m{:}2 * Nat \vdash [2/n]J$. Similarly, if $\Gamma, n{:}Nat \vdash_\Sigma m{:}n * Nat$ is a derivable judgement we could abstract over n and derive the judgement $\Gamma \vdash_\Sigma [n{:}Nat]m{:}\Pi(n{:}Nat, n * Nat)$.

Sometimes it makes sense to declare some assumptions as constants and not allow substitution and abstraction on them. For a clear distinction between assumptions about variables and assumptions about constants, Harper et al.[HHP93] introduced signatures in Edinburgh Logical Framework. Judgements in the system with signatures look like

1. Σ *sig* which states that Σ a valid signature,

2. $\vdash_\Sigma \Gamma$ which states that Γ is a valid context under the signature Σ

3. $\Gamma \vdash_\Sigma K$ which states that K is a valid kind under the signature Σ and context Γ

4. $\Gamma \vdash_\Sigma k{:}K$ which states that k is a term of kind K under the signature Σ and context Γ

In this thesis I will use Σ *valid* instead of Σ *sig*.

In the formulation from [HHP93], initially contexts are sequences of type assignments to variables and signatures are sequences of type and kinds assignments to constants and later in the paper they prove that a type theory with signatures is essentially equivalent with a type theory with context obtained from gluing signatures and contexts. This suggests that signatures can be seen simply as prefixes of contexts. More concretely, if we consider their logical framework with signatures λP_S and the logical framework λP, the one with contexts only, we have

1. Σ *valid* is derivable in λP_S if and only if $\vdash \Sigma$ is derivable in λP.

2. $\vdash_\Sigma \Gamma$ is derivable in λP_S if and only if $\vdash \Sigma, \Gamma$.

3. $\Gamma \vdash_\Sigma J$ is derivable in λP_S if and only if $\Sigma, \Gamma \vdash J$ is derivable in λP.

Note that, here, I also use $\vdash \Gamma$ instead of their $\Gamma \vdash \mathit{Type}$.

I will use the same approach for keeping track of constants in LF from [Luo94]. Note however that there are some subtle differences between LF, the logical framework from [Luo94] and λP, the Edinburgh Logical Framework from [HHP93], such as the η rule which holds for LF but not for λP.

2.3 Subtyping

In this section I shall give more details about subsumptive subtyping. After this I will also present in more details coercive subtyping as it was previously formulated for LF, the logical framework from [Luo94], and the motivations.

2.3.1 Subsumptive Subtyping

Subsumptive subtyping is typically represented through the idea that all objects of a type are also objects of its supertypes. I shall denote the fact that A is a subtype of A' under context Γ by the judgement $\Gamma \vdash A \leq A'$. The rule that represents the idea of subsumptive subtyping is

$$\frac{\Gamma \vdash M{:}A \quad \Gamma \vdash A \leq A'}{\Gamma \vdash M{:}A'}$$

I will refer to this as subsumption rule.

Some of the work done on subtyping for dependent type theories is by Betarte and Tasistro [BT98] for Martin-Löf's logical framework analyzing subkinding between kinds (called types), Barthe and Frade [BF99] on constructor subtyping and Aspinall and Compagnoni [AC01] on a form of subsumptive subtyping.

Because the latter inspires one of the practical situation that I will formally represent in the system I introduce here, the one in Section 4.1.1, and it also gives an understanding of how one can add subtyping for systems specified in logical frameworks, I will briefly present it first. I will finish this subsection by giving an intuition about how constructor subtyping works, this being another source of inspiration for practical situation discussed in Section4.3.

λP

The system developed by Aspinall and Compagnoni [AC01] is an extension of Edinburgh Logical Framework (λP) [HHP93] initially with a subsumption rule. This system introduces subtyping through contexts which now contains, apart from membership entries also subtyping entries $\alpha \leq A$. Aspinall and Compagnoni [AC01] initially formulates a system which extends λP to $\lambda P_{\leq}$ which add rules essentially like those in Figure 2.6.

General Subtyping Rules

$$\frac{\Gamma \vdash A{:}K \quad \Gamma \vdash B{:}K \quad \Gamma \vdash A =_{\beta} B}{\Gamma \vdash A \leq B} \qquad \frac{\Gamma \vdash A \leq B \quad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C}$$

$$\frac{\Gamma \vdash a{:}A \quad \Gamma \vdash A \leq A'}{\Gamma \vdash a{:}A'}$$

Subtyping in Contexts

$$\frac{\Gamma \vdash A{:}K \quad \alpha \notin FV(\Gamma)}{\vdash \Gamma, \alpha \leq A{:}K} \qquad \frac{\vdash \Gamma, \alpha \leq A, \Gamma'}{\Gamma, \alpha \leq A, \Gamma' \vdash \alpha{:}\,\mathit{Type}} \qquad \frac{\vdash \Gamma, \alpha \leq A, \Gamma'}{\Gamma, \alpha \leq A, \Gamma' \vdash \alpha \leq A{:}\,\mathit{Type}}$$

Dependent Product

$$\frac{\Gamma \vdash A' \leq A \quad \Gamma, x{:}A' \vdash B \leq B'}{\Gamma \vdash \Pi(A, B) \leq \Pi(A', B')}$$

$$\frac{\Gamma, x{:}A \vdash B \leq B'}{\Gamma \vdash \lambda x{:}A.B \leq \lambda x{:}AB'} \qquad \frac{\Gamma \vdash B \leq B' \quad \Gamma \vdash B'M{:}K}{\Gamma \vdash BM \leq B'M}$$

Figure 2.6: Subtyping Rules for $\lambda P_{\leq}$

Note that there are some differences between λP and LF. In the first place LF has rules to derive definitional equality, including β and η rules, whereas λP leaves definitional equality and β conversion at meta-level and η rule does not hold here.

The first three rules are rules for equality, transitivity and the subsumption rule and the next three enable us to form assumptions with subtyping and to consume them. Then we have the rule for dependent product, lambda terms and application. The system also contains the rules inherited from λP among which is the rule for abstraction,

$$\frac{\Gamma, x{:}A \vdash M{:}B}{\Gamma \vdash \lambda x{:}A.M{:}\Pi(A, \lambda x{:}A.B(x))}$$

They do not add rules for abstracting over subtyping entries, concretely, there are no such rules as

$$\frac{\Gamma, \alpha \leq A \vdash M{:}B}{\Gamma \vdash \lambda\alpha \leq A.M{:}\Pi\alpha \leq A.B}$$

Similarly whereas the rule

$$\frac{\Gamma, x{:}A \vdash B \leq B'}{\Gamma \vdash \Pi(A, B) \leq \Pi(A, B')}$$

is a particular case of the subtyping for dependent product rule when $A \equiv A'$, there is no such rule as

$$\frac{\Gamma \vdash A' \leq A \quad \Gamma, \alpha \leq A' \vdash B \leq B'}{\Gamma \vdash \Pi(\alpha \leq A, B) \leq \Pi(\alpha \leq A', B')}$$

This last rule has been proven by Pierce [Pie93] to make the subtyping relation undecidable for system F and hence the type checking.

The important point here is, that whenever we have a subtyping entry in a context it blocks abstraction of all the variables occurring before it in the context. This is because we do not have rules to move the subtyping entry from context and the abstraction rules only apply when the last entry of the context is a membership one. This observation is important because it poses the question, how could we go around the undecidability noticed by Pierce and at the same time be able to abstract freely over the variables in contexts? Contexts do not seem to be suitable to keep track of subtyping entries so an important point of this thesis is to find their suitable place.

Note that, as discussed by Aspinall and Companioni [AC01], we are not able to prove the subject reduction property for the system with subsumption rule. The reason why proving subject reduction for dependent types subtyping relation is complicated is due to the inability to prove that, if $\Gamma \vdash \Pi(A, B) \leq \Pi(A', B')$ then $\Gamma \vdash A' \leq A$ and $\Gamma, x{:}A' \vdash B \leq B'$. Aspinall and Companion solve this with the algorithmic system they developed in [AC01] which drops some subtyping rules, among which is also the subsumption rule. In exchange they change some rules inherited from λP, such as

$$\frac{\Gamma \vdash M{:}\Pi x{:}A.B \quad \Gamma \vdash N{:}A}{\Gamma \vdash MN{:}[N/x]B}$$

which now becomes

$$\frac{\Gamma \vdash M{:}\Pi x{:}A.B \quad \Gamma \vdash N{:}A' \quad \Gamma \vdash A' \leq A}{\Gamma \vdash MN{:}[N/x]B}$$

which essentially allows polymorphic application.

Constructor Subtyping

Constructor subtyping, studied by Barthe and Frade [BF99] is another form of subsumptive subtyping in which an (inductive) type is considered to be a subtype of another if the latter has more constructors than the former. For example, if we consider Even Numbers as a subtype of *Nat* with the argument that the constructors of *Even* are 0 and successor of *Odd*, where *Odd* is the type of Odd Numbers given by the constructor successor of *Even*. Then, in *Nat* the successor constructor is overloaded to a lifting of these constructors. Formally they write:

```
datatype Odd = S of Even and
Even = 0
      |S of Odd
datatype Nat  = 0
      |S of Nat
```

```
    |S  of  Odd
    |S  of  Even
```

What this essentially says is that, if $A \leq B$ then the constructors of A are among the constructors of B. This is a development which does not refer to a setting similar to LF, but rather to functional programming practices. However, this perspective is a very interesting as it achieves, through overloading, the same feel for the concept of subtypes as for the concept of subsets, where the values of a set are also values of its supersets.

2.3.2 Coercive Subtyping

I mentioned earlier, in Subsection 2.3.1 that a logical framework and a type theory specified in a logical framework has some useful properties like canonicity, subject reduction and strong normalization. Subsumptive subtyping, if introduced in such a type theory, does not preserve these properties. In particular for inductive types, in order to preserve the canonicity property, values given by constructors of the subtype should reduce to values given by constructors of the supertypes, which does not happen. For example, the empty list of even natural numbers will not reduce to the empty list of natural numbers as noted in [Luo99, Luo96, LSX13, Luo12b].

When talking about a subtyping relation between A and B we need to have a way of understanding terms of A as terms of B. When the types are sets we can simply say that the values of A are values of B, however, when talking about inductive types, this is not trivial anymore. Let us consider $A \leq A'$ and the constant pairs types $A \times A$ and $A' \times A'$. Let us then consider the eliminator operator for $\times$ type applied to $C{:}(A' \times A')\,\mathit{Type}$, $f{:}(x{:}A')(y{:}A')C(\mathit{pair}(A', A', x, y))$, $a{:}A$ and $b{:}A$

$$\mathit{Elim}(A', A', C, f, \mathit{pair}(A, A, a, b)){:}C(\mathit{pair}(A, A, a, b))$$

By the computation rule this reduces to

$$f(a, b){:}C(pair(A', A', a, b))$$

However, to prove subject reduction we need to prove that

$$C(pair(A, A, a, b)) = C(pair(A', A', a, b)){:}\,Type$$

which we cannot. The general problem here is that the types that are subject to subtyping occur in objects of types affected by this subtyping and a fix for this will be shown in the next subsection.

Coercive Subtyping through Sets of Judgements

What Luo et al. [Luo96, LSX13, SL02, Luo05, Xue13b] proposed with coercive subtyping is to add a set of initial coercive subtyping judgements and some rules to further infer subtyping judgements. A coercive subtyping judgement is of the form $\Gamma \vdash A <_c B$. For a type theory T, specified in LF, we denote its enrichment with a set of coercive subtyping judgements $\mathcal{C}$ and all the rules that will follow in Figure 2.7 as $T[\mathcal{C}]$.

First note that subkinding is essentially a structural lifting of subtyping. Then, the last three rules (two coercive application rules and one coercive definition rule) in this figure assume if c is a coercion between two kinds then it is a mapping between the two kinds. Also note that if we had two coercions between the same kinds, the last rule would make two terms definitionally equal even if these terms were not definitionally equal in the system without subtyping. This is precisely why, for the system above to be consistent, the restricted system, without these last three rules, denoted by $\mathbf{T}[\mathcal{C}]_{0K}$, would have to satisfy what was introduced in [LSX13, Xue13b] as coherence condition which essentially says that, if c is a coercion between A and A', then c is a mapping from A to A' and it is the only coercion between the two types up to definitional equality. Formally the following three, conditions are satisfied:

Subtyping Rules

$$\frac{\Gamma \vdash A <_c B{:}Type \in \mathcal{C}}{\Gamma \vdash A <_c B{:}Type}$$

Congruence

$$\frac{\Gamma \vdash A <_c B : Type \quad \Gamma \vdash A = A' : Type \quad \Gamma \vdash B = B' : Type \quad \Gamma \vdash c = c' : (A)B}{\Gamma \vdash A' <_{c'} B' : Type}$$

Transitivity

$$\frac{\Gamma \vdash A <_c A' : Type \quad \Gamma \vdash A' <_{c'} A'' : Type}{\Gamma \vdash A <_{c' \circ c} A'' : Type}$$

Weakening

$$\frac{\Gamma, \Gamma' \vdash A \leq_d B : Type \quad \Gamma \vdash K\ kind}{\Gamma, x{:}K, \Gamma' \vdash A \leq_d B : Type} \quad (x \notin dom(\Gamma, \Gamma'))$$

Context Replacement

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash A <_c B\,Type \quad \Gamma_0 \vdash K = K'}{\Gamma_0, x{:}K', \Gamma_1 \vdash A <_c B\,Type}$$

Substitution

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash A <_c B\,Type \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]A <_{[k/x]c} [k/x]B{:}Type}$$

Basic Subkinding Rule and Identity

$$\frac{\Gamma \vdash A <_c B{:}Type}{\Gamma \vdash El(A) <_c El(B)} \qquad \frac{\Gamma \vdash K\ kind}{\Gamma \vdash K <_{[x:K]x} K}$$

Structural Subkinding Rules

$$\frac{\Gamma \vdash K_1 <_c K_2 \quad \Gamma \vdash K_1 = K_1' \quad \Gamma \vdash K_2 = K_2' \quad \Gamma \vdash c = c'{:}(K_1)K_2}{\Gamma \vdash K_1' <_{c'} K_2'}$$

$$\frac{\Gamma \vdash K <_c K' \quad \Gamma \vdash K' <_{c'} K''}{\Gamma \vdash K <_{c' \circ c} K''}$$

$$(x \notin dom(\Gamma, \Gamma')) \qquad \frac{\Gamma, \Gamma' \vdash K \leq_d K' \quad \Gamma \vdash K_0\ kind}{\Gamma, x{:}K_0, \Gamma' \vdash K \leq_d K'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash L \leq_d L' \quad \Gamma_0 \vdash K = K'}{\Gamma_0, x{:}K', \Gamma_1 \vdash L \leq_d L'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash K_1 <_c K_2 \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K_1 <_{[k/x]c} [k/x]K_2}$$

Coercive Application

$$(CA_1) \qquad \frac{\Gamma \vdash f{:}(x{:}K)K' \quad \Gamma \vdash k_0{:}K_0 \quad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f(k_0){:}[c(k_0)/x]K'}$$

$$(CA_2) \qquad \frac{\Gamma \vdash f = f'{:}(x{:}K)K' \quad \Gamma \vdash k_0 = k_0'{:}K_0 \quad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f(k_0) = f'(k_0'){:}[c(k_0)/x]K'}$$

Coercive Definition

$$(CD) \qquad \frac{\Gamma \vdash f{:}(x{:}K)K' \quad \Gamma \vdash k_0{:}K_0 \quad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f(k_0) = f(c(k_0)){:}[c(k_0)/x]K'}$$

Figure 2.7: Inference Rules for $\mathbf{T}[\mathcal{C}]$

- $\Gamma \vdash A <_c A$ is not derivable in $\mathbf{T}[\mathcal{C}]_{0K}$ for any $\Gamma \vdash A{:}Type$ and $\Gamma \vdash c{:}(A)A$;

- if $\Gamma \vdash A <_c B$ is derivable in $\mathbf{T}[\mathcal{C}]_{0K}$ then $\Gamma \vdash c{:}(A)B$ is derivable in $\mathbf{T}[\mathcal{C}]_{0K}$;

- if $\Gamma \vdash A <_c B$ and $\Gamma \vdash A <_{c'} B$ are derivable in $\mathbf{T}[\mathcal{C}]_{0K}$ then $\Gamma \vdash c = c'{:}(A)B$ is derivable in $\mathbf{T}[\mathcal{C}]_{0K}$.

Indeed, with these conditions, Luo et al. [LSX13, Xue13b] prove that the judgements in the system such constructed are consistent with the original system (without subtyping), the coercive application rule being nothing more than an abbreviation of the ordinary application. In other words, this formulation enables us to benefit from the ability to use objects of a type wherever objects of a super type are expected.

With this let us examine again the example considered earlier, the one with $A \times A$ as a subtype of $A' \times A'$ whenever A is a subtype of A' and let us rewrite it with coercive subtyping. If we consider $\times$-type then $\mathbf{T}[\mathcal{C}]$ should also have the structural subtyping rule

$$\frac{\Gamma \vdash A <_c A' \quad \Gamma \vdash B <_{c'} B'}{\Gamma \vdash A \times B <_{(c,c')} A' \times B'}$$

where $(c, c')(pair(A, B, a, b)) = pair(A', B', c(a), c'(b)){:}A' \times B'$ as studied in [LA08]. Now, if $A \leq_c A$ and $A \times A \leq_d A' \times A'$, with (CA_1), we infer

$$C(pair(A, A, a, b)) = C(d(pair(A, A, a, b))){:}\,Type$$

with (CD), we infer

$$C(d(pair(A, A, a, b))) = C((c, c)(pair(A, A, a, b))){:}\,Type$$

and, with (CA_1) again, we infer

$$C(pair(A', A', a, b)) = C(pair(A', A', c(a), c(b))){:}\,Type$$

which at last proves

$$C(pair(A, A, a, b)) = C(pair(A', A', a, b)) : Type$$

The system presented above offers an alternative to the notion of subsumptive subtyping which is correct and adequate from theoretical point of view. However, in practice, for example for proof assistants which use such type theories there are two aspects in which this theoretical development is not completely satisfactory.

In the first place if we look at the rules in Figure 2.7 we will see that deciding subtyping relation between two types eventually amounts to deciding whether a judgement or more belong to a set, which in theory can be infinite so there is no formal guarantee on the decidability of subtyping relation.

The second point is that there is a presentation difference between proof assistants which support coercive subtyping such as Coq [Coq10] and the system from [LSX13, Xue13b]. To relate two types via subtyping, in Coq, one annotates a predefined map coercion as part of the assumptions. The system from [LSX13, Xue13b] adds the coercions to the system through a separate set of coercive subtyping judgements and then, in order for the obtained system to be consistent, it asks for the fulfillment the coherence condition, which includes that coercions are mappings.

Local Coercions

I mentioned that the approach of adding coercive subtyping through sets of coercions does not entirely reflect a possible programming model of tools that use a dependent type theory. Some work to adjust coercive subtyping to achieve this objective has been initiated in [LP13] where the authors add coercive subtyping entries in contexts. They named this kind of coercions local coercions. Of course, when adding coercions in contexts the question that follows is how to use these assumptions. In [LP13] this was expressed through the rule

$$\frac{\Gamma, A \leq_c B \vdash k{:}K}{\Gamma \vdash (\mathit{coercion}\ A \leq_c B\ \mathit{in}\ k){:}(\mathit{coercion}\ A \leq_c B\ \mathit{in}\ K)}$$

which enables us to move coercions to the right of the $\vdash$ or else they would obstruct the abstraction for anything in their left.

The system such presented, although powerful, has a difficult meta-theory. For instance, to make this work one should consider additional computation rules like

$$\frac{\Gamma, A \leq_c B\ \ \mathit{valid} \quad \Gamma \vdash J}{\Gamma \vdash J = (\mathit{coercion}\ A \leq_c B\ \mathit{in}\ J)}$$

For example for $\Gamma \vdash (\mathit{coercion}\ A \leq_c B\ \mathit{in}\ k) = k{:}(\mathit{coercion}\ A \leq_c B\ \mathit{in}\ K)$ and $\Gamma \vdash (\mathit{coercion}\ A \leq_c B\ \mathit{in}\ k)(\mathit{coercion}\ A \leq_c B\ \mathit{in}\ l) = (\mathit{coercion}\ A \leq_c B\ \mathit{in}\ kl){:}[(\mathit{coercion}\ A \leq_c B\ \mathit{in}\ l)/x](\mathit{coercion}\ A \leq_c B\ \mathit{in}\ K)$.

Although, it was claimed earlier (Luo et al. [LL01, LLS02]) that coercive subtyping is a general approach to subtyping, it has been presented over the years as an alternative to subsumptive subtyping and it has not been explained in what way it is a generalization and the relation between the two forms of subtyping has not yet been discussed. What I propose in this thesis is a system that is able to represent some practical situations of subtyping. On the one hand it represents some subsumptive subtyping systems as particular cases of coercive subtyping, which intuitively is the case, and on the other hand it offers a type theory with subtyping entries at assumption level which is close to the programming model of proof assistants like Coq [Coq10] without becoming too difficult to work with.

Chapter 3

Coercive Subtyping in Signatures

In this chapter I introduce a system with coercive subtyping at assumption level, specifically in signatures, and I prove its consistency.

As mentioned in the previous chapter the aim is to formulate a system which achieves a balance between a reasonably easy meta-theory and being able to represent practical situations. Essentially the aim is a system which exhibits the same level of theoretical correctness as the system in [LSX13, Xue13b] but in which subtyping relations are introduced as part of assumptions rather than through a set of judgements.

Previous work done to introduce subtyping relations as assumption can be found in Aspinall and Compagnoni [AC01], for subsumptive subtyping and Luo and Part [LP13] for coercive subtyping. Both these approaches add subtyping in contexts. The first one has entries of the form $\alpha \leq A$ with α a variable and poses restrictions on abstraction and substitution of the variables occurring in the context before the last subtyping entry, as discussed in Subsection 2.3.1. The second has entries of the form $A \leq B$ in the context with techniques to move such an entry to the righthand side of the $\vdash$ which complicate the meta-theory, as discussed in the Subsection 2.3.2. I argue that this complication is unnecessary to represent certain practical situation and a much simpler version with coercive subtyping entries in signatures is enough.

Why Signatures? Because we don't abstract or substitute entries of signatures and these operations are not affected in any way by the presence of subtyping entries in the signatures and we don't have to find a way to move them to the righthand side of the $\vdash$ sign. Signatures were first introduced in [HHP93] to differentiate between the kinds assigned to constants and kinds assigned to variables.

In this chapter, first I introduce the logical framework with signatures, then, for a type system specified in this logical framework, I present what it means to extend it with coercive subtyping and I finish by giving a proof of adequacy of such an extension.

An important part of the work presented in this chapter and in the next chapter is also presented in [LLss]. The system introduced here was also mentioned in a talk given at BCTCS'17.

3.1 LF_S

LF_S is a logical framework with signatures obtained from Luo's logical framework, *LF* [Luo94] by adding signatures and inference rules for signature validity and assumption in signatures. In addition, it also has weakening and context and signature replacement as rules, as in the formulation of *LF* given by Luo et al. [SL02, LSX13, Xue13b]. In LF_S, there are six forms of judgements:

- $\langle\rangle$ *valid* is the signature of length 0
- Σ *valid*, asserting that Σ is a valid signature.
- $\vdash_\Sigma \Gamma$, asserting that Γ is a valid context under Σ.
- $\Gamma \vdash_\Sigma K\ kind$, asserting that K is a kind in Γ under Σ.
- $\Gamma \vdash_\Sigma k : K$, asserting that k is an object of kind K in Γ under Σ.
- $\Gamma \vdash_\Sigma K_1 = K_2$, asserting that K_1 and K_2 are equal kinds in Γ under Σ.

- $\Gamma \vdash_\Sigma k_1 = k_2 : K$, asserting that k_1 and k_2 are equal objects of kind K in Γ under Σ.

The inference rules of the logical framework LF_S are given in Figure 3.1.

3.2 $T_{S,\leq}$

To a type theory T_S, specified in LF_S, I add subtyping in signatures by extending it with the form of judgement

$$\Gamma \vdash_\Sigma A \leq_c B : Type$$

to represent subtyping. As we have seen, whenever $\Gamma \vdash_\Sigma \vdash A{:}Type$ is derivable we also have the derivable judgement $\Gamma \vdash_\Sigma \vdash El(A)\ kind$. In what follows I will often omit $El()$ and use A to also refer to the kind of the elements of the type A.

Subtyping, not subkinding relations, can be specified in a signature by means of entries $A \leq_c B : Type$ (or simply written as $A \leq_c B$), where A and B are types and $c : (A)B$. To infer this judgement and to use it, I add the rules in Figure 3.2.

I also add the form of judgement

$$\Gamma \vdash_\Sigma K \leq_c L$$

to represent subkinding judgements and the corresponding rules in Figure 3.3 to infer and use such judgements.

Let the system composed by the rules in Figures 3.1, 3.2 and 3.3 be denoted by $\mathbf{T}^{0K}_{S,\leq}$. All the rules for it are also put together in the Appendix C. As in the formulation from [LSX13, Xue13b], the specifications of subtyping relations are also required to be coherent. Coherence is crucial as it ensures a coercive application abbreviates a unique functional application. Here is the definition of coherence of a signature, which intuitively says that, under a coherent signature, there cannot be two different coercions between the same types.

Validity of Signature/Contexts, Assumptions

$$\frac{}{\langle\rangle\ \mathit{valid}} \qquad \frac{\vdash_\Sigma K\,\mathit{kind} \quad c \notin \mathit{dom}(\Sigma)}{\Sigma, c{:}K\ \ \mathit{valid}} \qquad \frac{\vdash_{\Sigma,c:K,\Sigma'} \Gamma}{\Gamma \vdash_{\Sigma,c:K,\Sigma'} c{:}K}$$

$$\frac{\Sigma\ \ \mathit{valid}}{\vdash_\Sigma \langle\rangle} \qquad \frac{\Gamma \vdash_\Sigma K\,\mathit{kind} \quad x \notin \mathit{dom}(\Sigma) \cup \mathit{dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}K} \qquad \frac{\vdash_\Sigma \Gamma, x{:}K, \Gamma'}{\Gamma, x{:}K, \Gamma' \vdash_\Sigma x{:}K}$$

Weakening

$$\frac{\Gamma \vdash_{\Sigma,\ \Sigma'} J \quad \vdash_\Sigma K\ \mathit{kind} \quad c \notin \mathit{dom}(\Sigma, \Sigma')}{\Gamma \vdash_{\Sigma,\ c:K,\ \Sigma'} J}$$

$$\frac{\Gamma, \Gamma' \vdash_\Sigma J \quad \Gamma \vdash_\Sigma K\ \mathit{kind} \quad x \notin \mathit{dom}(\Gamma, \Gamma')}{\Gamma, x{:}K, \Gamma' \vdash_\Sigma J}$$

Equality Rules

$$\frac{\Gamma \vdash_\Sigma K\ \ \mathit{kind}}{\Gamma \vdash_\Sigma K = K} \qquad \frac{\Gamma \vdash_\Sigma K = K'}{\Gamma \vdash_\Sigma K' = K} \qquad \frac{\Gamma \vdash_\Sigma K = K' \quad \Gamma \vdash_\Sigma K' = K''}{\Gamma \vdash_\Sigma K = K''}$$

$$\frac{\Gamma \vdash_\Sigma k{:}K}{\Gamma \vdash_\Sigma k = k{:}K} \qquad \frac{\Gamma \vdash_\Sigma k = k'{:}K}{\Gamma \vdash_\Sigma k' = k{:}K} \qquad \frac{\Gamma \vdash_\Sigma k = k'{:}K \quad \Gamma \vdash_\Sigma k' = k''{:}K}{\Gamma \vdash_\Sigma k = k''{:}K}$$

$$\frac{\Gamma \vdash_\Sigma k{:}K \quad \Gamma \vdash_\Sigma K = K'}{\Gamma \vdash_\Sigma k{:}K'} \qquad \frac{\Gamma \vdash_\Sigma k = k'{:}K \quad \Gamma \vdash_\Sigma K = K'}{\Gamma \vdash_\Sigma k = k'{:}K'}$$

Signature Replacement

$$\frac{\Gamma \vdash_{\Sigma_0,c:L,\Sigma_1} J \quad \vdash_{\Sigma_0} L = L'}{\Gamma \vdash_{\Sigma_0,c:L',\Sigma_1} J}$$

Context Replacement

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma J \quad \Gamma_0 \vdash_\Sigma K = K'}{\Gamma_0, x{:}K', \Gamma_1 \vdash_\Sigma J}$$

Substitution Rules

$$\frac{\vdash_\Sigma \Gamma_0, x{:}K, \Gamma_1 \quad \Gamma_0 \vdash_\Sigma k{:}K}{\vdash_\Sigma \Gamma_0, [k/x]\Gamma_1}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma K'\ \ \mathit{kind} \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]K'\ \ \mathit{kind}} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma L = L' \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]L = [k/x]L'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma k'{:}K' \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma l = l'{:}K' \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]l = [k/x]l'{:}[k/x]K'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma K'\ \ \mathit{kind} \quad \Gamma_0 \vdash_\Sigma k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]K' = [k'/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma l{:}K' \quad \Gamma_0 \vdash_\Sigma k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]l = [k'/x]l{:}[k/x]K'}$$

Dependent Product Kinds

$$\frac{\Gamma \vdash_\Sigma K\,\mathit{kind} \quad \Gamma, x{:}K \vdash_\Sigma K'\,\mathit{kind}}{\Gamma \vdash_\Sigma (x{:}K)K'\,\mathit{kind}} \qquad \frac{\Gamma \vdash_\Sigma K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash_\Sigma K_1' = K_2'}{\Gamma \vdash_\Sigma (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Gamma, x{:}K \vdash_\Sigma y{:}K'}{\Gamma \vdash_\Sigma [x{:}K]y{:}(x{:}K)K'} \qquad \frac{\Gamma \vdash_\Sigma K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash_\Sigma k_1 = k_2{:}K}{\Gamma \vdash_\Sigma [x{:}K_1]k_1 = [x{:}K_2]k_2{:}(x{:}K_1)K}$$

$$\frac{\Gamma \vdash_\Sigma f{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k{:}K}{\Gamma \vdash_\Sigma f(k){:}[k/x]K'} \qquad \frac{\Gamma \vdash_\Sigma f = f'{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k_1 = k_2{:}K}{\Gamma \vdash_\Sigma f(k_1) = f'(k_2){:}[k_1/x]K'}$$

$$\frac{\Gamma, x{:}K \vdash_\Sigma k'{:}K' \quad \Gamma \vdash_\Sigma k{:}K}{\Gamma \vdash_\Sigma ([x{:}K]k')(k) = [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma \vdash_\Sigma f{:}(x{:}K)K' \quad x \notin FV(f)}{\Gamma \vdash_\Sigma [x{:}K]f(x) = f{:}(x{:}K)K'}$$

The kind Type

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathit{Type}\ \ \mathit{kind}} \qquad \frac{\Gamma \vdash_\Sigma A{:}\mathit{Type}}{\Gamma \vdash_\Sigma \mathit{El}(A)\ \ \mathit{kind}} \qquad \frac{\Gamma \vdash_\Sigma A = B{:}\mathit{Type}}{\Gamma \vdash_\Sigma \mathit{El}(A) = \mathit{El}(B)}$$

Figure 3.1: Inference Rules for LF_S

Signature Rules for Subtyping

$$\frac{\vdash_\Sigma A : \mathit{Type} \quad \vdash_\Sigma B : \mathit{Type} \quad \vdash_\Sigma c : (A)B}{\Sigma, A \leq_c B \ \ \mathit{valid}} \qquad \frac{\vdash_{\Sigma_0, A \leq_c B : \mathit{Type}, \Sigma_1} \Gamma}{\Gamma \vdash_{\Sigma_0, A \leq_c B : \mathit{Type}, \Sigma_1} A \leq_c B : \mathit{Type}}$$

Congruence

$$\frac{\Gamma \vdash_\Sigma A \leq_c B : \mathit{Type} \quad \Gamma \vdash_\Sigma A = A' : \mathit{Type} \quad \Gamma \vdash_\Sigma B = B' : \mathit{Type} \quad \Gamma \vdash_\Sigma c = c' : (A)B}{\Gamma \vdash_\Sigma A' \leq_{c'} B' : \mathit{Type}}$$

Transitivity

$$\frac{\Gamma \vdash_\Sigma A \leq_c A' : \mathit{Type} \quad \Gamma \vdash_\Sigma A' \leq_{c'} A'' : \mathit{Type}}{\Gamma \vdash_\Sigma A \leq_{c' \circ c} A'' : \mathit{Type}}$$

Weakening

$$\frac{\Gamma \vdash_{\Sigma,\ \Sigma'} A \leq_d B : \mathit{Type} \quad \vdash_\Sigma K \ \mathit{kind}}{\Gamma \vdash_{\Sigma,\ c:K,\ \Sigma'} A \leq_d B : \mathit{Type}} \quad (c \notin \mathit{dom}(\Sigma, \Sigma'))$$

$$\frac{\Gamma, \Gamma' \vdash_\Sigma A \leq_d B : \mathit{Type} \quad \Gamma \vdash_\Sigma K \ \mathit{kind}}{\Gamma, x{:}K, \Gamma' \vdash_\Sigma A \leq_d B : \mathit{Type}} \quad (x \notin \mathit{dom}(\Gamma, \Gamma'))$$

Signature Replacement

$$\frac{\Gamma \vdash_{\Sigma_0, c:L, \Sigma_1} A \leq_d B : \mathit{Type} \quad \vdash_{\Sigma_0} L = L'}{\Gamma \vdash_{\Sigma_0, c:L', \Sigma_1} A \leq_d B : \mathit{Type}}$$

Context Replacement

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma A \leq_d B : \mathit{Type} \quad \Gamma_0 \vdash_\Sigma K = K'}{\Gamma_0, x{:}K', \Gamma_1 \vdash_\Sigma A \leq_d B : \mathit{Type}}$$

Substitution

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma A \leq_c B \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]A \leq_{[k/x]c} [k/x]B}$$

Identity Coercion

$$\frac{\Gamma \vdash_\Sigma A : \mathit{Type}}{\Gamma \vdash_\Sigma A \leq_{[x:A]x} A : \mathit{Type}}$$

Figure 3.2: Inference Rules for Subtyping in $T^{0K}_{S,\leq}$ (1)

Definition 5. *A signature Σ is* ***coherent*** *if, in $\mathbf{T}^{0K}_{S,\leq}$, $\Gamma \vdash_\Sigma A \leq_c B$ and $\Gamma \vdash_\Sigma A \leq_{c'} B$ imply $\Gamma \vdash_\Sigma c = c' : (A)B$.*

Note that, in comparison with earlier formulations such as [LSX13, Xue13b], I have switched from strict subtyping relation $<$ to $\leq$ and the coherence condition is changed accordingly as well; in particular, under a coherent signature, any coercion from a type to itself must be equal to the identity function. This is a special case of the above condition when $B \equiv A$: because we always have $A \leq_{[x:A]x} A$, if $A \leq_c A$, then $c = [x{:}A]x : (A)A$. Note also that, it is easy to prove by induction that, if $\Gamma \vdash_\Sigma A \leq_c B : \mathit{Type}$, then $\Gamma \vdash_\Sigma A, B : \mathit{Type}$ and

Basic Subkinding Rule and Identity Coercion

$$\frac{\Gamma \vdash_\Sigma A \leq_c B : \mathit{Type}}{\Gamma \vdash_\Sigma \mathit{El}(A) \leq_c \mathit{El}(B)} \qquad \frac{\Gamma \vdash_\Sigma K \ \mathit{kind}}{\Gamma \vdash_\Sigma K \leq_{[x:K]x} K}$$

Structural Subkinding Rules

$$\frac{\Gamma \vdash_\Sigma K_1 \leq_c K_2 \quad \Gamma \vdash_\Sigma K_1 = K_1' \quad \Gamma \vdash_\Sigma K_2 = K_2' \quad \Gamma \vdash_\Sigma c = c' : (K_1)K_2}{\Gamma \vdash_\Sigma K_1' \leq_{c'} K_2'}$$

$$\frac{\Gamma \vdash_\Sigma K \leq_c K' \quad \Gamma \vdash_\Sigma K' \leq_{c'} K''}{\Gamma \vdash_\Sigma K \leq_{c' \circ c} K''}$$

$$\frac{\Gamma \vdash_{\Sigma,\ \Sigma'} K \leq_d K' \quad \vdash_\Sigma K_0 \ \mathit{kind}}{\Gamma \vdash_{\Sigma,\ c:K_0,\ \Sigma'} K \leq_d K'} \quad (c \notin \mathit{dom}(\Sigma, \Sigma'))$$

$$\frac{\Gamma, \Gamma' \vdash_\Sigma K \leq_d K' \quad \Gamma \vdash_\Sigma K_0 \ \mathit{kind}}{\Gamma, x{:}K_0, \Gamma' \vdash_\Sigma K \leq_d K'} \quad (x \notin \mathit{dom}(\Gamma, \Gamma'))$$

$$\frac{\Gamma \vdash_{\Sigma_0, c:L, \Sigma_1} K \leq_d K' \quad \vdash_{\Sigma_0} L = L'}{\Gamma \vdash_{\Sigma_0, c:L', \Sigma_1} K \leq_d K'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma L \leq_d L' \quad \Gamma_0 \vdash_\Sigma K = K'}{\Gamma_0, x{:}K', \Gamma_1 \vdash_\Sigma L \leq_d L'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma K_1 \leq_c K_2 \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]K_1 \leq_{[k/x]c} [k/x]K_2}$$

Subkinding for Dependent Product Kind

$$\frac{\Gamma \vdash_\Sigma K_1' \leq_{c_1} K_1 \ \ \Gamma, x{:}K_1 \vdash_\Sigma K_2 \ \mathit{kind} \ \ \Gamma, x'{:}K_1' \vdash_\Sigma K_2' \ \mathit{kind} \ \ \Gamma, x{:}K_1 \vdash_\Sigma [c_1(x')/x]K_2 \leq_{c_2} K_2'}{\Gamma \vdash_\Sigma (x{:}K_1)K_2 \leq_{[f:(x:K_1)K_2][x':K_1']c_2(f(c_1(x')))} (x{:}K_1')K_2'}$$

Figure 3.3: Inference Rules for Subkinding in $T_{S,\leq}^{0K}$ (2)

$\Gamma \vdash_\Sigma c : (A)B$.

It is also important to note the difference between a judgement with signature in the current calculus and that in the calculus employed in [LSX13, Xue13b] where there are no signatures. For example, the signatures $\Sigma_1 \equiv \Sigma_0, A \leq_c B$ and $\Sigma_2 \equiv \Sigma_0, A \leq_d B$ can both be coherent signatures even if the only difference between them are the coercions and it is not the case that $\Gamma \vdash_{\Sigma_0} c = d{:}(A)B$ is derivable, while such a situation can only be considered in the earlier setting by having two different type systems $T[\mathcal{C}_1]$ and $T[\mathcal{C}_2]$ as we will see in Subsection 3.4.4. Otherwise, if we had $\Gamma \vdash A \leq_c B \in \mathcal{C}$ and $\Gamma \vdash A \leq_d B \in \mathcal{C}$ with the judgement $\Gamma \vdash c = d{:}(A)B$ not being derivable, $\mathcal{C}$ would not be coherent.

At this point, I can add the rules for coercive application and coercive definition in Figure 3.4.

Coercive Application

$$(CA_1) \qquad \frac{\Gamma \vdash_\Sigma f{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k_0{:}K_0 \quad \Gamma \vdash_\Sigma K_0 \leq_c K}{\Gamma \vdash_\Sigma f(k_0){:}[c(k_0)/x]K'}$$

$$(CA_2) \qquad \frac{\Gamma \vdash_\Sigma f = f'{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k_0 = k_0'{:}K_0 \quad \Gamma \vdash_\Sigma K_0 \leq_c K}{\Gamma \vdash_\Sigma f(k_0) = f'(k_0'){:}[c(k_0)/x]K'}$$

Coercive Definition

$$(CD) \qquad \frac{\Gamma \vdash_\Sigma f{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k_0{:}K_0 \quad \Gamma \vdash_\Sigma K_0 \leq_c K}{\Gamma \vdash_\Sigma f(k_0) = f(c(k_0)){:}[c(k_0)/x]K'}$$

Figure 3.4: The coercive application and definition rules in $T_{S,\leq}$

I denote by $T_{S,\leq}$ the system obtained with the rules in Figures 3.1, 3.2, 3.3 and 3.4. The rules for this system are also listed in the Appendix C.

Again, as in Subsection 2.3.2, when I mentioned coherence as defined in [LSX13, Xue13b] for the system $T[\mathcal{C}]$, note that coherence condition only makes sense for the signatures of the system $T^{0K}_{S,\leq}$ and not $T_{S,\leq}$. The reason is that the coercive definition rule (CD) will force any two coercions to be equal, therefore, if I defined the notion of coherence for the system including the (CD) rule, every signature would be coherent by definition but inconsistent. For instance, let the judgements $\Gamma \vdash c{:}(A)B$ and $\Gamma \vdash d{:}(A)B$ be derivable in T_S such that the judgement $\Gamma \vdash c = d{:}(A)B$ is not derivable in T_S. If the judgements $\Gamma \vdash_\Sigma A \leq_c B$ and $\Gamma \vdash_\Sigma A \leq_d B$ are derivable in $T_{S,\leq}$ and coherence was not be defined before introducing this system, then, with the coercive definition rule and symmetry and transitivity of defintional equality we could derive $\Gamma \vdash_\Sigma f(c(k_0)) = f(d(k_0)){:}[k_0/x]C$ in $T_{S,\leq}$ for any f, C and k_0 such that $\Gamma \vdash_\Sigma f{:}(x{:}B)C$ and $\Gamma \vdash_\Sigma k_0{:}A$ are derivable in $T_{S,\leq}$. In particular we could have some f, C and k_0 such that $\Gamma \vdash_\Sigma f{:}(x{:}B)C$ and $\Gamma \vdash_\Sigma k_0{:}A$ are derivable in T_S and $\Gamma \vdash_\Sigma f(c(k_0)) = f(d(k_0)){:}[k_0/x]C$ would be syntactically in T_S but not derivable in this system. The extension $T_{S,\leq}$ would then be inconsistent with the original system.

3.3 Meta-theoretic properties of $T_{S,\leq}$

In this section I present some of the results related to the consistency of the system $\mathbf{T}_{S,\leq}$ as an extension of T_S.

3.3.1 Coherence for Kinds

Note that the coherence definition 5 refers to types. In what follows I prove that coherence for types implies coherence for kinds. I first categorise kinds and show that they can be related via definitional equality or subtyping only if they are of the same category. For this, I define the degree of a kind K which intuitively denotes how many dependent product occurrences are in K.

The next two lemmas state that a coercive subtyping or subkinding relation between two types or kinds must be associated with an underlying mapping between the two types or kinds.

Lemma 1. *If* $\Gamma \vdash_\Sigma A \leq_c B{:}Type$ *is derivable in* $\mathbf{T}_{S,\leq}$ *then* $\Gamma \vdash_\Sigma c{:}(A)B$ *is derivable in* $\mathbf{T}_{S,\leq}$.

Proof. By induction on the structure of derivations. □

Note that this lemma shows precisely that the second coherence condition according to the definition in [LSX13, Xue13b] which I reproduced in Subsection 2.3.2 is naturally admissible for the system I introduced in this chapter.

Lemma 2. *If* $\Gamma \vdash_\Sigma K \leq_c L$ *is derivable in* $\mathbf{T}^{0K}_{S,\leq}$ *then* $\Gamma \vdash_\Sigma c{:}(K)L$ *is derivable in* $\mathbf{T}^{0K}_{S,\leq}$.

Proof. By induction on the structure of derivations.

- If the last rule of the derivation tree is

$$\frac{\Gamma \vdash_\Sigma A \leq_c B{:}Type}{\Gamma \vdash_\Sigma El(A) \leq_c El(B)}$$

  then we know by the previous lemma that $\Gamma \vdash_\Sigma c{:}(A)B$ is derivable and hence $\Gamma \vdash_\Sigma c{:}(El(A))El(B)$ is derivable.

- If the last rule is
$$\frac{\Gamma \vdash_\Sigma K \ \ kind}{\Gamma \vdash_\Sigma K \leq_{[x:K]x} K}$$
then we already know that $\Gamma \vdash_\Sigma [x{:}K]x{:}(K)K$ is derivable if $\Gamma \vdash_\Sigma K \ \ kind$ is derivable.

- If the last rule is
$$\frac{\Gamma \vdash_\Sigma K_1 \leq_c K_2 \quad \Gamma \vdash_\Sigma K_1 = K_1' \quad \Gamma \vdash_\Sigma K_2 = K_2' \quad \Gamma \vdash_\Sigma c = c'{:}(K_1)K_2}{\Gamma \vdash_\Sigma K_1' \leq_{c'} K_2'}$$
then, by induction hypothesis we have that $\Gamma \vdash_\Sigma c{:}(K_1)K_2$ is derivable and then, by equality rules of T_S we can derive $\Gamma \vdash_\Sigma c'{:}(K_1')K_2'$

- If the last rule is
$$\frac{\Gamma \vdash_\Sigma K \leq_c K' \quad \Gamma \vdash_\Sigma K' \leq_{c'} K''}{\Gamma \vdash_\Sigma K \leq_{c' \circ c} K''}$$
then by induction hypothesis, $\Gamma \vdash_\Sigma c{:}(K)K'$ and $\Gamma \vdash_\Sigma c'{:}(K')K''$ are derivable and from them we can derive $\Gamma \vdash_\Sigma c' \circ c{:}(K)K''$

- If the last rule is one of weakening in signatures like
$$\frac{\Gamma \vdash_{\Sigma,\ \Sigma'} K \leq_d K' \quad \vdash_\Sigma K_0 \ kind}{\Gamma \vdash_{\Sigma,\ c:K_0,\ \Sigma'} K \leq_d K'}$$
with $(c \notin dom(\Sigma, \Sigma'))$, by induction hypothesis we have $\Gamma \vdash_{\Sigma,\ \Sigma'} d{:}(K)K'$ and by weakening rule inherited from LF_S, we can derive $\Gamma \vdash_{\Sigma,\ c:K_0,\ \Sigma'} d{:}(K)K'$. Similarly, if the last rule is weakening in contexts.

- If the last rule is one for signature replacement like
$$\frac{\Gamma \vdash_{\Sigma_0, c:L, \Sigma_1} K \leq_d K' \quad \vdash_{\Sigma_0} L = L'}{\Gamma \vdash_{\Sigma_0, c:L', \Sigma_1} K \leq_d K'}$$
we have by induction hypothesis that $\Gamma \vdash_{\Sigma_0, c:L, \Sigma_1} d{:}(K)K'$ and hence by the signature replacement rule inherited from LF_S, we have that $\Gamma \vdash_{\Sigma_0, c:L', \Sigma_1} d{:}(K)K'$ is derivable. The case for context replacement is

the same.

- If the last rule is

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma K_1 \leq_c K_2 \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]K_1 \leq_{[k/x]c} [k/x]K_2}$$

  then, by induction hypothesis, $\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma c{:}(K_1)K_2$ is derivable and by the substitution rule inherited from LF_S, we can derive $\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]c{:}([k/x]K_1)[k/x]K_2$

- The more interesting case is if $K \equiv (x{:}K_1)K_2$ and $L \equiv (x{:}L_1)L_2$ and a derivation tree for $\Gamma \vdash_\Sigma K \leq_c L$ ends with the rule for dependent product kind with premises $\Gamma \vdash_\Sigma L_1 \leq_{c_1} K_1$, $\Gamma, x{:}K_1 \vdash_\Sigma K_2$ *kind*, $\Gamma, y{:}L_1 \vdash_\Sigma L_2$ *kind* and $\Gamma, y{:}L_1 \vdash_\Sigma [c_1(y)/x]K_2 \leq_{c_2} L_2$.

  By induction hypothesis we have $\Gamma \vdash_\Sigma c_1{:}(L_1)K_1$ and
  $\Gamma, y{:}L_1 \vdash_\Sigma c_2{:}([c_1(y)/x]K_2)L_2$.

  By weakening $\Gamma, f{:}(x{:}K_1)K_2, y{:}L_1 \vdash_\Sigma c_2{:}([c_1(y)/x]K_2)L_2$ and
  $\Gamma, f{:}(x{:}K_1)K_2, y{:}L_1 \vdash_\Sigma c_1{:}(L_1)K_1$.

  We have $\Gamma, f{:}(x{:}K_1)K_2, y{:}L_1 \vdash_\Sigma y{:}L_1$ so by application
  $\Gamma, f{:}(x{:}K_1)K_2, y{:}L_1 \vdash_\Sigma c_1(y){:}K_1$.

  We have $\Gamma, f{:}(x{:}K_1)K_2, y{:}L_1 \vdash_\Sigma f{:}(x{:}K_1)K_2$ so by application we have $\Gamma, f{:}(x{:}K_1)K_2, y{:}L_1 \vdash_\Sigma f(c_1(y)){:}[c_1(y)/x]K_2$. By application again we have $\Gamma, f{:}(x{:}K_1)K_2, y{:}L_1 \vdash_\Sigma c_2(f(c_1(y))){:}L_2$ and by abstraction
  $\Gamma \vdash_\Sigma [f{:}(x{:}K_1)K_2][y{:}L_1]c_2(f(c_1(y))){:}((x{:}K_1)K_2)(y{:}L_1)L_2$.

$\square$

Lemma 3. *If $\Gamma \vdash_\Sigma K \leq_c L$ is derivable in $\mathbf{T}^{0K}_{S,\leq}$ then*

- *if $K \equiv El(A)$ for some A such that $\Gamma \vdash_\Sigma A{:}Type$ is derivable in $\mathbf{T}^{0K}_{S,\leq}$ then $L \equiv El(B)$ for some B such that $\Gamma \vdash_\Sigma B{:}Type$ is derivable in $\mathbf{T}^{0K}_{S,\leq}$.*

- *if $L \equiv El(B)$ for some B such that $\Gamma \vdash_\Sigma B{:}Type$ is derivable in $\mathbf{T}^{0K}_{S,\leq}$ then $K \equiv El(A)$ for some A such that $\Gamma \vdash_\Sigma A{:}Type$ is derivable in $\mathbf{T}^{0K}_{S,\leq}$.*

- *if $K \equiv El(A)$ and $L \equiv El(B)$ then $\Gamma \vdash_\Sigma A \leq_c B{:}Type$ is derivable in $\mathbf{T}^{0K}_{S,\leq}$.*

- *if $K \equiv (x{:}K_1)K_2$ for some K_1 and K_2 such that $\Gamma \vdash_\Sigma K_1$ kind and $\Gamma, x{:}K_1 \vdash_\Sigma K_2$ kind are derivable in $\mathbf{T}^{0K}_{S,\leq}$ then $L \equiv (x{:}L_1)L_2$ for some L_1 and L_2 such that $\Gamma \vdash_\Sigma L_1$ kind and $\Gamma, x{:}L_1 \vdash_\Sigma L_2$ kind are derivable in $\mathbf{T}^{0K}_{S,\leq}$.*

- *if $L \equiv (x{:}L_1)L_2$ for some L_1 and L_2 such that $\Gamma \vdash_\Sigma L_1$ kind and $\Gamma, x{:}L_1 \vdash_\Sigma L_2$ kind are derivable in $\mathbf{T}^{0K}_{S,\leq}$ then $K \equiv (x{:}K_1)K_2$ for some K_1 and K_2 such that $\Gamma \vdash_\Sigma K_1$ kind and $\Gamma, x{:}K_1 \vdash_\Sigma K_2$ kind are derivable in $\mathbf{T}^{0K}_{S,\leq}$.*

Proof. Induction on the structure of derivations of $\Gamma \vdash_\Sigma K \leq_c L$. By induction on the structure of derivations.

- If the last rule of the derivation tree is

$$\frac{\Gamma \vdash_\Sigma A \leq_c B{:}Type}{\Gamma \vdash_\Sigma El(A) \leq_c El(B)}$$

  then we are done.

- If the last rule is

$$\frac{\Gamma \vdash_\Sigma K \ \ kind}{\Gamma \vdash_\Sigma K \leq_{[x:K]x} K}$$

  then it is obvious.

- If the last rule is

$$\frac{\Gamma \vdash_\Sigma K_1 \leq_c K_2 \quad \Gamma \vdash_\Sigma K_1 = K_1' \quad \Gamma \vdash_\Sigma K_2 = K_2' \quad \Gamma \vdash_\Sigma c = c'{:}(K_1)K_2}{\Gamma \vdash_\Sigma K_1' \leq_{c'} K_2'}$$

  then, if $K_1' \equiv El(A')$, for $\Gamma \vdash_\Sigma A'{:}Type$, then $K_1 \equiv El(A)$ for $\Gamma \vdash_\Sigma A{:}Type$ and, by induction hypothesis $K_2 \equiv El(B)$ for $\Gamma \vdash_\Sigma B{:}Type$, hence $K_2' \equiv El(B')$ for $\Gamma \vdash_\Sigma B'{:}Type$. The case for $K_1' \equiv (x{:}L_1')L_2'$ is identical. Likewise are $K_2' \equiv El(B')$ and $K_2' \equiv (x{:}L_1'')L_2''$

For the third case, if $K'_1 \equiv El(A')$ and $K'_2 \equiv El(B')$, then, $K_1 \equiv El(A)$ and $K_2 \equiv El(B)$ with $\Gamma \vdash_\Sigma A = A'{:}Type$ and $\Gamma \vdash_\Sigma B = B'{:}Type$ and by induction hypothesis $\Gamma \vdash_\Sigma A \le B{:}Type$

- For the first two points, if it comes from transitivity from the premises $\Gamma \vdash_\Sigma K \le_{c_0} K_0$ and $\Gamma \vdash_\Sigma K_0 \le_{c_1} L$ with $c \equiv c_1 \circ c_0$ then by induction hypothesis first point, if $K \equiv El(A)$ for some $\Gamma \vdash_\Sigma A{:}\,Type$, then $K_0 \equiv El(C)$ for some $\Gamma \vdash_\Sigma C{:}\,Type$ and further by induction hypothesis second point, $L \equiv El(B)$ for some $\Gamma \vdash_\Sigma B{:}\,Type$ and the statement is proven. The last two point are identical to the first two.

  For the third point if the judgement follows from transitivity from premises $\Gamma \vdash_\Sigma K \le_{c_1} M$ and $\Gamma \vdash_\Sigma M \le_{c_2} L$ with $\Gamma \vdash_\Sigma c = c_2 \circ c_1{:}(K)L$ then by the previous points we have that $M \equiv El(C)$ for some $\Gamma \vdash_\Sigma C{:}\,Type$ and by induction hypothesis $\Gamma \vdash_\Sigma A \le_{c_1} C{:}\,Type$ and $\Gamma \vdash_\Sigma C \le_{c_2} B{:}\,Type$ and we can apply transitivity to obtain $\Gamma \vdash_\Sigma A \le_{c_2 \circ c_1} B{:}\,Type$

- If the last rule is one of weakening in signatures like

$$\frac{\Gamma \vdash_{\Sigma,\ \Sigma'} K \le_d K' \quad \vdash_\Sigma K_0\ kind}{\Gamma \vdash_{\Sigma,\ c:K_0,\ \Sigma'} K \le_d K'}$$

  with $(c \notin dom(\Sigma, \Sigma'))$, it simply follows by induction hypothesis and then weakening. Similarly, if the last rule is weakening in contexts.

- If the last rule is one for signature replacement like

$$\frac{\Gamma \vdash_{\Sigma_0, c:L, \Sigma_1} K \le_d K' \quad \vdash_{\Sigma_0} L = L'}{\Gamma \vdash_{\Sigma_0, c:L', \Sigma_1} K \le_d K'}$$

  again, if follows by induction hypothesis and then signature replacement. The case for context replacement is the same.

- If the last rule is

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma K_1 \le_c K_2 \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]K_1 \le_{[k/x]c} [k/x]K_2}$$

with $[k/x]K_1 \equiv El(A)$ for $\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma A{:}Type$ then $K_1 \equiv El(A')$ for $\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma A'{:}Type$ s.t. $[k/x]El(A') \equiv El(A)$, and, by induction hypothesis, we have $K_2 \equiv El(B')$ for $\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma A'{:}Type$ so $[k/x]K_1 \equiv [k/x]B'$.

For the third point, of the lemma, if $[k/x]K_1 \equiv El(A)$ and $[k/x]K_2 \equiv El(B)$, then $K_1 \equiv El(A')$ and $K_2 \equiv El(B')$ s.t. $[k/x]El(A') \equiv El(A)$ and $[k/x]El(B') \equiv El(B)$, hence $[k/x]A' \equiv A$ and $[k/x]B' \equiv B$. By induction hypothesis $\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma A' \leq_c B'{:}Type$ so by substitution $\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma A \leq_{[k/x]c} B{:}Type$.

- For the last two points, if the judgement follows from a subkinding for dependent product kind rule then the statement holds trivially.

□

As many of the cases for induction on the structure of derivations are straightforward or similar, in what follows, I will only present the interesting cases.

The following lemma states that, if there is a subkinding relation between two dependent kinds, then the coercion can be obtained by the subtyping for dependent product kind rule from Figure 3.3. Note that for this to hold it is essential that we only have subtyping entries in signatures and not subkinding. It might be worth noting again that subkinding is a structural lifting of subtyping relation.

Lemma 4. *If* $\Gamma \vdash_\Sigma (x{:}K_1)K_2 \leq_d (y{:}L_1)L_2$ *is derivable in* $\mathbf{T}^{0K}_{S,\leq}$ *then there exist derivable judgements in* $\mathbf{T}^{0K}_{S,\leq}$ $\Gamma \vdash_\Sigma c_1{:}(L_1)K_1$ *and* $\Gamma, y{:}L_1 \vdash_\Sigma c_2{:}([c_1(y)/x]K_2)L_2$ *such that*

- $\Gamma \vdash_\Sigma L_1 \leq_{c_1} K_1$
- $\Gamma, y{:}K_1' \vdash_\Sigma [c_1(y)/x]K_2 \leq_{c_2} L_2$ *and*
- $\Gamma \vdash_\Sigma d = [f{:}(x{:}K_1)K_2][y{:}L_1]c_2(f(c_1(y))){:}((x{:}K_1)K_2)(y{:}L_1)L_2$

are derivable in $\mathbf{T}^{0K}_{S,\leq}$.

Proof. By induction on the structure of derivation of $\Gamma \vdash_\Sigma (x{:}K_1)K_2 \leq_d (y{:}L_1)L_2$. The only non trivial case is when it comes from transitivity.

$$\frac{\Gamma \vdash_\Sigma (x{:}K_1)K_2 \leq_{d_1} C \quad \Gamma \vdash_\Sigma C \leq_{d_2} (y{:}L_1)L_2}{\Gamma \vdash_\Sigma (x{:}K_1)K_2 \leq_{d_2 \circ d_1} (y{:}L_1)L_2}$$

By the previous lemma $C \equiv (z{:}M_1)M_2$. By induction hypothesis we have that

- $\Gamma \vdash_\Sigma M_1 \leq_{c'_1} K_1$
- $\Gamma, z{:}M_1 \vdash_\Sigma [c'_1(z)/x]K_2 \leq_{c'_2} M_2$
- $\Gamma \vdash_\Sigma d_1 = [f{:}(x{:}K_1)K_2][z{:}M_1]c'_2(f(c'_1(z))){:}((x{:}K_1)K_2)(z{:}M_1)M_2$

and

- $\Gamma \vdash_\Sigma L_1 \leq_{c''_1} M_1$
- $\Gamma, y{:}L_1 \vdash_\Sigma [c''_1(y)/z]M_2 \leq_{c''_2} L_2$
- $\Gamma \vdash_\Sigma d_2 = [f{:}(z{:}M_1)M_2][y{:}L_1]c''_2(f(c''_1(y))){:}((z{:}M_1)M_2)(y{:}L_1)L_2$

are derivable.

We apply transitivity to obtain $\Gamma \vdash_\Sigma L_1 \leq_{c'_1 \circ c''_1} K_1$ and by weakening and substitution in addition, $\Gamma, y{:}L_1 \vdash_\Sigma [c'_1(c''_1(y))/x]K_2 \leq_{c''_2 \circ [c''_1(y)/z]c'_2} L_2$ and what is left to prove is that

$$\begin{aligned}\Gamma \;\; \vdash_\Sigma \;\; & d_2 \circ d_1 = [f{:}(x{:}K_1)K_2][y{:}L_1](c''_2 \circ [c''_1(y)/z]c'_2)(f((c'_1 \circ c''_1)(y))) \\ & {:}((x{:}K_1)K_2)(y{:}L_1)L_2\end{aligned}$$

Let $\Gamma \vdash_\Sigma F{:}(x{:}K_1)K_2$

$$\begin{aligned}
d_2 \circ d_1(F) &= d_2(d_1(F)) \\
&= d_2([f{:}(x{:}K_1)K_2][z{:}M_1]c_2'(f(c_1'(z)))(F)) \\
&= d_2([F/f][z{:}M_1]c_2'(f(c_1'(z)))) \\
&= d_2([z{:}M_1]c_2'(F(c_1'(z)))) \\
&= ([f{:}(z{:}M_1)M_2][y{:}L_1]c_2''(f(c_1''(y))))([z{:}M_1]c_2'(F(c_1'(z)))) \\
&= [z{:}M_1]c_2'(F(c_1'(z)))/f]([y{:}L_1]c_2''(f(c_1''(y)))) \\
&= [y{:}L_1]c_2''([z{:}M_1]c_2'(F(c_1'(z)))(c_1''(y))) \\
&= [y{:}L_1]c_2''([c_1''(y)/z]c_2'(F(c_1'(c_1''(y))))) \\
&= [y{:}L_1]c_2''([c_1''(y)/z]c_2'(F(c_1'(c_1''(y))))) \\
&= [y{:}L_1](c_2'' \circ [c_1''(y)/z]c_2')(F((c_1' \circ c_1'')(y))) \\
&= ([f{:}(x{:}K_1)K_2][y{:}L_1](c_2'' \circ [c_1''(y)/z]c_2')(f((c_1' \circ c_1'')(y))))(F)
\end{aligned}$$

$\square$

The following definition gives us a measure for the structure of kinds. I will use this measure when proving coherence for kinds. It is particularly important and I will use the fact that this measure is not increased by substitution.

Definition 6. *For $\Gamma \vdash_\Sigma K$ we define the* ***degree of*** $\Gamma \vdash_\Sigma K$ *kind $deg(K) \in \mathbb{N}$ as follows:*

1. $deg(Type) = 1$
2. $deg(El(A)) = 1$
3. $deg((x{:}K)L) = deg(K) + deg(L)$

The following lemma shows that if two kinds are related by equality or subtyping then their degree is the same.

Lemma 5. *The following hold:*

- *if $\Gamma \vdash_\Sigma K = L$ then $deg(K) = deg(L)$*

- *if* $\Gamma \vdash_\Sigma K \leq_c L$ *then* $deg(K) = deg(L)$

Proof. We do induction on the structure of derivations of $\Gamma \vdash_\Sigma K = L$ respectively $\Gamma \vdash_\Sigma K \leq L$. For example if it comes from the rule

$$\frac{\Gamma \vdash_\Sigma K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash_\Sigma K_1' = K_2'}{\Gamma \vdash_\Sigma (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

by induction hypothesis, $deg(K_1) = deg(K_2)$ and $deg(K_1') = deg(K_2')$, hence $deg((x{:}K_1)K_1') = deg((x{:}K_2)K_2')$ □

Lemma 6 (Coherence for Kinds)**.** *If* $\Gamma \vdash_\Sigma K \leq_c L$ *and* $\Gamma \vdash_\Sigma K \leq_{c'} L$ *are derivable in* $\mathbf{T}^{0K}_{S,\leq}$, *then* $\Gamma \vdash_\Sigma c = c' : (K)L$ *is derivable in* $\mathbf{T}^{0K}_{S,\leq}$.

Proof. By induction on $n = deg(K)$.

1. For $n = 1$:

   - If $\Gamma \vdash_\Sigma K = El(A)$ and $\Gamma \vdash_\Sigma L = El(B)$ then by Lemma 3 we have $\Gamma \vdash_\Sigma A \leq_c B$ and $\Gamma \vdash_\Sigma A \leq_{c'} B$ and from coherence for types $\Gamma \vdash_\Sigma c = c'{:}(A)B$, hence $\Gamma \vdash_\Sigma c = c'{:}(K)L$
   - If $\Gamma \vdash_\Sigma K = Type$ and $\Gamma \vdash_\Sigma L = Type$ then we can only have $\Gamma \vdash_\Sigma c = Id{:}(K)L$.

2. For $n > 1$, $K \equiv (x{:}K_1)K_2$ and $L \equiv (x{:}L_1)L_2$, by Lemma 4

   - $\Gamma \vdash_\Sigma L_1 \leq_{c_1} K_1$,
   - $\Gamma, x{:}K_1 \vdash_\Sigma [c_1(y)/x]K_2 \leq_{c_2} L_2$ and
   - $\Gamma \vdash_\Sigma c = [f{:}(x{:}K_1)K_2][y{:}L_1]c_2(f(c_1(y))){:}((x{:}K_1)K_2)(y{:}L_1)L_2$

   are derivable for some $\Gamma \vdash_\Sigma c_1{:}(L_1)K_1$ and $\Gamma, x{:}K_1 \vdash_\Sigma c_2{:}([c_1(x)/x]K_2)L_2$ and $deg(L_1)$, $deg(K_1)$, $deg([c_1(y)/x]K_2)$, $deg(L_2)$ are all smaller than n. If

   - $\Gamma \vdash_\Sigma L_1 \leq_{c_1'} K_1$,
   - $\Gamma, x{:}K_1 \vdash_\Sigma [c_1'(y)/x]K_2 \leq_{c_2'} L_2$ and
   - $\Gamma \vdash_\Sigma c' = [f{:}(x{:}K_1)K_2][y{:}L_1]c_2'(f(c_1'(y))){:}((x{:}K_1)K_2)(y{:}L_1)L_2$

are derivable for some other coercions $\Gamma \vdash_{\Sigma} c_1':(L_1)K_1$ and $\Gamma, x{:}K_1 \vdash_{\Sigma} c_2':([c_1'(y)/x]K_2)L_2$ then by induction hypothesis we have $\Gamma \vdash_{\Sigma} c_1 = c_1':(L_1)K_1$ and $\Gamma, x{:}K_1 \vdash_{\Sigma} c_2 = c_2':([c_1'(y)/x]K_2)L_2$ and we are done.

□

3.3.2 Weakening and Context/Signature Replacement

If we look at the rules in Figures 3.1, 3.2 and 3.3 or in the Appendix C at Figures C.1, C.3 and C.5. We observe that there are rules for weakening and signature replacement for a membership entry. I will use later in this chapter weakening and replacement for subtyping entries as well. The following lemma proves the admissibility of such a result.

Lemma 7. *The following hold:*

- *If* $\Gamma \vdash_{\Sigma,\Sigma'} J$ *and* $\vdash_{\Sigma, A\leq_c B:Type,\Sigma'} \Gamma$ *are derivable in* $T^{0K}_{S,\leq}$ *then* $\Gamma \vdash_{\Sigma, A\leq_c B:Type,\Sigma'} J$ *is derivable in* $T^{0K}_{S,\leq}$.
- *If* $\Gamma \vdash_{\Sigma, A\leq_c B:Type,\Sigma'} J$, $\vdash_{\Sigma} A = A'{:}Type$, $\vdash_{\Sigma} B = B'{:}Type$ *and* $\vdash_{\Sigma} c = c'{:}(A)B$ *are derivable in* $T^{0K}_{S,\leq}$ *then* $\Gamma \vdash_{\Sigma, A'\leq_{c'} B':Type,\Sigma'} J$ *is derivable in* $T^{0K}_{S,\leq}$.

Proof. By induction on the structure of derivation. □

Note that rules are not required for weakening and context/signature replacement to hold as they are naturally admissible even for membership entries in the system $T_{S,\leq}$. I express the admissibility result in the following lemma only for completion purpose as the rest of the chapter will be using the corresponding rules.

Lemma 8 (Weakening and Signature/Context Replacement)**.**

1. *(Weakening)*
   - *If* $\Gamma \vdash_{\Sigma,\Sigma'} J$ *and* $\vdash_{\Sigma} L$ *kind are derivable in* $T_{S,\leq}$ *and* $l \notin dom(\Gamma) \cup dom(\Sigma, \Sigma')$, *then* $\Gamma \vdash_{\Sigma, l:L,\Sigma'} J$ *is derivable in* $T_{S,\leq}$.

- *If* $\Gamma, \Gamma' \vdash_\Sigma J$ *and* $\Gamma \vdash_\Sigma K$ *kind are derivable in* $T_{S,\leq}$ *and* $x \notin dom(\Gamma, \Gamma') \cup dom(\Sigma)$, *with* $x \neq l$, *then* $\Gamma, x{:}K, \Gamma' \vdash_\Sigma J$ *is derivable in* $T_{S,\leq}$.

Note that, in the above hypothesys, if $l \notin dom(\Gamma) \cup dom(\Sigma, \Sigma')$ *then* $\vdash_{\Sigma,l:L\Sigma'} \Gamma$ *is derivable in* $T_{S,\leq}$ *and similarly if* $x \notin dom(\Gamma, \Gamma') \cup dom(\Sigma)$.

2. *(Context and Signature Replacement)*

- *If* $\Gamma \vdash_{\Sigma,m:M,\Sigma'} J$ *and* $\vdash_\Sigma M = N$ *then* $\Gamma \vdash_{\Sigma,m:N,\Sigma'} J$.
- *If* $\Gamma, x{:}K, \Gamma' \vdash_\Sigma J$, *and* $\Gamma \vdash_\Sigma K = L$ *then* $\Gamma, x{:}L, \Gamma' \vdash_\Sigma J$.

Proof. By induction on the structure of derivation. I only illustrate a case for weakening.

If $\Gamma \vdash_{\Sigma,\Sigma'} J$ is Σ, Σ' *valid*, coming from the rule

$$(*)\frac{\vdash_{\Sigma_1} L' \ \mathit{kind} \quad l' \notin dom(\Sigma_1)}{\Sigma_1, l'{:}L' \ \mathit{valid}}$$

with $\Sigma' \equiv \Sigma'', l'{:}L'$ or $\Sigma \equiv \Sigma'', l'{:}L'$ and $\Sigma' \equiv \langle\rangle$.

- If $\Sigma' \equiv \Sigma'', l'{:}L'$, for Σ, Σ'' *valid*, we have, by induction hypothesis, that $\vdash_{\Sigma,l:L,\Sigma''} L'$ *kind*, and since $l \neq l'$, if $l' \notin dom(\Sigma, \Sigma'')$ then $l' \notin dom(\Sigma, l{:}L, \Sigma'')$ and we can apply the rule $(*)$ to get $\Sigma, l{:}L, \Sigma'', l'{:}L'$ *valid.*

- If $\Sigma \equiv \Sigma'', l'{:}L'$ and $\Sigma' \equiv \langle\rangle$, then the lemma is precisely the rule $(*)$.

□

The reason I will use the corresponding rules instead of the admissibility results is because I prove the well-behavedness of this system by establishing a relation with the previous formulation from [LSX13, Xue13b]. These rules are not admissible for the system $T[\mathcal{C}]$ from Figure 2.7, in particular for the case of subtyping judgements. For example, without the weakening rule we do not have that if $\Gamma_0, \Gamma_1 \vdash A \leq_c B \in \mathcal{C}$ then $\Gamma_0, x{:}K, \Gamma_1 \vdash A \leq_c B$. The necessity of such properties has first been studied by Soloviev and Luo [SL02] and Yong Luo [Luo05] where the author denotes the set $\mathcal{C}$ which satisfies such properties as a *well-defined set of coercions (WDC)*. A WDC is a set which is coherent, and respects congruence, transitivity, substitution and weakening. The latter four properties have been adopted as rules in [LSX13, Xue13b].

3.4 Subtyping in signatures as a well-behaved extension

In this section I prove that the system $T_{S,\leq}$ is a well-behaved extension of T_S in that it is conservative - if a judgement is syntactically in T_S and it is derivable in $T_{S,\leq}$, then it is derivable in T_S too.

Another way in which the system introduced in this chapter is well-behaved is that, if a judgement can be derived in $T_{S,\leq}$, then a *corresponding* judgement can be derived in T_S. This section will present what it means for a derivable judgement J in T_S to *correspond* to a derivable judgement J' in $T_{S,\leq}$, but roughly, it means that any judgement J' derivable in $T_{S,\leq}$ is *related by definitional equality* to a judgement J_0 derivable in $T^{0K}_{S,\leq}$ and for every such judgement J_0, there exists a certain set of judgements derivable in T_S.

First I formulate the relation given by definitional equality:

Definition 7. *In the system $T_{S,\leq}$*

- $\langle\rangle = \langle\rangle$, $\Sigma, c{:}K = \Sigma', c{:}K'$ *if and only if* $\Sigma = \Sigma'$ *and* $\vdash_\Sigma K = K'$ *are derivable in* $T_{S,\leq}$
- $\vdash_\Sigma \Gamma, x{:}K =\vdash_{\Sigma'} \Gamma', x{:}K'$ *if and only if* $\vdash_\Sigma \Gamma =\vdash_{\Sigma'} \Gamma'$ *and* $\Gamma \vdash_\Sigma K = K'$ *are derivable in* $T_{S,\leq}$
- $\Gamma \vdash_\Sigma K$ *kind* $= \Gamma' \vdash_{\Sigma'} K'$ *kind if and only if* $\vdash_\Sigma \Gamma =\vdash_{\Sigma'} \Gamma'$ *and* $\Gamma \vdash_\Sigma K = K'$ *are derivable in* $T_{S,\leq}$
- $\Gamma \vdash_\Sigma k{:}K = \Gamma' \vdash_{\Sigma'} k'{:}K'$ *if and only if* $\Gamma \vdash_\Sigma K$ *kind* $= \Gamma' \vdash_{\Sigma'} K'$ *kind and* $\Gamma \vdash_\Sigma k = k'{:}K$ *are derivable in* $T_{S,\leq}$

- $\Gamma \vdash_{\Sigma} k = l{:}K = \Gamma' \vdash_{\Sigma'} k' = l'{:}K'$ *if and only if* $\Gamma \vdash_{\Sigma} K\ \ kind = \Gamma' \vdash_{\Sigma'} K'\ \ kind$ *and* $\Gamma \vdash_{\Sigma} k = k'{:}K$ *and* $\Gamma \vdash_{\Sigma} l = l'{:}K$ *are derivable in* $T_{S,\leq}$

- $\Gamma \vdash_{\Sigma} A \leq_c B = \Gamma' \vdash_{\Sigma'} A' \leq_{c'} B'$ *if and only if*
  $\Gamma \vdash_{\Sigma} A{:}\mathit{Type} = \Gamma' \vdash_{\Sigma'} A'{:}\mathit{Type}$ *and* $\Gamma \vdash_{\Sigma} B{:}\mathit{Type} = \Gamma' \vdash_{\Sigma'} B'{:}\mathit{Type}$ *and* $\Gamma \vdash_{\Sigma} c{:}(A)B = \Gamma' \vdash_{\Sigma'} c'{:}(A')B'$ *are derivable in* $T_{S,\leq}$

Similarly we define this relation on judgements in $\mathbf{T}^{0K}_{S,\leq}$ *and* T_S

A similar relation was formulated in [LSX13, Xue13b, Xue13a] for the system $T[\mathcal{C}]$. Observe that there, unless $\mathcal{C}$ was coherent, one could have a situation in which $\Gamma \vdash c{:}(A)B = \Gamma \vdash c'{:}(A)B$ hold in $T[\mathcal{C}]$ because they are both declared as coercions but $\Gamma \vdash c{:}(A)B = \Gamma \vdash c'{:}(A)B$ might not hold in T if $\mathcal{C}$ is not coherent. However this cannot happen for the system in this thesis because, if $\Gamma \vdash_{\Sigma} c{:}(A)B = \Gamma \vdash_{\Sigma} c'{:}(A)B$ and they are both coercions, it cannot be that Σ is a signature in T_S unless they are both the identity coercions, in which case they are trivially equal in T_S.

In what follows, unless otherwise specified when using the definitional equality between judgements, I will refer to the relation defined on judgements in $T_{S,\leq}$.

Lemma 9. *The above relation is an equivalence relation on judgements derivable in* $T_{S,\leq}$*.*

Proof. By induction on the structure of derivation of the judgement at the lefthand side of the equal. I just exemplify for the case $\vdash_{\Sigma} \Gamma, x{:}K \ = \vdash_{\Sigma'} \Gamma', x{:}K'$, the other cases being similar.

By induction hypothesis, $\vdash_{\Sigma'} \Gamma' = \vdash_{\Sigma} \Gamma$.

By context and signature replacement for each entry in Σ and Γ, if
$\Gamma \vdash_{\Sigma} K = K'$, then $\Gamma' \vdash_{\Sigma'} K = K'$ □

This kind of well-behavedness was first studied in [LSX13, Xue13b, Xue13a] where the authors generalise Kleene's [Kle52] formulation of extension by definition from first order logic, essentially naming *definitional extension* of a system T an extension of it T' which

1. is conservative: any judgement in T derivable in T' is derivable in T and
2. every valid derivation tree D' in T' can be translated into a valid derivation tree D in T such that the conclusion of D' is definitionally equal to the conclusion of D

This expresses that the extension is nothing more than an abbreviation for certain judgements in the base system. A further discussion about this follows in Section 5.1.

For the conservativity condition, the authors of [LSX13, Xue13b] prove that $T[\mathcal{C}]$ is equivalent to a classically conservative extension $T[\mathcal{C}]^*$ of $T[\mathcal{C}]_{0K}$, where coercive applications of the form $\Gamma \vdash f * k_0{:}[c(k_0)/x]K'$ are derivable whenever $\Gamma \vdash f{:}(x{:}K)K'$, $\Gamma \vdash K_0 \leq K$ and $\Gamma \vdash k_0{:}K_0$ are derivable in $T[\mathcal{C}]^*$. Further, $T[\mathcal{C}]_{0K}$ is a conservative extension of T as it only derives subtyping judgements which are not syntactically in T. They also prove that $T[\mathcal{C}]$ respects the second condition as an extension of $T[\mathcal{C}]_{0K}$ whenever $\mathcal{C}$ is coherent in $T[\mathcal{C}]_{0K}$. Essentially, they define a translation which transforms valid derivation trees in $T[\mathcal{C}]$ in valid derivation trees in $T[\mathcal{C}]_{0K}$ by inserting the coercions wherever coercive application rules are used. $T[\mathcal{C}]_{0K}$ is further a definitional extension of T because subtyping judgements can only be used to infer other subtyping judgements in the restricted system. So any non-subtyping judgement derivable in $T[\mathcal{C}]_{0K}$ is definitionally equal to a judgement in T, and for subtyping judgements the second condition of coherence from the definition in [LSX13, Xue13b] guarantees that there exists a corresponding non-subtyping judgements representing the well-typedness of the underlying mapping.

For the system I introduced here, it is not true that any non-subtyping judgement derivable in $T_{S,\leq}^{0K}$ is definitionally equal to a judgement in T_S, because we have subtyping entries in signatures. However I will prove that there exists a corresponding set of derivable judgements. For a non subtyping judgement $\Gamma \vdash_\Sigma J$, this set will include the judgement $\Gamma \vdash_{\Sigma^*} J$, where Σ^* is just Σ without the subtyping entries. If J is of the form $A \leq_c B$, then the set will contain simply the judgement for the underlying coercion $\Gamma \vdash_{\Sigma^*} c{:}(A)B$. Apart from this, in both cases the set will contain corresponding judgements for all the subtyping entries of Σ. For example, for the signature $\Sigma_1, A \leq_c B{:}\mathit{Type}, \Sigma_2$ valid in $T^{0K}_{S,\leq}$, the judgement corresponding to the subtyping entry $A \leq_c B{:}\mathit{Type}$ will be $\vdash_{\Sigma_1} c{:}(A)B$.

I have mentioned often the system $T[\mathcal{C}]$ in analogy or in contrast with the system I introduced in this chapter. It is indeed a very strong relation between these two systems, or to be more precise, between $\mathbf{T}_{S,\leq}$ and an adjustment of $T[\mathcal{C}]$ which I will denote by $T[\mathcal{C}]^{;}$. I will use this relation to prove that $\mathbf{T}_{S,\leq}$ is also well-behaved in a similar sense.

3.4.1 Conservativity of $T_{S,\leq}$ as an extension of T_S

The system $T[\mathcal{C}]$ from [LSX13, Xue13b] is equivalent to a conservative extension of T. $T_{S,\leq}$ is a classically conservative extension of T_S and this follows directly from the fact that $T_{S,\leq}$ keeps track of subtyping entries in the signatures and it carries them along in derivations. More precisely we prove that if a judgement is derivable in $T_{S,\leq}$ and not in T_S then it cannot be written in T_S.

The following two lemmas state that any subtyping or subkinding judgement can only be derived with a signature containing subtyping entries, and hence the signature cannot be written in T_S

Lemma 10. *If $\Gamma \vdash_\Sigma A \leq_c B{:}\mathit{Type}$ is derivable in $T_{S,\leq}$, then Σ contains at least a subtyping entry or $\Gamma \vdash_\Sigma A = B{:}\mathit{Type}$ and $\Gamma \vdash_\Sigma c = \mathit{Id}{:}(A)A$ are derivable in $T_{S,\leq}$.*

Proof. By induction on the structure of derivation. For example if it comes from transitivity from premises $\Gamma \vdash_\Sigma A \leq_c A' : \mathit{Type}$ and $\Gamma \vdash_\Sigma A' \leq_{c'} B : \mathit{Type}$ then the statement simply is true by induction hypothesis.

If it comes from a dependent product rule with premises
$\Gamma \vdash_\Sigma A_2 \leq_{c_1} A_1{:}\mathit{Type}$ and $\Gamma, x{:}A_1 \vdash_\Sigma B_1(x) \leq_{c2[x]} B_2(x)$, with
$\Gamma \vdash_\Sigma \Pi(A_1, B_1) \leq_c \Pi(A_2, B_2){:}\mathit{Type} \equiv \Gamma \vdash_\Sigma A \leq_c B{:}\mathit{Type}$ where

$c \equiv [F : \Pi(A_1, B_1)]\lambda(A_2, B_2 \circ c_1, [x{:}A_2]c_2[x](app(A_1, B_1, F, c_1(x))))$. By induction hypothesis, if at least one of the c_1 or $c_2[x]$ is not the identity, Σ contains at least one subtyping entry and we are done. If both are the identity, then c is also the identity. □

Lemma 11. *If $\Gamma \vdash_\Sigma K \leq_c L$ is derivable, then Σ contains at least a subtyping entry or $\Gamma \vdash_\Sigma K = L$ and $\Gamma \vdash_\Sigma c = Id{:}(K)L$ are derivable in $T_{S,\leq}$.*

Proof. By induction on the structure of derivation.

If it comes from the rule

$$\frac{\Gamma \vdash_\Sigma A \leq_c B{:}\mathit{Type}}{\Gamma \vdash_\Sigma \mathit{El}(A) \leq_c \mathit{El}(B)}$$

then it follows from $\Gamma \vdash_\Sigma A \leq_c B{:}\mathit{Type}$ by the previous lemma.

For all the other cases, the proof mirrors the proof of the previous lemma. For example if it comes from transitivity from premises $\Gamma \vdash_\Sigma K \leq_c M$ and $\Gamma \vdash_\Sigma M \leq_{c'} L$ then the statement simply is true by induction hypothesis and similarly for dependent product kind. □

The following lemma extends the statement to express the fact that it is enough for a judgement to contain a non trivial subtyping or subkinding entry (not the identity coercion) in its derivation tree to have a signature that cannot be written in T_S. This is important because it allows us to conclude that the extension cannot derive judgements syntactically in T_S which are not already derivable in T_S.s

Lemma 12. *If D is a derivation tree with the conclusion $\Gamma \vdash_\Sigma J$, valid in $T_{S,\leq}$ and $\Gamma_1 \vdash_{\Sigma_1} K_1 \leq_{c_0} K_2$ is present in D then, either Σ contains at least a subtyping entry or $\Gamma_1 \vdash_{\Sigma_1} K_1 = K_2$ and $\Gamma_1 \vdash_{\Sigma_1} c_0 = Id_{K_1}{:}(K_1)K_1$ are derivable in $T_{S,\leq}$.*

Proof. If $\Gamma \vdash_\Sigma J$ is a subtyping or subkinding judgement it follows directly from the previous lemmas 10, 4. Otherwise we do induction on the depth of D.

We have two possibilities for the last rule in D.

If it ends with a rule in $T^{0K}_{S,\leq}$, for instance

$$\frac{\Gamma \vdash_{\Sigma} K\,kind \quad \Gamma, x{:}K \vdash_{\Sigma} K'\,kind}{\Gamma \vdash_{\Sigma} (x{:}K)K' \;\; kind}$$

then the subkinding judgements must be in at least one of the subderivations concluding $\Gamma \vdash_{\Sigma} K \;\; kind$ and $\Gamma, x{:}K \vdash_{\Sigma} K' \;\; kind.$ The statement then holds by induction hypothesis. The other rules are similar.

The second possibility is that D ends with a coercive application or coercive definition rule. Let us assume, for instance, that the judgement $\Gamma \vdash_{\Sigma} J$ is $\Gamma \vdash_{\Sigma} f(k_0){:}[c(k_0)/x]K'$ and a derivation tree for it ends with a coercive application rule with premises $\Gamma \vdash_{\Sigma} f{:}(x{:}K)K'$, $\Gamma \vdash_{\Sigma} k_0{:}K_0$ and $\Gamma \vdash_{\Sigma} K_0 \leq_c K$, then by the previous lemma either c is trivial or Σ contains at least a subtyping entry. If Σ contains at least one subtyping entry then we are done. If c is trivial then by induction hypothesis either the derivation trees for $\Gamma \vdash_{\Sigma} f{:}(x{:}K)K'$, $\Gamma \vdash_{\Sigma} k_0{:}K_0$ and $\Gamma \vdash_{\Sigma} K_0 \leq_c K$ contain no judgement of non trivial subtyping or subkinding or Σ contains at least a subtyping entry. □

The following lemma states that, if a judgements is derived in $T_{S,\leq}$ using only trivial coercions then it can be derived in T_S.

Lemma 13. *If in a derivation tree of a judgement derivable in $T_{S,\leq}$ which is not subtyping or subkinding judgement all of the subtyping and subkinding judgements are of the form $\Gamma_1 \vdash_{\Sigma_1} A \leq_{Id_A} A{:}Type$ respectively
$\Gamma_1 \vdash_{\Sigma_1} K \leq_{[x:K]x} K$ then the judgement is derivable in T_S.*

Proof. By induction on the structure of derivations. If the derivation tree D that only contains trivial coercions ends with one of the rules of T_S,

$$\frac{\frac{D_1}{J_1}\cdots\frac{D_n}{J_n}}{J}(R)$$

then J_1,..., J_n also have derivation trees D_1,...,D_n which only contain at most trivial coercions, hence, by induction hypothesis, they are derivable in T_S. We can apply to them, with D_1,...,D_n replaced by their derivation in T_S the same rule R to obtain the judgement J and the derivation is in T_S.

Otherwise, if for example the derivation containing only trivial coercions ends with coercive application

$$\frac{\Gamma \vdash_\Sigma f{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k_0{:}K \quad \Gamma \vdash_\Sigma K \leq_{[x:K]x} K}{\Gamma \vdash_\Sigma f(k_0){:}[[x{:}K]x(k_0)/x]K'}$$

$\Gamma \vdash_\Sigma [[x{:}K]x(k_0)/x]K' = [k_0/x]K'$ and $\Gamma \vdash_\Sigma f{:}(x{:}K)K'$ and $\Gamma \vdash_\Sigma k_0{:}K$ are derivable in T_S by induction hypothesis, and from them it follows directly by functional application, in T_S, $\Gamma \vdash_\Sigma f(k_0){:}[k_0/x]K'$ □

Finally, the following corollary states the conservativity of $T_{S,\leq}$ as an extension of T_S, namely that it does not derive judgements syntactically written in T_S which can not already be derived in T_S. This happens because, for a judgement to be derived using subtyping judgements it needs to keep track of them in its own syntax, specifically in its signature. Again, this is unlike in the system $T[\mathcal{C}]^{;}$, which does not carry references to the subtyping judgements it uses in the syntax and hence is just equivalent to a classically conservative extension.

Corollary 1 (Conservativity)**.** *If a judgement is derivable in $T_{S,\leq}$ but not in T_S, its signature will contain subtyping entries, and hence it cannot be written in T_S.*

Proof. From the previous lemma, a judgement can only be derivable in $T_{S,\leq}$ but not in T_S when it contains in all of its derivation trees non trivial subtyping or subkinding judgements. If the judgements is itself a subtyping or subkinding judgement then it vacuously cannot be written in T_S. Otherwise, by Lemma 12 it follows that either all of the subtyping and subkinding judgements are of the form $\Gamma_1 \vdash_{\Sigma_1} A \leq_{Id_A} A{:}Type$ respectively $\Gamma_1 \vdash_{\Sigma_1} K \leq_{[x:K]x} K$ in which case the judgement is derivable in T_S or its signature contains subtyping entries, in which case it cannot be written in T_S. □

3.4.2 The relation between $T_{S,\leq}^{0K}$ and T_S

Here I show that, if a judgement J is derivable in $T_{S,\leq}^{0K}$, we obtain a set of judgements, one of which is of same nature as J up to erasing the subtyping entries from a signature. The idea here is that, for any the valid signature in $T_{S,\leq}^{0K}$ and all the judgements using it, we can remove the subtyping entries from it to obtain a valid signature in T_S and corresponding judgements using this signature.

Definition 8. *We define $erase(\cdot)$, a map which simply removes subtyping entries from signature as follows:*

- $erase(\langle\rangle) = \langle\rangle$
- $erase(\Sigma, c{:}K) = erase(\Sigma), c{:}K$
- $erase(\Sigma, A \leq_c B) = erase(\Sigma)$

Lemma 14. *For $\Sigma \equiv \Sigma_0, A_0 \leq_{c_0} B_0, \Sigma_1, ..., A_{n-1} \leq_{c_{n-1}} B_{n-1}\Sigma_n$ a valid signature in $T_{S,\leq}^{0K}$ we will consider the following set of judgements $(\star)(\Sigma) = \{\vdash_{erase(\Sigma_0,...,\Sigma_i)} c_i{:}(A_i)B_i\}$, where $i \in 0, ..., n$. Then the following statements hold:*

1. *$\vdash_\Sigma \Gamma$ is derivable in $T_{S,\leq}^{0K}$ if and only if $\vdash_{erase(\Sigma)} \Gamma$ and all the judgements in $(\star)$ are derivable in T_S.*
2. *$\Gamma \vdash_\Sigma J$ is not a subtyping judgement and is derivable in $T_{S,\leq}^{0K}$ if and only if $\Gamma \vdash_{erase(\Sigma)} J$ and all the judgements in $(\star)$ are derivable in T_S.*
3. *If $\Gamma \vdash_\Sigma A \leq_c B$ is derivable in $T_{S,\leq}^{0K}$ then $\Gamma \vdash_{erase(\Sigma)} c{:}(A)B$ and all the judgements in $(\star)$ are derivable in T_S.*
4. *If $\Gamma \vdash_\Sigma K \leq_c L$ is derivable in $T_{S,\leq}^{0K}$ then $\Gamma \vdash_{erase(\Sigma)} c{:}(K)L$ and all the judgements in $(\star)$ are derivable in T_S.*

Proof. The only if implication for the first two cases goes by induction on the structure of derivations. Since subtyping judgements do not contribute to deriving any other type of judgement in $T^{0K}_{S,\leq}$, the proof is relatively straightforward. It is worth stressing that, if $\Sigma \equiv \langle\rangle$, the whole judgement needs to be in T_S and $(\star)(\Sigma)$. I am mentioning this as I will consider the cases for $\Sigma \equiv \langle\rangle$ trivial in this proof and will not mention them.

For the first point, if $\Gamma \equiv \langle\rangle$, then $\vdash_\Sigma \langle\rangle$ can only follow from the judgement $\Sigma\ \ valid$. We have two subcases. If $\Sigma \equiv \Sigma', k{:}K$, then it can only follow from the premise $\vdash_{\Sigma'} K\ \ kind$ with $k \notin Dom(\Sigma')$. By induction hypothesis we know that $\vdash_{erase(\Sigma')} K\ \ kind$ is derivable in T_S and all the judgements in $(\star)(\Sigma')$ are derivable. If $k \notin Dom(\Sigma')$ then $k \notin Dom(erase(\Sigma'))$. So we can derive $erase(\Sigma'), k{:}K\ \ valid$. But $erase(\Sigma'), k{:}K \equiv erase(\Sigma', k{:}K)$ and
$\Sigma', k{:}K \equiv \Sigma$, so we have that $erase(\Sigma)\ \ valid$ is derivable in T_S, hence
$\vdash_{erase(\Sigma)} \langle\rangle$ is derivable in T_S. Likewise, by how we defined $(\star)(\cdot)$,
$(\star)(\Sigma') = (\star)(\Sigma)$.

The second case is when $\Sigma \equiv \Sigma', A \leq_c B{:}Type$. Now the judgement for the validity of Σ can only come from the premise $\vdash_{\Sigma'} c{:}(A)B$. By induction hypothesis $\vdash_{erase(\Sigma')} c{:}(A)B$ and all the judgements in $(\star)(\Sigma')$ are derivable in T_S. But $erase(\Sigma') \equiv erase(\Sigma)$ and $(\star)(\Sigma) = (\star)(\Sigma') \cup \{\vdash_{erase(\Sigma')} c{:}(A)B\}$ which we already know are all derivable and we can derive $\vdash_{erase(\Sigma')} \langle\rangle$.

If $\Gamma \equiv \Gamma', x{:}K$ then, the proof is straightforward as, by induction hypothesis, $\Gamma \vdash_\Sigma K\ \ kind$ is derivable in $T^{0K}_{S,\leq}$ only if $\Gamma \vdash_{erase(\Sigma)} K\ \ kind$ is derivable in T_S and we are done.

The second case is also straight forward.

For the if implication, Lemma 7 is used.

For the fourth point let us assume the judgement comes from a derivation tree ending with the rule

$$\frac{\vdash_{\Sigma_0, A \leq_c B:Type, \Sigma_1} \Gamma}{\Gamma \vdash_{\Sigma_0, A \leq_c B:Type, \Sigma_1} A \leq_c B{:}Type}$$

Again, as above we have the case when $\Gamma \equiv \langle\rangle$ and the case when
$\Gamma \equiv \Gamma_0, x{:}K$. If we consider the first one, its validity judgement can only come from the premise $\Sigma_0, A \leq_c B{:}Type, \Sigma_1\ \ valid$. Now we have again the

subcases $\Sigma_1 \equiv \langle\rangle$ $\Sigma_1 \equiv \Sigma', cK$ and $\Sigma_1 \equiv \Sigma', A' \leq_{c'} B'$. For the first subcase, $\Sigma_0, A \leq_c B{:}\mathit{Type}\ \mathit{valid}$ can only be derived from $\vdash_{\Sigma_0} c{:}(A)B$. By induction hypothesis, $\vdash_{erase(\Sigma_0)} c{:}(A)B$ and all the judgements in $(\star)(\Sigma_0)$ are derivable in T_S. But $erase(\Sigma_0) \equiv erase(\Sigma)$ and
$(\star)(\Sigma) = (\star)(\Sigma_0) \cup \{\vdash_{erase(\Sigma_0)} c{:}(A)B\}$ so we are done. For the second subcase, we have by induction hypothesis that $\vdash_{erase(\Sigma_0, A \leq_c B:\mathit{Type}, \Sigma')} K\ \mathit{kind}$ and all the judgements in $(\star)(\Sigma_0, A \leq_c B{:}\mathit{Type}, \Sigma')$ are derivable in T_S. We have that $\vdash_{erase(\Sigma_0)} c{:}(A)B \in (\star)(\Sigma_0, A \leq_c B{:}\mathit{Type}, \Sigma')$ so the judgement is derivable. By weakening, the judgement $\vdash_{erase(\Sigma_0, \Sigma', c:K)} c{:}(A)B$ is also derivable in T_S. Note that we can apply weakening because if $\vdash_{erase(\Sigma_0, A \leq_c B:\mathit{Type}, \Sigma')} K\ \mathit{kind}$ is derivable in T_S, and $k \notin dom(erase(\Sigma_0, A \leq_c B{:}\mathit{Type}, \Sigma'))$ then we can derive $erase(\Sigma_0, A \leq_c B{:}\mathit{Type}, \Sigma'), k{:}K\ \mathit{valid}$. The other cases and the next point are similar. $\square$

3.4.3 $T[\mathcal{C}]^{;}$

The system $T[\mathcal{C}]$ from [LSX13, Xue13b] was proven to be a well behaved extension of T in that it is equivalent to a classically conservative extension and any valid derivation tree of $T[\mathcal{C}]$ can be translated into a derivation tree of T such that their conclusion are definitionally equal in $T[\mathcal{C}]$. The translation essentially transforms all the coercive application and coercive definition rules of the derivation tree in $T[\mathcal{C}]$ into ordinary functional application and equality judgements such as reflexivity. The result is that all the coercions are inserted into the judgement obtained as conclusion.

Here I consider a system $T[\mathcal{C}]^{;}$ similar to the system $T[\mathcal{C}]$ as presented in [LSX13, Xue13b]. The difference is that here we fix some prefixes of a context, not allowing substitution and abstraction for these prefixes. In more details, the judgements of $T[\mathcal{C}]^{;}$ will be of the form $\Sigma; \Gamma \vdash J$ instead of $\Gamma \vdash J$, where Σ and Γ are just contexts and substitution and abstraction can be applied to entries in Γ but not Σ. We call this system $T[\mathcal{C}]^{;}$. To delimitate these prefixes we use the symbol ";" and the judgement forms will be as follows:

- $\vdash \Sigma;\Gamma$
- $\Sigma;\Gamma \vdash K \ \ kind$
- $\Sigma;\Gamma \vdash k{:}K$
- $\Sigma;\Gamma \vdash K = K'$
- $\Sigma;\Gamma \vdash k = k'{:}K$

where the first judgement says that $\Sigma;\Gamma$ is a valid context. The rules of the system $T[\mathcal{C}]^{;}$ are the ones in Figures 3.5,3.6, 3.7 and 3.8. These rules are also listed in the Appendix The difference between these rules and those described in [LSX13, Xue13b] is that there is now an additional set of rules for the prefixes of contexts, apart from substitution and abstraction rules which are only available for regular contexts. More detailed, I duplicate contexts, assumptions, weakening, context replacement. For example, the assumption rule

$$\frac{\vdash \Gamma, x{:}K, \Gamma'}{\Gamma, x{:}K, \Gamma' \vdash x{:}K}$$

now becomes the set of two rules

$$\frac{\vdash \Sigma, c{:}K, \Sigma'; \Gamma}{\Sigma, c{:}K, \Sigma'; \Gamma \vdash c{:}K} \quad \frac{\vdash \Sigma; \Gamma, x{:}K, \Gamma'}{\Sigma; \Gamma, x{:}K, \Gamma' \vdash x{:}K}$$

For all other rules I adjust them to the new forms of judgements by replacing $\Gamma \vdash J$ with $\Sigma;\Gamma \vdash J$. Notice that I do not duplicate substitution as only the context at the righthand side of the ; supports substitution. I will consider the system $T[\mathcal{C}]^{;}_{0K}$ to be the one without coercive application and definition rules, namely it only contains the rules in Figures 3.5,3.6 and 3.7. $\mathcal{C}$ is formed of subtyping judgements and we have the following rule in $T[\mathcal{C}]^{;}_{0K}$

$$\frac{\Gamma \vdash A \leq_c B \in \mathcal{C}}{\Gamma \vdash A \leq_c B}$$

For the system $T[\mathcal{C}]$ coercive application is added as an abbreviation to ordinary functional application and this is ensured by coercive definition together with *coherence* of $\mathcal{C}$. Indeed, it was proved in [LSX13, Xue13b] that, when $\mathcal{C}$ is coherent, $T[\mathcal{C}]$ is a well behaved extension of $T[\mathcal{C}]_{0K}$ in that every valid derivation tree D in $T[\mathcal{C}]$ can be translated into a valid derivation tree D' in $T[\mathcal{C}]_{0K}$ and the conclusion of D is definitionally equal to the conclusion of D' in $T[\mathcal{C}]$. I want to avoid doing the complex proof in [LSX13, Xue13b] again and assume that the properties of $T[\mathcal{C}]$ carry over to $T[\mathcal{C}]^{;}$. So next I give the definition of coherence for the set $\mathcal{C}$.

Definition 9. *The set $\mathcal{C}$ of subtyping judgements is coherent if the following two conditions hold in $T[\mathcal{C}]^{;}_{0K}$:*

- *If $\Sigma;\Gamma \vdash A \leq_c B$ is derivable, then $\Sigma;\Gamma \vdash c{:}(A)B$ is derivable.*
- *If $\Sigma;\Gamma \vdash A \leq_c B$ and $\Sigma;\Gamma \vdash A \leq_{c'} B$ are derivable, then $\Sigma;\Gamma \vdash c = c'{:}(A)B$ is derivable.*

Note that in the original formulation $\Sigma;\Gamma \vdash A <_{[x:A]x} A$ was not allowed. The fact that there is no coercion between a type and itself which I mentioned when I described the system $T[\mathcal{C}]$ from Subsection 2.3.2 of the previous chapter was used to prove the consistency of the system in [LSX13] and [Xue13b] essentially to prove that a coercive application judgement cannot come from a functional application of the same function. However with the current condition one can prove that, if this is the case, the coercion has to be equal to the identity.

I will also refer to a relation given by definitional equality for the system $T[\mathcal{C}]^{;}$:

Definition 10.

- *$\langle\rangle = \langle\rangle$, $\Sigma, c{:}K = \Sigma', c{:}K'$ if and only if $\Sigma = \Sigma'$ and $\Sigma;\langle\rangle \vdash K = K'$.*
- *$\vdash \Sigma;\Gamma, x{:}K =\vdash \Sigma';\Gamma', x{:}K'$ if and only if $\vdash \Sigma;\Gamma =\vdash \Sigma';\Gamma'$ and $\Sigma;\Gamma \vdash K = K'$.*

Validity of Signature/Contexts, Assumptions

$$\frac{}{\vdash \langle\rangle} \qquad \frac{\Sigma;\langle\rangle \vdash K\,kind \quad c \notin dom(\Sigma)}{\vdash \Sigma, c{:}K} \qquad \frac{\vdash \Sigma, c{:}K, \Sigma'; \Gamma}{\Sigma, c{:}K, \Sigma'; \Gamma \vdash c{:}K}$$

$$\frac{\vdash \Sigma}{\vdash \Sigma;\langle\rangle} \qquad \frac{\Sigma;\Gamma \vdash K\,kind \quad x \notin dom(\Sigma) \cup dom(\Gamma)}{\vdash \Sigma;\Gamma, x{:}K} \qquad \frac{\vdash \Sigma;\Gamma, x{:}K, \Gamma'}{\Sigma;\Gamma, x{:}K, \Gamma' \vdash x{:}K}$$

Weakening

$$\frac{\Sigma, \Sigma';\Gamma \vdash J \quad \Sigma;\langle\rangle \vdash K\ kind \quad c \notin dom(\Sigma, \Sigma')}{\Sigma,\ c{:}K,\ \Sigma';\Gamma \vdash J}$$

$$\frac{\Sigma;\Gamma, \Gamma' \vdash J \quad \Sigma;\Gamma \vdash K\ kind \quad x \notin dom(\Gamma, \Gamma')}{\Sigma;\Gamma, x{:}K, \Gamma' \vdash J}$$

Equality Rules

$$\frac{\Sigma;\Gamma \vdash K\ \ kind}{\Sigma;\Gamma \vdash K = K} \qquad \frac{\Sigma;\Gamma \vdash K = K'}{\Sigma;\Gamma \vdash K' = K} \qquad \frac{\Sigma;\Gamma \vdash K = K' \quad \Sigma;\Gamma \vdash K' = K''}{\Sigma;\Gamma \vdash K = K''}$$

$$\frac{\Sigma;\Gamma \vdash k{:}K}{\Sigma;\Gamma \vdash k = k{:}K} \qquad \frac{\Sigma;\Gamma \vdash k = k'{:}K}{\Sigma;\Gamma \vdash k' = k{:}K} \qquad \frac{\Sigma;\Gamma \vdash k = k'{:}K \quad \Sigma;\Gamma \vdash k' = k''{:}K}{\Sigma;\Gamma \vdash k = k''{:}K}$$

$$\frac{\Sigma;\Gamma \vdash k{:}K \quad \Sigma;\Gamma \vdash K = K'}{\Sigma;\Gamma \vdash k{:}K'} \qquad \frac{\Sigma;\Gamma \vdash k = k'{:}K \quad \Sigma;\Gamma \vdash K = K'}{\Sigma;\Gamma \vdash k = k'{:}K'}$$

Context Replacement

$$\frac{\Sigma_0, c{:}L, \Sigma_1;\Gamma \vdash J \quad \Sigma_0 \vdash L = L'}{\Sigma_0, c{:}L', \Sigma_1;\Gamma \vdash J} \qquad \frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash J \quad \Sigma;\Gamma_0 \vdash K = K'}{\Sigma;\Gamma_0, x{:}K', \Gamma_1 \vdash J}$$

Substitution Rules

$$\frac{\vdash \Sigma;\Gamma_0, x{:}K, \Gamma_1 \quad \Sigma;\Gamma_0 \vdash k{:}K}{\vdash \Sigma;\Gamma_0, [k/x]\Gamma_1}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash K'\ \ kind \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K'\ \ kind} \qquad \frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash L = L' \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]L = [k/x]L'}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash k'{:}K' \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]k'{:}[k/x]K'} \qquad \frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash l = l'{:}K' \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]l = [k/x]l'{:}[k/x]K'}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash K'\ \ kind \quad \Sigma;\Gamma_0 \vdash k = k'{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K' = [k'/x]K'}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash l{:}K' \quad \Sigma;\Gamma_0 \vdash k = k'{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]l = [k'/x]l{:}[k/x]K'}$$

Dependent Product Kinds

$$\frac{\Sigma;\Gamma \vdash K\,kind \quad \Sigma;\Gamma, x{:}K \vdash K'\,kind}{\Sigma;\Gamma \vdash (x{:}K)K'\,kind} \qquad \frac{\Sigma;\Gamma \vdash K_1 = K_2 \quad \Sigma;\Gamma, x{:}K_1 \vdash K_1' = K_2'}{\Sigma;\Gamma \vdash (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Sigma;\Gamma, x{:}K \vdash y{:}K'}{\Sigma;\Gamma \vdash [x{:}K]y{:}(x{:}K)K'} \qquad \frac{\Sigma;\Gamma \vdash K_1 = K_2 \quad \Sigma;\Gamma, x{:}K_1 \vdash k_1 = k_2{:}K}{\Sigma;\Gamma \vdash [x{:}K_1]k_1 = [x{:}K_2]k_2{:}(x{:}K_1)K}$$

$$\frac{\Sigma;\Gamma \vdash f{:}(x{:}K)K' \quad \Sigma;\Gamma \vdash k{:}K}{\Sigma;\Gamma \vdash f(k){:}[k/x]K'} \qquad \frac{\Sigma;\Gamma \vdash f = f'{:}(x{:}K)K' \quad \Sigma;\Gamma \vdash k_1 = k_2{:}K}{\Sigma;\Gamma \vdash f(k_1) = f'(k_2){:}[k_1/x]K'}$$

$$\frac{\Sigma;\Gamma, x{:}K \vdash k'{:}K' \quad \Sigma;\Gamma \vdash k{:}K}{\Sigma;\Gamma \vdash ([x{:}K]k')(k) = [k/x]k'{:}[k/x]K'} \qquad \frac{\Sigma;\Gamma \vdash f{:}(x{:}K)K' \quad x \notin FV(f)}{\Sigma;\Gamma \vdash [x{:}K]f(x) = f{:}(x{:}K)K'}$$

The kind Type

$$\frac{\vdash \Sigma;\Gamma}{\Sigma;\Gamma \vdash Type\ \ kind} \qquad \frac{\Sigma;\Gamma \vdash A{:}Type}{\Sigma;\Gamma \vdash El(A)\ \ kind} \qquad \frac{\Sigma;\Gamma \vdash A = B{:}Type}{\Sigma;\Gamma \vdash El(A) = El(B)}$$

Figure 3.5: Inference Rules for $LF^{;}$

Subtyping Rules

$$\frac{\Sigma;\Gamma \vdash A \leq_c B \in \mathcal{C}}{\Sigma;\Gamma \vdash A \leq_c B}$$

Congruence

$$\frac{\Sigma;\Gamma \vdash A \leq_c B : Type \quad \Sigma;\Gamma \vdash A = A' : Type \quad \Sigma;\Gamma \vdash B = B' : Type \quad \Sigma;\Gamma \vdash c = c' : (A)B}{\Sigma;\Gamma \vdash A' \leq_{c'} B' : Type}$$

Transitivity

$$\frac{\Sigma;\Gamma \vdash A \leq_c A' : Type \quad \Sigma;\Gamma \vdash A' \leq_{c'} A'' : Type}{\Sigma;\Gamma \vdash A \leq_{c' \circ c} A'' : Type}$$

Weakening

$$\frac{\Sigma, \Sigma';\Gamma \vdash A \leq_d B : Type \quad \Sigma \vdash K\ kind}{\Sigma,\ c{:}K,\ \Sigma';\Gamma \vdash A \leq_d B : Type} \quad (c \notin dom(\Sigma, \Sigma'))$$

$$\frac{\Sigma;\Gamma, \Gamma' \vdash A \leq_d B : Type \quad \Sigma;\Gamma \vdash K\ kind}{\Sigma;\Gamma, x{:}K, \Gamma' \vdash A \leq_d B : Type} \quad (x \notin dom(\Gamma, \Gamma'))$$

Context Replacement

$$\frac{\Sigma_0, c{:}L, \Sigma_1;\Gamma \vdash A \leq_c B \quad \Sigma_0 \vdash L = L'}{\Sigma_0, c{:}L', \Sigma_1;\Gamma \vdash A \leq_c B} \qquad \frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash A \leq_c B \quad \Sigma;\Gamma_0 \vdash K = K'}{\Sigma;\Gamma_0, x{:}K', \Gamma_1 \vdash A \leq_c B}$$

Substitution

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash A \leq_c B \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]A \leq_{[k/x]c} [k/x]B}$$

Identity Coercion

$$\frac{\Sigma;\Gamma \vdash A{:}Type}{\Sigma;\Gamma \vdash A \leq_{[x:A]x} A{:}Type}$$

Figure 3.6: Subtyping Rules for $T[\mathcal{C}]^{;}_{0K}$ (1)

- $\Sigma;\Gamma \vdash K = \Sigma';\Gamma' \vdash K'$ *if and only if* $\vdash \Sigma;\Gamma =\vdash \Sigma';\Gamma'$ *and* $\Sigma;\Gamma \vdash K = K'$.

- $\Gamma \vdash_\Sigma k{:}K = \Gamma' \vdash_{\Sigma'} k'{:}K'$ *if and only if* $\Sigma;\Gamma \vdash K\ \ kind = \Sigma';\Gamma' \vdash K'\ \ kind$ *and* $\Sigma;\Gamma \vdash k = k'{:}K$.

- $\Sigma;\Gamma \vdash k = l{:}K = \Sigma';\Gamma' \vdash k' = l'{:}K'$ *if and only if* $\Sigma;\Gamma \vdash K\ \ kind = \Sigma';\Gamma' \vdash K'\ \ kind$ *and* $\Sigma;\Gamma \vdash k = k'{:}K$ *and* $\Sigma;\Gamma \vdash l = l'{:}K$.

- $(\Sigma;\Gamma \vdash K = L) = (\Sigma';\Gamma' \vdash K' = L')$ *if and only if* $\Sigma;\Gamma \vdash K\ \ kind = \Sigma';\Gamma' \vdash K'\ \ kind$ *and* $\Sigma;\Gamma \vdash L\ \ kind = \Sigma';\Gamma' \vdash L'\ \ kind$

- $\Sigma;\Gamma \vdash A \leq_c B = \Sigma';\Gamma' \vdash A' \leq_{c'} B'$ *if and only if*

Basic Subkinding Rule and Identity

$$\frac{\Sigma;\Gamma\vdash A\le_c B{:}Type}{\Sigma;\Gamma\vdash El(A)\le_c El(B)}\qquad\qquad \frac{\Sigma;\Gamma\vdash K\ \ kind}{\Sigma;\Gamma\vdash K\le_{[x:K]x} K}$$

Structural Subkinding Rules

$$\frac{\Sigma;\Gamma\vdash K_1\le_c K_2\quad \Sigma;\Gamma\vdash K_1=K_1'\quad \Sigma;\Gamma\vdash K_2=K_2'\quad \Sigma;\Gamma\vdash c=c'{:}(K_1)K_2}{\Sigma;\Gamma\vdash K_1'\le_{c'} K_2'}$$

$$\frac{\Sigma;\Gamma\vdash K\le_c K'\quad \Sigma;\Gamma\vdash K'\le_{c'} K''}{\Sigma;\Gamma\vdash K\le_{c'\circ c} K''}$$

$$\frac{\Sigma,\Sigma';\Gamma\vdash K\le_d K'\quad \Sigma;\langle\rangle\vdash K_0\ kind}{\Sigma,c{:}K_0,\Sigma';\Gamma\vdash K\le_d K'}\qquad (c\notin dom(\Sigma,\Sigma'))$$

$$\frac{\Sigma;\Gamma,\Gamma'\vdash K\le_d K'\quad \Sigma;\Gamma\vdash K_0\ kind}{\Sigma;\Gamma,x{:}K_0,\Gamma'\vdash K\le_d K'}\qquad (x\notin dom(\Gamma,\Gamma'))$$

$$\frac{\Sigma_0,c{:}L,\Sigma_1;\Gamma\vdash K\le_d K'\quad \Sigma_0;\langle\rangle\vdash L=L'}{\Sigma_0,c{:}L',\Sigma_1;\Gamma\vdash K\le_d K'}$$

$$\frac{\Sigma;\Gamma_0,x{:}K,\Gamma_1\vdash L\le_d L'\quad \Sigma;\Gamma_0\vdash K=K'}{\Sigma;\Gamma_0,x{:}K',\Gamma_1\vdash L\le_d L'}$$

$$\frac{\Sigma;\Gamma_0,x{:}K,\Gamma_1\vdash K_1\le_c K_2\quad \Sigma;\Gamma_0\vdash k{:}K}{\Sigma;\Gamma_0,[k/x]\Gamma_1\vdash [k/x]K_1\le_{[k/x]c} [k/x]K_2}$$

Subkinding for Dependent Product Kind

$$\frac{\Sigma;\Gamma\vdash K_1'\le_{c_1} K_1\ \Sigma;\Gamma,x{:}K_1\vdash K_2\ kind\ \Sigma;\Gamma,x'{:}K_1'\vdash K_2'\ kind\ \Sigma;\Gamma,x{:}K_1\vdash [c_1(x')/x]K_2\le_{c_2} K_2'}{\Sigma;\Gamma\vdash (x{:}K_1)K_2\le_{[f:(x:K_1)K_2][x':K_1']c_2(f(c_1(x')))} (x{:}K_1')K_2'}$$

Figure 3.7: Subtyping Rules for $T[\mathcal{C}]^{;}_{0K}$ (2)

Coercive Application

$$(CA_1)\qquad \frac{\Sigma;\Gamma\vdash f{:}(x{:}K)K'\quad \Sigma;\Gamma\vdash k_0{:}K_0\quad \Sigma;\Gamma\vdash K_0\le_c K}{\Sigma;\Gamma\vdash f(k_0){:}[c(k_0)/x]K'}$$

$$(CA_2)\qquad \frac{\Sigma;\Gamma\vdash f=f'{:}(x{:}K)K'\quad \Sigma;\Gamma\vdash k_0=k_0'{:}K_0\quad \Sigma;\Gamma\vdash K_0\le_c K}{\Sigma;\Gamma\vdash f(k_0)=f'(k_0'){:}[c(k_0)/x]K'}$$

Coercive Definition

$$(CD)\qquad \frac{\Sigma;\Gamma\vdash f{:}(x{:}K)K'\quad \Sigma;\Gamma\vdash k_0{:}K_0\quad \Sigma;\Gamma\vdash K_0\le_c K}{\Sigma;\Gamma\vdash f(k_0)=f(c(k_0)){:}[c(k_0)/x]K'}$$

Figure 3.8: The coercive application and definition rules in $T[\mathcal{C}]^{;}$

$\Sigma;\Gamma\vdash A{:}Type = \Sigma';\Gamma'\vdash A'{:}Type$ *and* $\Sigma;\Gamma\vdash B{:}Type = \Sigma';\Gamma'\vdash B'{:}Type$ *and* $\Sigma;\Gamma\vdash c{:}(A)B = \Sigma';\Gamma'\vdash c'{:}(A')B'$.

Similar relations can be defined on subsystems $T[\mathcal{C}]^{;}_{0K}$ and $T^{;}$. Note however that now there can be a case like $\Sigma;\Gamma\vdash c{:}(A)B = \Sigma;\Gamma\vdash c'{:}(A)B$ in $T[\mathcal{C}]^{;}$ and the judgements $\Sigma;\Gamma\vdash c{:}(A)B$ and $\Sigma;\Gamma\vdash c'{:}(A)B$ derivable in $T^{;}$ but such that $\Sigma;\Gamma\vdash c{:}(A)B = \Sigma;\Gamma\vdash c'{:}(A)B$ is not the case in $T^{;}$ for $\mathcal{C}$ not coherent because the coercions now are tracked separately in a set.

As the relation from the Definition 7, this is also an equivalence relation on judgements derivable in $T[\mathcal{C}]^{;}$.

Observe that this system, unlike $T[\mathcal{C}]$ from [LSX13, Xue13b], has fixed prefixes of contexts, which do not allow abstraction or substitution. By using $T[\mathcal{C}]^{;}$, we place our argument close enough to $T_{S,\leq}$ as the prefixes of contexts before ; act just like signatures. The difference between $T[\mathcal{C}]^{;}$ and $T_{S,\leq}$ will be in how the subtyping judgements are introduced in the system and with this regards, the system $T[\mathcal{C}]^{;}$ is like $T[\mathcal{C}]$ for which we have the well behavedness result from [LSX13, Xue13b].

3.4.4 The relation between $T_{S,\leq}$ and $T[\mathcal{C}]^{;}$

There are important differences between the new $T_{S,\leq}$ and $T[\mathcal{C}]^{;}$ following from the fact that I introduce coercive subtyping through signature. More precisely coercions are now local to the signatures that introduce them. This enables us to have more coercions between two types under the same kinding assumptions (of the form c:K, x:K) and still have coherence satisfied, whereas by enriching a system with a set of coercive subtyping, our coercions are introduced globally and only one coercion (up to definitional equality) can exist between two types under the same kinding assumptions. I illustrate this with the following example. Let us consider the two signature $\Sigma_0, A \leq_c B{:}Type, \Sigma_1$ and $\Sigma_0, A \leq_d B{:}Type, \Sigma_1$ with Σ_0 and Σ_1 free of subtyping entries. In $T^{0K}_{S,\leq}$, both these signatures are coherent provided they are valid, however, if we introduced coercive judgements through a set $\mathcal{C}$ which contains $\Sigma_0; \Sigma_1 \vdash A \leq_c B{:}Type$ and $\Sigma_0; \Sigma_1 \vdash A \leq_d B{:}Type$, then this set would not be coherent in $T[\mathcal{C}]^{;}_{0K}$.

Another very important difference is that $T_{S,\leq}$ here is classically conservative extension of T_S as we saw earlier in Subsection 3.4.1 which cannot be proven for system $T[\mathcal{C}]^{;}$ as an extension of $T^{;}$ since this one does not carry the coercive entries in the syntax. Yet another important difference is that, in $T_{S,\leq}$, weakening and context and signature replacement are admissible as we saw in Subsection 3.3.2. This is not true for the system $T[\mathcal{C}]^{;}$ without additional properties of $\mathcal{C}$ as if, for example, $\Sigma;\Gamma_0,\Gamma_1 \vdash A \leq_c B{:}Type$ is a judgement in $\mathcal{C}$ and hence derivable in $T[\mathcal{C}]^{;}$, we could not derive, without the weakening rule and without adding the requirement of closure under weakening, that $\Sigma;\Gamma_0,x{:}K,\Gamma_1 \vdash A \leq_c B{:}Type$ is derivable.

However, because signatures are technically just prefix of contexts for which abstraction and substitution are not available (in [HHP93] it is proven that the derivability of the judgements in the system with signatures is equivalent to the derivability of the corresponding judgements obtained from moving the signatures before contexts in the system with contexts only), we naturally expect that there is a relation between $T_{S,\leq}$ and $T[\mathcal{C}]^{;}$. And indeed here we shall show that for any valid signature Σ in $T_{S,\leq}$, we can represent a class of judgements of $T_{S,\leq}$ depending on Σ as judgements in a $T[\mathcal{C}_\Sigma]^{;}$, where $\mathcal{C}_\Sigma$ is defined below in Definition 11.

In what follows some proofs are more interesting if dependent types are introduced to the system and the more interesting case is the one that derives that type. Because I would like to present these cases, let us consider $T^{0K}_{S,\leq}$ with the dependent product type rules with the subtyping rule

$$\frac{\Gamma \vdash_\Sigma A : Type \quad \Gamma, x{:}A \vdash_\Sigma B(x) : Type}{\Gamma \vdash_\Sigma \Pi(A,B) : Type}$$

$$\frac{\Gamma \vdash_\Sigma A : Type \quad \Gamma \vdash_\Sigma B : (A)Type \quad \Gamma \vdash_\Sigma f : (x{:}A)B(x)}{\Gamma \vdash_\Sigma \lambda(A,B,f) : \Pi(A,B)}$$

$$\frac{\Gamma \vdash_\Sigma g : \Pi(A,B) \quad \Gamma \vdash_\Sigma a : A}{\Gamma \vdash_\Sigma app(A,B,g,a) : B(a)}$$

$$\frac{\begin{array}{c}\Gamma \vdash_\Sigma A : Type \quad \Gamma \vdash_\Sigma B : (A)Type \\ \Gamma \vdash_\Sigma f : (x{:}A)B(x) \quad \Gamma \vdash_\Sigma a : A\end{array}}{\Gamma \vdash_\Sigma app(A,B,\lambda(A,B,f),a) = f(a) : B(a)}$$

Figure 3.9: Inference Rules for Π-type

$$\frac{\Gamma \vdash_\Sigma A' \leq_{c_1} A : Type \quad \Gamma \vdash_\Sigma B, B' : (A)Type \quad \Gamma, x{:}A \vdash_\Sigma B(x) \leq_{c_2[x]} B'(x) : Type}{\Gamma \vdash_\Sigma \Pi(A,B) \leq_d \Pi(A', B' \circ c_1) : Type}$$

where $d \equiv [F : \Pi(A,B)]\lambda(A', B' \circ c_1, [x{:}A']c_2[x](app(A,B,F,c_1(x))))$

This system will be introduced in more detail in Subsection 4.1.2 of the next chapter.

First I consider just $T^{0K}_{S,\leq}$ and $T[\mathcal{C}]^{;}_{0K}$ which are the systems without coercive application and coercive definition and we define a way to transfer coercive subtyping entries of a signature Σ in $T^{0K}_{S,\leq}$ to a set of coercive subtyping judgements of $T[\mathcal{C}_\Sigma]^{;}_{0K}$. For this I give the following definition which extracts from a signature Σ of $T^{0K}_{S,\leq}$ the sequence of nonsubtyping entries which we shall later see that is a valid context prefixes in $T[\mathcal{C}]^{;}_{0K}$ provided that Σ. For any valid signature Σ, I also define a way to extract a set formed by subtyping judgements corresponding to the subtyping entries of Σ. The set will be denoted as $\mathcal{C}_\Sigma$ and the subtyping judgements contained in it will be syntactically in $T[\mathcal{C}_\Sigma]^{;}_{0K}$

Definition 11. *Let Σ be a signature (not necessarily valid) in $T^{0K}_{S,\leq}$. We define Γ_Σ as follows:*

- $\Gamma_{\langle\rangle} = \langle\rangle$
- $\Gamma_{\Sigma_0,k:K} = \Gamma_{\Sigma_0}, k{:}K$
- $\Gamma_{\Sigma_0,A\leq_c B:\mathit{Type}} = \Gamma_{\Sigma_0}$

If Σ is valid in $T^{0K}_{S,\leq}$ we define $\mathcal{C}_\Sigma$ as follows:

- $\mathcal{C}_{\langle\rangle} = \emptyset$
- $\mathcal{C}_{\Sigma_0,k:K} = \mathcal{C}_{\Sigma_0}$
- $\mathcal{C}_{\Sigma_0,A\leq_c B:\mathit{Type}} = \mathcal{C}_{\Sigma_0} \cup \{\Gamma_{\Sigma_0};\langle\rangle \vdash A \leq_c B{:}\mathit{Type}\}$

Lemma 15. *Let $\Sigma \equiv \Sigma_0, A \leq_c B{:}\mathit{Type}, \Sigma_1$. If Σ valid is derivable in $T^{0K}_{S,\leq}$, then $\Gamma_\Sigma \equiv \Gamma_{\Sigma_0,\Sigma_1}$ and $\mathcal{C}_\Sigma = \mathcal{C}_{\Sigma_0,\Sigma_1} \cup \{\Gamma_{\Sigma_0};\langle\rangle \vdash A \leq_c B{:}\mathit{Type}\}$.*

Proof. By induction on the length of Σ_1. If $\Sigma_1 \equiv \langle\rangle$, then, by definition, $\Gamma_\Sigma \equiv \Gamma_{\Sigma_0}$ and $\mathcal{C}_\Sigma = \mathcal{C}_{\Sigma_0} \cup \{\Gamma_{\Sigma_0};\langle\rangle \vdash A \leq_c B{:}\mathit{Type}\}$. Otherwise, we have two cases.

If $\Sigma_1 \equiv \Sigma_1', A' \leq_{c'} B'{:}\mathit{Type}$, then, by definition, $\Gamma_\Sigma \equiv \Gamma_{\Sigma_0,A\leq_c B:\mathit{Type},\Sigma_1'}$ and $\mathcal{C}_\Sigma = \mathcal{C}_{\Sigma_0,A\leq_c B:\mathit{Type},\Sigma_1'} \cup \{\Gamma_{\Sigma_0,A\leq_c B:\mathit{Type},\Sigma_1'};\langle\rangle \vdash A' \leq_{c'} B'{:}\mathit{Type}\}$. But by induction, the statement holds for Σ_1' so $\Gamma_{\Sigma_0,A\leq_c B:\mathit{Type},\Sigma_1'} \equiv \Gamma_{\Sigma_0,\Sigma_1'} \equiv \Gamma_{\Sigma_0,\Sigma_1}$

and

$\mathcal{C}_{\Sigma_0,A\leq_c B:\,Type,\Sigma_1'} \cup \{\Gamma_{\Sigma_0,A\leq_c B:\,Type,\Sigma_1'};\langle\rangle \vdash A' \leq_{c'} B'{:}\,Type\} =$
$= \mathcal{C}_{\Sigma_0,\Sigma_1'} \cup \{\Gamma_{\Sigma_0};\langle\rangle \vdash A \leq_c B{:}\,Type\} \cup \{\Gamma_{\Sigma_0,A\leq_c B:\,Type,\Sigma_1'};\langle\rangle \vdash A' \leq_{c'} B'{:}\,Type\} =$
$= \mathcal{C}_{\Sigma_0,\Sigma_1} \cup \{\Gamma_{\Sigma_0};\langle\rangle \vdash A \leq_c B{:}\,Type\}.$

If $\Sigma_1 \equiv \Sigma_1', k{:}K$ then, by definition $\Gamma_\Sigma \equiv \Gamma_{\Sigma_0,A\leq_c B:\,Type,\Sigma_1'}, k{:}K$. But by induction hypothesis, $\Gamma_{\Sigma_0,A\leq_c B:\,Type,\Sigma_1'} \equiv \Gamma_{\Sigma_0,\Sigma_1'}$. By definition, $\Gamma_{\Sigma_0,\Sigma_1'}, k{:}K \equiv \Gamma_{\Sigma_0,\Sigma_1',k:K}$. For $\mathcal{C}_\Sigma = \mathcal{C}_{\Sigma_0,A\leq_c B:\,Type,\Sigma_1'}$, again, by induction $\mathcal{C}_{\Sigma_0,A\leq_c B:\,Type,\Sigma_1'} = \mathcal{C}_{\Sigma_0,\Sigma_1'} \cup \{\Gamma_{\Sigma_0};\langle\rangle \vdash A \leq_c B{:}\,Type\} =$
$= \mathcal{C}_{\Sigma_0,\Sigma_1',k:K} \cup \{\Gamma_{\Sigma_0};\langle\rangle \vdash A \leq_c B{:}\,Type\}$ □

The results that follow will analyse what happens when we interleave sequences in valid signature. The reason why I do so is because, for a signature Σ valid in $T^{0K}_{S,\leq}$, I consider the system $T[\mathcal{C}_\Sigma]^{;}_{0K}$. In this system, I can always obtain new judgements, by weakening or context replacement in Γ_Σ and it's prefixes and I am studying precisely the consequence of this. Note that Γ_Σ and its prefixes cannot be altered by substitution, unlike in [LSX13, Xue13b], as it is not allowed for the part of the context before ;. Note further that abstraction is not allowed either, so the entries of Γ_Σ are not variables, the only difference between Γ_Σ is that it does not contain subtyping entries, but instead it it participates in the subtyping judgements belonging to $\mathcal{C}_\Sigma$.

Lemma 16. *Let Σ_1, Σ_3 and $\Sigma_1, \Sigma_2, \Sigma_3$ be valid signatures in $T^{0K}_{S,\leq}$. If J is derivable in $T[\mathcal{C}_{\Sigma_2,\Sigma_3}]^{;}_{0K}$ then J is derivable in $T[\mathcal{C}_{\Sigma_1,\Sigma_2,\Sigma_3}]^{;}_{0K}$.*

Proof. By induction on the structure of derivation of J. □

According to Luo et al. [LSX13, Xue13b], if we add coercive subtyping and coercive definition rules from Figure 3.8 to a system enriched with a coherent set of subtyping judgements $\mathcal{C}_\Sigma$, any derivation tree in $T[\mathcal{C}_\Sigma]^{;}$ can be translated to a derivation tree in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ (that is a derivation tree that does not use coercive application and definition rules - CA_1, CA_2 and CD) and their conclusions are definitionally equal in $T[\mathcal{C}_\Sigma]^{;}$. I aim to use that result to prove that for any judgement using a coherent signature in $T_{S,\leq}$, there exists a judgement definitionally equal to it in $T^{0K}_{S,\leq}$. For this I shall first prove that $\mathcal{C}_\Sigma$ is coherent in the sense of the Definition 9 if Σ is coherent in the sense of the Definition 5. To prove this we need to describe the possible contexts at the lefthand side of ; in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ used to infer coercive subtyping judgements.

I first prove a theorem used throughout the section which allows us to argue about judgements in $T^{0K}_{S,\leq}$ and judgements in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ interchangeably. I start by presenting a lemma representing the base case and then the theorem appears as an extension proven by induction. The lemma is not required to prove the theorem but it gives a better intuition. The theorem essentially states that, for contexts at the lefthand side of ; obtained by interleaving membership entries in the image through $\Gamma.$ of a valid signature Σ or its prefixes give judgements in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ corresponding to judgements in $T^{0K}_{S,\leq}$. We will see later that all the contexts at the lefthand side of ; in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ are in fact obtained by interleaving membership entries in prefixes of Σ.

Lemma 17. *Let $\Sigma \equiv \Sigma_1, \Sigma_2, \Sigma_3$ be a valid signature in $T^{0K}_{S,\leq}$ then, for any c, K and Σ'_1, Σ'_2 such that $\Sigma_1 = \Sigma'_1$ and $\Sigma_1, \Sigma_2 = \Sigma'_1, \Sigma'_2$ the following hold:*

- *$\vdash \Gamma_{\Sigma'_1}, c{:}K, \Gamma_{\Sigma'_2}; \langle\rangle$ is derivable in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ if and only if $\Sigma'_1, c{:}K, \Sigma'_2$ valid is derivable in $T^{0K}_{S,\leq}$*
- *$\vdash \Gamma_{\Sigma'_1}, c{:}K, \Gamma_{\Sigma'_2}; \Gamma$ is derivable in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ if and only if $\vdash_{\Sigma'_1, c:K, \Sigma'_2} \Gamma$ is derivable in $T^{0K}_{S,\leq}$*
- *$\Gamma_{\Sigma'_1}, c{:}K, \Gamma_{\Sigma'_2}; \Gamma \vdash J$ is derivable in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ if and only if $\Gamma \vdash_{\Sigma'_1, c:K, \Sigma'_2} J$ is derivable in $T^{0K}_{S,\leq}$.*

Proof. By induction on the structure of derivation. □

By repeatedly applying the previous lemma (except for the case when we weaken with the empty sequence, which is straight forward by induction on the structure of derivations) we can prove:

Theorem 1 (Equivalence for $T^{0K}_{S,\leq}$)**.** *Let $\Sigma \equiv \Sigma_1, ..., \Sigma_n$ be a valid signature in $T^{0K}_{S,\leq}$ then, for any $1 \leq k \leq n$, for any $\{\Gamma_i\}_{i \in \{0..k\}}$ sequences free of subtyping* *entries and and $\Sigma'_1, ..., \Sigma'_k$ such that $\Sigma_1, ..., \Sigma_k = \Sigma'_1, ..., \Sigma'_k$ for any $i \in \{1..k\}$ the following hold:*

- $\vdash \Gamma_0, \Gamma_{\Sigma'_1}, \Gamma_1, \Gamma_{\Sigma'_2}, \Gamma_2, ..., \Gamma_{k-1}\Gamma_{\Sigma'_k}, \Gamma_k; \langle\rangle$ *is derivable in* $T[\mathcal{C}_\Sigma]^{;}_{0K}$ *if and only if* $\Gamma_0, \Sigma'_1, \Gamma_1, \Sigma'_2, \Gamma_2, ..., \Gamma_{k-1}, \Sigma'_k, \Gamma_k$ *valid is derivable in* $T^{0K}_{S,\leq}$
- $\vdash \Gamma_0, \Gamma_{\Sigma'_1}, \Gamma_1, \Gamma_{\Sigma'_2}, \Gamma_2, ..., \Gamma_{k-1}\Gamma_{\Sigma'_k}, \Gamma_k; \Gamma$ *is derivable in* $T[\mathcal{C}_\Sigma]^{;}_{0K}$ *if and only if* $\vdash_{\Gamma_0, \Sigma'_1, \Gamma_1, \Sigma'_2, \Gamma_2, ..., \Gamma_{k-1}\Sigma'_k, \Gamma_k} \Gamma$ *is derivable in* $T^{0K}_{S,\leq}$
- $\Gamma_0, \Gamma_{\Sigma'_1}, \Gamma_1, \Gamma_{\Sigma'_2}, \Gamma_2, ..., \Gamma_{k-1}\Gamma_{\Sigma'_k}, \Gamma_k; \Gamma \vdash J$ *is derivable in* $T[\mathcal{C}_\Sigma]^{;}_{0K}$ *if and only if* $\Gamma \vdash_{\Gamma_0, \Sigma'_1, \Gamma_1, \Sigma'_2, \Gamma_2, ..., \Gamma_{k-1}\Sigma'_k, \Gamma_k} J$ *is derivable in* $T^{0K}_{S,\leq}$.

Now the aim is to prove that we do not introduce any new subtyping entries in $T^{0K}_{S,\leq}$ by weakening (up to definitional equality). Note that, for this, it is essential that the weakening rules do not add subtyping entries. More precisely, in the following lemma I prove a form of strengthening, which roughly says that by strengthening the assumptions of a subtyping judgement, we can still derive it (up to definitional equality).

Lemma 18. *Let $\Sigma \equiv \Sigma_1, \Sigma_2$ a valid signature in $T^{0K}_{S,\leq}$, for any c, K, $\Sigma'_1 = \Sigma_1$ and $\Sigma'_1, \Sigma'_2 = \Sigma_1, \Sigma_2$, if $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} A \leq_c B$ is derivable in $T^{0K}_{S,\leq}$ then there exists A', c', B' such that $\vdash_\Sigma A' \leq_{c'} B'$, $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} A = A' : Type$, $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} B = B' : Type$ and $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} c = c' : (A)B$ derivable in $T^{0K}_{S,\leq}$.*

Proof. By induction on the structure of derivation of $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} A \leq_c B$. If it comes from transitivity with the premises $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} A \leq_{c_1} C$ and $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} C \leq_{c_2} B$ then, by induction hypothesis, there exist A', C', c'_1, C'', B', c'_2 such that $\vdash_\Sigma A' \leq_{c'_1} C'$ and $\vdash_\Sigma C'' \leq_{c'_2} B'$ and
$\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} A = A' : Type$, $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} B = B' : Type$,
$\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} C = C' : Type$, $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} C = C'' : Type$,
$\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} c_1 = c'_1 : (A)C$ and $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} c_2 = c'_2 : (C)B$. By transitivity of equality we have $\Gamma \vdash_{\Sigma'_1, k:K, \Sigma'_2} C' = C'' : Type$. By Lemma 14 we have that $\Gamma \vdash_{erase(\Sigma'_1, k:K, \Sigma'_2)} C' = C'' : Type$ is derivable in T_S. Similarly, because $\vdash_\Sigma C' : Type$ and $\vdash_\Sigma C'' : Type$ we have that $\vdash_{erase(\Sigma'_1, k:K, \Sigma'_2)} C' : Type$ and $\vdash_{erase(\Sigma'_1, k:K, \Sigma'_2)} C'' : Type$ are derivable in T_S. From Strengthening Lemma from [Gog94] which

holds for T_S we have that $\vdash_{erase(\Sigma)} C' = C''{:}\,Type$ is derivable. Again, by 14 we obtain $\vdash_\Sigma C' = C''{:}\,Type$. At last, we can apply congruence and transitivity $\vdash_\Sigma A' \leq_{c'_2 \circ c'_1} B'$.

Let us now consider the dependent product rule

$$\frac{\Gamma \vdash_{\Sigma'_1,k:K,\Sigma'_2} A'' \leq_{c_1} A' \qquad \begin{array}{c}\Gamma \vdash_{\Sigma'_1,k:K,\Sigma'_2} B', B'' : (A')\,Type \\ \Gamma, x{:}A' \vdash_{\Sigma'_1,k:K,\Sigma'_2} B'(x) \leq_{c_2[x]} B''(x)\end{array}}{\Gamma \vdash_{\Sigma'_1,k:K,\Sigma'_2} \Pi(A',B') \leq_c \Pi(A'', B'' \circ c_1)}$$

with $A \equiv \Pi(A',B')$, $B \equiv \Pi(A'', B'' \circ c_1)$ and
$c \equiv [F : \Pi(A',B')]\lambda(A'', B'' \circ c_1, [x{:}A'']c_2[x](app(A',B',F,c_1(x))))$.

By induction hypothesis, there exist A''_0, A'_0, c'_1, B'_0, B''_0, c'_2 such that $\vdash_\Sigma A''_0 \leq_{c'_1} A'_0$, $\vdash_\Sigma B' \leq_{c'_2} B''$ and $\Gamma \vdash_{\Sigma'_1,k:K,\Sigma'_2} A'' = A''_0{:}\,Type$, $\Gamma \vdash_{\Sigma'_1,k:K,\Sigma'_2} A' = A'_0{:}\,Type$, $\Gamma, x{:}A' \vdash_{\Sigma'_1,k:K,\Sigma'_2} B''(x) = B''_0{:}\,Type$, $\Gamma, x{:}A' \vdash_{\Sigma'_1,k:K,\Sigma'_2} B'(x) = B'_0{:}\,Type$, $\Gamma \vdash_{\Sigma'_1,k:K,\Sigma'_2} c_1 = c'_1{:}(A'')A'$ and $\Gamma, x{:}A' \vdash_{\Sigma'_1,k:K,\Sigma'_2} c_2(x) = c'_2{:}(B'(x))B''(x){:}\,Type$ are derivable. We apply dependent product rule for the case when types are constants and obtain $\vdash A'_0 \longrightarrow B'_0 \leq'_c A''_0 \longrightarrow B''_0$ with $c' \equiv [F : A'_0 \longrightarrow B'_0][x{:}A''_0](c''_2(F(c'_1(x))))$. By normal equality rules for dependent product and its terms we have that $\Gamma \vdash_{\Sigma'_1,k:K,\Sigma'_2} A'_0 \longrightarrow B'_0 = \Pi(A',B')$, $\Gamma \vdash_{\Sigma'_1,k:K,\Sigma'_2} A''_0 \longrightarrow B''_0 = \Pi(A'',B'')$ and $\Gamma \vdash_{\Sigma'_1,k:K,\Sigma'_2} c = c'{:}(\Pi(A',B'))\Pi(A'',B'')$ $\square$

By repeatedly applying the previous lemma we obtain

Corollary 2. *For Σ valid derivable in $T^{0K}_{S,\leq}$, for any partition $\Sigma \equiv \Sigma_1, ..., \Sigma_n$, for any*
$\{\Gamma_i\}_{i\in\{0..n\}}$ sequences free of subtyping entries and $\{\Sigma'_i\}_{i\in\{1..n\}}$ such that $\Sigma_1, ..., \Sigma_i = \Sigma'_1, ..., \Sigma'_i$ for any $i \in \{1..n\}$, if
$\Gamma \vdash_{\Gamma_0,\Sigma_1,\Gamma_1,\Sigma_2,\Gamma_2,...,\Gamma_{n-1}\Sigma_n,\Gamma_n} A \leq_c B$ is derivable in $T^{0K}_{S,\leq}$ then there exists A', c', B' such that $\vdash_\Sigma A' \leq_{c'} B'$, $\Gamma \vdash_{\Gamma_0,\Sigma_1,\Gamma_1,\Sigma_2,\Gamma_2,...,\Gamma_{n-1}\Sigma_n,\Gamma_n} A = A'{:}\,Type$, $\Gamma \vdash_{\Gamma_0,\Sigma_1,\Gamma_1,\Sigma_2,\Gamma_2,...,\Gamma_{n-1}\Sigma_n,\Gamma_n} B = B'{:}\,Type$ and
$\Gamma \vdash_{\Gamma_0,\Sigma_1,\Gamma_1,\Sigma_2,\Gamma_2,...,\Gamma_{n-1}\Sigma_n,\Gamma_n} c = c'{:}(A)B$ derivable in $T^{0K}_{S,\leq}$.

Next I prove that weakening does not break coherence:

Lemma 19. *For* Σ *valid in* $T^{0K}_{S,\leq}$*, if* $\Sigma \equiv \Sigma_1, \Sigma_2, \Sigma_3$ *is coherent, for any* Σ'_1, Σ'_2 *such that* $\Sigma_1 = \Sigma'_1$ *and* $\Sigma_1, \Sigma_2 = \Sigma'_1, \Sigma'_2$*, for any* c, K *such that* $\Sigma'_1, c{:}K, \Sigma'_2$ *is valid,* $\Sigma'_1, c{:}K, \Sigma'_2$ *coherent.*

Proof. Let us consider the derivable judgements $\Gamma \vdash_{\Sigma'_1,c:K,,\Sigma'_2} A \leq_c B$ and $\Gamma \vdash_{\Sigma'_1,c:K,,\Sigma'_2} A \leq_d B$. Then we know from Corollary 2 that there exist A', B', A'', B'', c', d' such that $\vdash_{\Sigma_1,\Sigma_2} A' \leq_{c'} B'$, $\vdash_{\Sigma_1,\Sigma_2} A'' \leq_{d'} B''$,
$\Gamma \vdash_{\Sigma'_1,c:K,,\Sigma'_2} A' = A{:}\mathit{Type}$, $\Gamma \vdash_{\Sigma'_1,c:K,,\Sigma'_2} B'' = B{:}\mathit{Type}$,
$\Gamma \vdash_{\Sigma'_1,c:K,,\Sigma'_2 s} B'' = B{:}\mathit{Type}$ $\Gamma \vdash_{\Sigma'_1,c:K,,\Sigma'_2} c = c'{:}(A)B$ and
$\Gamma \vdash_{\Sigma'_1,c:K,,\Sigma'_2} d = d'{:}(A)B$ are derivable in derivable in $T^{0K}_{S,\leq}$. As in the proof of the previous lemma, using Lemma 14 and Strengthening Lemma from [Gog94] we have that $\vdash_{\Sigma_1,\Sigma_2} A' = A''{:}\mathit{Type}$, $\vdash_{\Sigma_1,\Sigma_2} B' = B''{:}\mathit{Type}$. By congruence we have that $\vdash_{\Sigma_1,\Sigma_2} A' \leq_{d'} B'$ is derivable in $T^{0K}_{S,\leq}$. If Σ is coherent then any prefix of it $\Sigma_1, ..., \Sigma_k$ is coherent so $\vdash_{\Sigma_1,\Sigma_2} c' = d'{:}(A')B'$. Further, by weakening and Lemma 7, we have the desired result. □

By repeatedly applying the previous lemma we obtain:

Lemma 20. *For* Σ *valid in* $T^{0K}_{S,\leq}$*, if* $\Sigma \equiv \Sigma_1, ..., \Sigma_n$ *is coherent, for any* $1 \leq k \leq n$*, for any* $\{\Gamma_i\}_{i\in\{0..k\}}$ *sequences free of subtyping entries, for any* $\{\Sigma'_i\}_{i\in\{0..k\}}$ *such that* $\Sigma_1, ..., \Sigma_i = \Sigma'_1, ..., \Sigma'_i$ *for any* $i \in \{1..k\}$ *such that* $\Gamma_0, \Sigma_1, \Gamma_1, \Sigma_2, \Gamma_2, ..., \Gamma_{k-1}, \Sigma_k, \Gamma_k$ *is valid, then* $\Gamma_0, \Sigma_1, \Gamma_1, \Sigma_2, \Gamma_2, ..., \Gamma_{k-1}, \Sigma_k, \Gamma_k$ *is coherent.*

Finally, the following lemma describes the relation between parts of the context at the lefthand side of the ; of judgements in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ and Σ. This is a very important result for proving the coherence of $\mathcal{C}_\Sigma$ based on the coherence of Σ. It states that any such context is in fact obtained from weakening of a prefix of Σ. In addition from this Lemma, because all the derivable judgements in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ that are not in $T^{;}$ are subtyping judgements, we have as a consequence that all the judgements of $T[\mathcal{C}_\Sigma]^{;}_{0K}$ are equivalent to judgements in $T^{0K}_{S,\leq}$.

Lemma 21. *For* Σ *a valid signature in* $T^{0K}_{S,\leq}$*, for any derivable judgement* $\Gamma'; \Gamma \vdash J$ *in* $T[\mathcal{C}_\Sigma]^{;}_{0K}$ *there exists a partition of* $\Sigma \equiv \Sigma_1, ..., \Sigma_n$*,* $1 \leq k \leq n$*,*

$\Gamma_0, ..., \Gamma_k$ *free of subtyping entries and* $\Sigma'_1, ..., \Sigma'_k$ *with* $\Sigma'_1, ..., \Sigma'_i = \Sigma'_1, ..., \Sigma'_i$ *for any* $1 \leq i \leq k$ *such that* $\Gamma' \equiv \Gamma_{\Gamma_0, \Sigma_1, \Gamma_1, ..., \Sigma_k, \Gamma_k}$

Proof. By induction on the structure of derivation of the judgement in $T[\mathcal{C}_\Sigma]^{;}_{0K}$. I only prove a case for third point when the judgement is $\Gamma'; \Gamma \vdash A \leq_c B$. The only nontrivial case is when the judgements follows from weakening. Let us assume it comes from a derivation tree ending with

$$\frac{\Gamma'_1, \Gamma'_2; \Gamma \vdash A \leq_c B \quad \Gamma'_1; \langle\rangle \vdash K \ \ kind}{\Gamma'_1, c{:}K, \Gamma'_2; \Gamma \vdash A \leq_c B}$$

with $\Gamma' \equiv \Gamma_1, c{:}K, \Gamma_2$.

By induction hypothesis we know that there exists a partition $\Sigma \equiv \Sigma_1, ..., \Sigma_n$ and $1 \leq k \leq n$ and $\Gamma_0, ..., \Gamma_k$ and $\Sigma'_1, ..., \Sigma'_k$ with $\Sigma'_1, ..., \Sigma'_i = \Sigma'_1, ..., \Sigma'_i$ for any $1 \leq i \leq k$ such that $\Gamma'_1, \Gamma'_2 \equiv \Gamma_{\Gamma_0, \Sigma'_1, \Gamma_1, ..., \Sigma'_k, \Gamma_k}$ with $\Gamma \vdash_{\Gamma_0, \Sigma'_1, \Gamma_1 ..., \Sigma'_k, \Gamma_k} A \leq_c B$. Let us consider the case when $\Gamma'_1 \equiv \Gamma_{\Gamma_0, \Sigma'_1, \Gamma_1, ..., \Gamma_{i-1}, \Sigma^{1\prime}_i}$ and $\Gamma'_2 \equiv \Gamma_{\Sigma^{2\prime}_i, \Gamma_i, ..., \Sigma'_k, \Gamma_k}$. With $\Sigma'_i \equiv \Sigma^{1\prime}_i, \Sigma^{2\prime}_i$ for some $1 \leq i \leq k$. We consider the partition $\Sigma \equiv \Sigma_1, ..., \Sigma^1_i, \Sigma^2_i, ..., \Sigma_n$ such that $\Sigma'_1, ..., \Sigma^{1\prime}_i, \Sigma^{2\prime}_i, ..., \Sigma'_n$ $\Sigma_1, ..., \Sigma_l = \Sigma'_1, ..., \Sigma'_l$ for any $l \in 1..i-1$, $\Sigma_1, ..., \Sigma^1_i = \Sigma'_1, ..., \Sigma^{1\prime}_i$, $\Sigma_1, ..., \Sigma^1_i, \Sigma^2_i = \Sigma'_1, ..., \Sigma^{1\prime}_i, \Sigma^{2\prime}_i$ and $\Sigma_1, ..., \Sigma_l = \Sigma'_1, ..., \Sigma'_l$ for any $l \in i+1..n$ and $\Gamma_0, ..., \Gamma_{i-1}, c{:}K, \Gamma_i, ..., \Gamma_k$ such that $\Gamma' = \Gamma_{\Gamma_0, \Sigma_1, ..., \Gamma_{i-1}, \Sigma^1_i, c:K, \Sigma^2_i, \Gamma_i, ..., \Sigma_k, \Gamma_k}$. □

The next lemma refers to the ability to argue about coherence of a set of coercive subtyping judgements corresponding to a signature.

Theorem 2 (Equivalence of Coherence)**.** *Let* Σ *be a valid signature in* $T^{0K}_{S,\leq}$. *Then* Σ *is coherent in the sense of Definition 5 if and only if* $\mathcal{C}_\Sigma$ *is coherent for* $T[\mathcal{C}_\Sigma]^{;}_{0K}$ *in the sense of Definition 9.*

Proof. Only if: Let $\Gamma'; \Gamma \vdash A \leq_c B$ and $\Gamma'; \Gamma \vdash A \leq_d B$ be derivable in $T[\mathcal{C}_\Sigma]^{;}_{0K}$. From Lemma 21, it follows that there exists a partition of $\Sigma \equiv \Sigma_1, ..., \Sigma_n$ and $1 \leq k \leq n$ and $\Gamma_0, ..., \Gamma_k$ such that $\Gamma' = \Gamma_{\Gamma_0, \Sigma_1, ..., \Sigma_k, \Gamma_k}$. If Σ is coherent, then $\Gamma_0, \Sigma_1, ..., \Sigma_k, \Gamma_k$ is coherent (from Lemma 20). From Theorem 1, $\Gamma'; \Gamma \vdash A \leq_c B$ and $\Gamma'; \Gamma \vdash A \leq_d B$ are derivable in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ if and only if $\Gamma \vdash_{\Gamma_0,\Sigma_1,...,\Sigma_k,\Gamma_k} A \leq_c B$ and $\Gamma \vdash \Gamma_0, \Sigma_1, ..., \Sigma_k, \Gamma_k A \leq_d B$ are derivable in $T^{0K}_{S,\leq}$. From coherence here we have $\Gamma \vdash_{\Gamma_0,\Sigma_1,...,\Sigma_k,\Gamma_k} c = d{:}(A)B$ which is derivable in $T^{0K}_{S,\leq}$ if and only if $\Gamma'; \Gamma \vdash c = d{:}(A)B$ is derivable in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ (again by Theorem Theorem 1).

If: By Theorem 1, $\Gamma \vdash_\Sigma A \leq_c B{:}\,\mathit{Type}$ and $\Gamma \vdash_\Sigma A \leq_d B{:}\,\mathit{Type}$ are derivable in $T^{0K}_{S,\leq}$ if and only if $\Gamma_\Sigma; \Gamma \vdash A \leq_c B$ and $\Gamma_\Sigma; \Gamma \vdash A \leq_d B$ are derivable in $T[\mathcal{C}_\Sigma]^{;}_{0K}$. Because $\mathcal{C}_\Sigma$ is coherent, $\Gamma_\Sigma; \Gamma \vdash c = d{:}(A)B$ is derivable in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ which happens if and only if $\Gamma \vdash_\Sigma c = d{:}(A)B$ is derivable in $T^{0K}_{S,\leq}$ □

To prove that the system $T_{S,\leq}$ is well behaved I first prove that it is well behaved when all the signatures considered are valid in the restricted system $T^{0K}_{S,\leq}$. First I prove another equivalence lemma for this situation.

Theorem 3 (Equivalence between $T_{S,\leq}$ and $T[\mathcal{C}_\Sigma]^{;}$). *For Σ valid in $T^{0K}_{S,\leq}$, the following hold:*

- $\vdash \Gamma_\Sigma; \Gamma$ *is derivable in* $T[\mathcal{C}_\Sigma]^{;}$ *if and only if* $\vdash_\Sigma \Gamma$ *is derivable in* $T_{S,\leq}$
- $\Gamma_\Sigma; \Gamma \vdash J$ *is derivable in* $T[\mathcal{C}_\Sigma]^{;}$ *if and only if* $\Gamma \vdash_\Sigma J$ *is derivable in* $T_{S,\leq}$.

Proof. By induction on the structure of derivation. □

The following theorem shows that the system we defined here is well behaved and that every coercive subtyping application is really just an abbreviation.

Lemma 22. *If a valid signature Σ in $T^{0K}_{S,\leq}$ is coherent the following hold:*

1. *If $\vdash_\Sigma \Gamma$ is derivable in $T_{S,\leq}$ then there exists Γ' such that $\vdash_\Sigma \Gamma'$ is derivable in $T^{0K}_{S,\leq}$ and $\vdash_\Sigma \Gamma = \Gamma'$ is derivable in $T_{S,\leq}$.*
2. *If $\Gamma \vdash_\Sigma J$ is derivable in $T_{S,\leq}$ then there exists Γ', J' such that $\Gamma' \vdash_\Sigma J'$ is derivable in $T^{0K}_{S,\leq}$ and $\vdash_\Sigma \Gamma = \Gamma'$ and $\Gamma \vdash_\Sigma J = J'$ are derivable in $T_{S,\leq}$.*

Proof. By Theorem 2, since Σ is coherent in, $\mathcal{C}_\Sigma$ is coherent. If we look at the last case, by Theorem 3, $\Gamma \vdash_\Sigma J$ is derivable in $T_{S,\leq}$ if and only if $\Gamma_\Sigma; \Gamma \vdash J$ is derivable in $T[\mathcal{C}_\Sigma]^{;}$. From [LSX13, Xue13b] we know that, when $\mathcal{C}_\Sigma$ is coherent, any derivation tree of $\Gamma_\Sigma; \Gamma \vdash J$ can be translated into a derivation tree in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ which concludes with the judgement definitionally equal to $\Gamma_\Sigma; \Gamma \vdash J$. So let us consider one such derivation tree, its translation and the definitionally equal conclusion $\Gamma_\Sigma; \Delta \vdash J'$ ($\vdash \Gamma_\Sigma; \langle\rangle$ is already derivable in $T[\mathcal{C}_\Sigma]^{;}_{0K}$ so by inspecting the definition of the translation in [LSX13, Xue13b] we observe that Γ_Σ will not be changed by the translation). We have $\vdash \Gamma_\Sigma; \Gamma = \Gamma_\Sigma; \Delta$ and $\Gamma_\Sigma; \Gamma \vdash J = J'$ are derivable in $T[\mathcal{C}_\Sigma]^{;}$. From Theorem 3 we know that in this case $\vdash_\Sigma \Gamma = \Delta$ and $\Gamma \vdash_\Sigma J = J'$ are derivable in $T_{S,\leq}$ so the desired derivable judgement is simply $\Delta \vdash_\Sigma J'$. $\square$

3.4.5 The relation between $T_{S,\leq}$ and T_S

I can, at last, express the well behavedness of $T_{S,\leq}$. Note that Theorem 22 covers the well-behavedness of judgements derived under a signature that is valid in $T^{0K}_{S,\leq}$. We now prove further that any signature valid in $T_{S,\leq}$ is definitionally equal to a signature valid in $T^{0K}_{S,\leq}$, then because of signature replacement we have that any judgement derivable in in $T_{S,\leq}$ is definitionally equal to a judgement derivable in $T^{0K}_{S,\leq}$.

Lemma 23. *For any signature Σ valid in $T_{S,\leq}$ there exists Σ' valid in $T^{0K}_{S,\leq}$ such that $\Sigma = \Sigma'$ is derivable in $T_{S,\leq}$.*

Proof. By induction on the length of Σ. We assume $\Sigma = \Sigma_0, c{:}K$. By induction hypothesis we have that there exists Σ'_0 valid in $T^{0K}_{S,\leq}$ such that $\Sigma_0 = \Sigma'_0$. By repeatedly applying signature replacement to $\vdash_{\Sigma_0} K\ \ kind$ we have
$\vdash_{\Sigma'_0} K\ \ kind$ is derivable in $T_{S,\leq}$. By Lemma 22, we have that there exists K' such that $\vdash_{\Sigma'_0} K'\ \ kind$ is derivable in $T^{0K}_{S,\leq}$ with $\vdash_{\Sigma'_0} K = K'$. That means we can derive, in $T^{0K}_{S,\leq}$, $\Sigma'_0, c{:}K'\ \ valid$. Going back with context replacement we also have $\vdash_{\Sigma_0} K = K'$ derivable, so $\Sigma'_0, c{:}K'$ is the signature we are looking for. $\square$

I finish this section with the following theorem:

Theorem 4. *If a valid signature Σ in $T_{S,\leq}$ is coherent the following hold:*

1. *If $\vdash_\Sigma \Gamma$ is derivable in $T_{S,\leq}$ then there exists Σ' and Γ' such that $\vdash_{\Sigma'} \Gamma'$ is derivable in $T^{0K}_{S,\leq}$ and $\Sigma = \Sigma'$ and $\vdash_\Sigma \Gamma = \Gamma'$ are derivable in $T_{S,\leq}$.*

2. *If $\Gamma \vdash_\Sigma J$ is derivable in $T_{S,\leq}$ then there exists Σ', Γ', J' such that $\Gamma' \vdash_{\Sigma'} J'$ is derivable in $T^{0K}_{S,\leq}$ and $\Sigma = \Sigma'$, $\vdash_\Sigma \Gamma = \Gamma'$ and $\Gamma \vdash_\Sigma J = J'$ are derivable in $T_{S,\leq}$.*

Proof. According to the Lemma 23 there exist Σ' valid in $T^{0K}_{S,\leq}$ such that $\Sigma = \Sigma'$. If we consider the last point, by signature replacement $\Gamma \vdash_{\Sigma'} J$ is derivable $T_{S,\leq}$. Because Σ' valid in $T^{0K}_{S,\leq}$, we can apply the Lemma 22 to obtain $\Gamma' \vdash_{\Sigma'} J'$ such that $\vdash_{\Sigma'} \Gamma = \Gamma'$ and $\Gamma \vdash_{\Sigma'} J = J'$ are derivable in $T_{S,\leq}$. Again by signature replacement $\vdash_\Sigma \Gamma = \Gamma'$ and $\Gamma \vdash_\Sigma J = J'$. □

Further, according to the Lemma 14, the derivability of any nonsubtyping judgement in $T^{0K}_{S,\leq}$ is equivalent to the derivability of a judgement in T_S and any subtyping judgement in $T^{0K}_{S,\leq}$ implies a judgement in T_S.

To review, in this chapter I proved that, for any derivable judgement J in $T_{S,\leq}$, there exists a set of derivable judgements in T_S for which J is an abbreviation. The set of derivable judgements from T_S is formed by a corresponding judgement and the typing judgements for the underlying mappings of all coercions from signatures. I also proved that $T_{S,\leq}$ is a conservative extension of T_S in the classical sense.

Chapter 4

Case Studies: Subsumptive Subtyping, Universes, Injectivity

The aim of this chapter is to explore some practical situations and the conditions in which they can be represented by the system previously defined. First, I will consider a system with subsumptive subtyping $\Pi_{\leq}$, with subtyping entries in contexts, inspired by the system of Aspinall and Compagnoni [AC01]. To represent this system, I will use an instance of the system previously defined, more precisely that when T_S is the system specified in LF_S with Π-type. I will refer to this system as Π_S and to its extension with coercive subtyping as $\Pi_{S,\leq}$. The result in this first part is a faithful embedding of $\Pi_{\leq}$ into $\Pi_{S,\leq}$.

This is important because it establishes the connection between coercive subtyping and a form of subsumptive subtyping, more precisely I claim that coercive subtyping is a more general notion of subtyping which can, in particular, represent a form of subsumptive subtyping as opposed to giving coercive subtyping only as an alternative to subsumptive subtyping as it was previously done.

The second part of the chapter considers two forms of universes subtyping, Russell style universes with subsumptive subtyping and Tarski style universes with coercive subtyping. I will argue that the Tarski style setting with coercive subtyping entries in signatures can be used to represent the Russell style hierarchy.

I finish the chapter with a discussion on a property commonly exhibited by inclusions, namely injectivity and present a way to add it to the system with coercive subtyping entries in signatures. I will consider constructor subtyping with Leibniz equality as the practical situation that exhibits injectivity.

4.1 Embedding Subsumptive Subtyping in Coercive Subtyping

I start by presenting the system with subsumptive subtyping, then I will concretely present $T_{S,\leq}$ when $T_S \equiv \Pi_S$ and, at last, I will show how to embed $\Pi_{\leq}$ in $\Pi_{S,\leq}$.

4.1.1 $\Pi_{\leq}$

Here I consider the logical framework $\mathbb{LF}$ obtained from LF by changing $\vdash$ to $\Vdash$. It has the following forms of judgements

1. Γ *valid*,
2. $\Gamma \Vdash K\ kind$,
3. $\Gamma \Vdash k{:}K$,
4. $\Gamma \Vdash K = K'$ and
5. $\Gamma \Vdash k = k'{:}K$

and the rules from Figure 4.1

I specify the Π-type constants from Figure 4.2 as described in Subsection 2.1.2. This constants give the rules in Figure 4.3.

I add the forms of judgements $\Gamma \Vdash A \leq B{:}Type$ and $\Gamma \Vdash K \leq K'$ obtained with the rules from Figure 4.4. The rules from Figures 4.2, 4.3 and 4.4 constitute the system denoted by $\Pi_{\leq}$. This rules are also listed in the Appendix E.

Validity of Signature/Contexts, Assumptions

$$\frac{}{\Vdash \langle\rangle} \qquad \frac{\Gamma \Vdash K\,kind \quad x \notin dom(\Gamma)}{\Vdash \Gamma, x{:}K} \qquad \frac{\Vdash \Gamma, x{:}K, \Gamma'}{\Gamma, x{:}K, \Gamma' \Vdash x{:}K}$$

Equality Rules

$$\frac{\Gamma \Vdash K \;\; kind}{\Gamma \Vdash K = K} \qquad \frac{\Gamma \Vdash K = K'}{\Gamma \Vdash K' = K} \qquad \frac{\Gamma \Vdash K = K' \quad \Gamma \Vdash K' = K''}{\Gamma \Vdash K = K''}$$

$$\frac{\Gamma \Vdash k{:}K}{\Gamma \Vdash k = k{:}K} \qquad \frac{\Gamma \Vdash k = k'{:}K}{\Gamma \Vdash k' = k{:}K} \qquad \frac{\Gamma \Vdash k = k'{:}K \quad \Gamma \Vdash k' = k''{:}K}{\Gamma \Vdash k = k''{:}K}$$

$$\frac{\Gamma \Vdash k{:}K \quad \Gamma \Vdash K = K'}{\Gamma \Vdash k{:}K'} \qquad \frac{\Gamma \Vdash k = k'{:}K \quad \Gamma \Vdash K = K'}{\Gamma \Vdash k = k'{:}K'}$$

Substitution Rules

$$\frac{\Vdash \Gamma_0, x{:}K, \Gamma_1 \quad \Gamma_0 \Vdash k{:}K}{\Vdash \Gamma_0, [k/x]\Gamma_1}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash K' \;\; kind \quad \Gamma_0 \Vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]K' \;\; kind} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash L = L' \quad \Gamma_0 \Vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]L = [k/x]L'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash k'{:}K' \quad \Gamma_0 \Vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash l = l'{:}K' \quad \Gamma_0 \Vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]l = [k/x]l'{:}[k/x]K'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash K' \;\; kind \quad \Gamma_0 \Vdash k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]K' = [k'/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash l{:}K' \quad \Gamma_0 \Vdash k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]l = [k'/x]l{:}[k/x]K'}$$

Dependent Product Kinds

$$\frac{\Gamma \Vdash K\,kind \quad \Gamma, x{:}K \Vdash K'\,kind}{\Gamma \Vdash (x{:}K)K'\,kind} \qquad \frac{\Gamma \Vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \Vdash K_1' = K_2'}{\Gamma \Vdash (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Gamma, x{:}K \Vdash y{:}K'}{\Gamma \Vdash [x{:}K]y{:}(x{:}K)K'} \qquad \frac{\Gamma \Vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \Vdash k_1 = k_2{:}K}{\Gamma \Vdash [x{:}K_1]k_1 = [x{:}K_2]k_2{:}(x{:}K_1)K}$$

$$\frac{\Gamma \Vdash f{:}(x{:}K)K' \quad \Gamma \Vdash k{:}K}{\Gamma \Vdash f(k){:}[k/x]K'} \qquad \frac{\Gamma \Vdash f = f'{:}(x{:}K)K' \quad \Gamma \Vdash k_1 = k_2{:}K}{\Gamma \Vdash f(k_1) = f'(k_2){:}[k_1/x]K'}$$

$$\frac{\Gamma, x{:}K \Vdash k'{:}K' \quad \Gamma \Vdash k{:}K}{\Gamma \Vdash ([x{:}K]k')(k) = [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma \Vdash f{:}(x{:}K)K' \quad x \notin FV(f)}{\Gamma \Vdash [x{:}K]f(x) = f{:}(x{:}K)K'}$$

The kind Type

$$\frac{\Vdash \Gamma}{\Gamma \Vdash Type \;\; kind} \qquad \frac{\Gamma \Vdash A{:}Type}{\Gamma \Vdash El(A) \;\; kind} \qquad \frac{\Gamma \Vdash A = B{:}Type}{\Gamma \Vdash El(A) = El(B)}$$

Figure 4.1: Inference Rules for $\mathbb{LF}$

Constant declarations:

$$\begin{array}{rcl} \Pi & : & (A{:}Type)(B{:}(A)\,Type)\,Type \\ \lambda & : & (A{:}Type)(B{:}(A)\,Type)((x{:}A)B(x))\Pi(A,B) \\ app & : & (A{:}Type)(B{:}(A)\,Type)(\Pi(A,B))(x{:}A)B(x) \end{array}$$

Definitional equality rule

$$app(A,B,\lambda(A,B,f),a) = f(a) : B(a).$$

Figure 4.2: Constants for Π-types in logical framework

Besides the ordinary variables from contexts in Π, I allow contexts in $\Pi_{\leq}$ to have subtyping variables like $\alpha \leq A$.

$$\frac{\Gamma \Vdash A : \mathit{Type} \quad \Gamma, x{:}A \Vdash B(x) : \mathit{Type}}{\Gamma \Vdash \Pi(A,B) : \mathit{Type}}$$

$$\frac{\Gamma \Vdash A : \mathit{Type} \quad \Gamma \Vdash B : (A)\mathit{Type} \quad \Gamma \Vdash f : (x{:}A)B(x)}{\Gamma \Vdash \lambda(A,B,f) : \Pi(A,B)}$$

$$\frac{\Gamma \Vdash g : \Pi(A,B) \quad \Gamma \Vdash a : A}{\Gamma \Vdash \mathit{app}(A,B,g,a) : B(a)}$$

$$\frac{\begin{array}{c}\Gamma \Vdash A : \mathit{Type} \quad \Gamma \Vdash B : (A)\mathit{Type} \\ \Gamma \Vdash f : (x{:}A)B(x) \quad \Gamma \Vdash a : A\end{array}}{\Gamma \Vdash \mathit{app}(A,B,\lambda(A,B,f),a) = f(a) : B(a)}$$

Figure 4.3: Inference Rules for Π - types specified in $\mathbb{LF}$

General Subtyping Rules

$$\frac{\Gamma \Vdash K = K'}{\Gamma \Vdash K \le K'} \qquad \frac{\Gamma \Vdash K \le K' \quad \Gamma \Vdash K' \le K''}{\Gamma \Vdash K \le K''}$$

$$\frac{\Gamma \Vdash A = B{:}\mathit{Type}}{\Gamma \Vdash A \le B{:}\mathit{Type}}$$

$$\frac{\Gamma \Vdash A \le B{:}\mathit{Type} \quad \Gamma \Vdash B \le C{:}\mathit{Type}}{\Gamma \Vdash A \le C{:}\mathit{Type}}$$

Subtyping in Contexts

$$\frac{\Gamma \Vdash A{:}\mathit{Type} \quad \alpha \notin FV(\Gamma)}{\Gamma, \alpha \le A \;\; \mathit{valid}} \qquad \frac{\Gamma, \alpha \le A, \Gamma' \;\; \mathit{valid}}{\Gamma, \alpha \le A, \Gamma' \Vdash \alpha{:}\mathit{Type}} \qquad \frac{\Gamma, \alpha \le A, \Gamma' \;\; \mathit{valid}}{\Gamma, \alpha \le A, \Gamma' \Vdash \alpha \le A{:}\mathit{Type}}$$

Type Lifting and Subtyping

$$\frac{\Gamma \Vdash A \le B{:}\mathit{Type}}{\Gamma \Vdash \mathit{El}(A) \le \mathit{El}(B)} \qquad \frac{\Gamma \Vdash k{:}K \quad \Gamma \Vdash K \le K'}{\Gamma \Vdash k{:}K'} \qquad \frac{\Gamma \Vdash k = k'{:}K \quad \Gamma \Vdash K \le K'}{\Gamma \Vdash k = k'{:}K'}$$

Dependent Product

$$\frac{\begin{array}{c}\Gamma \Vdash \Pi(A,B){:}\mathit{Type} \quad \Gamma \Vdash \Pi(A',B'){:}\mathit{Type} \\ \Gamma \Vdash A' \le A{:}\mathit{Type} \quad \Gamma, x{:}A' \Vdash B \le B'{:}\mathit{Type}\end{array}}{\Gamma \Vdash \Pi(A,B) \le \Pi(A',B'){:}\mathit{Type}}$$

Figure 4.4: Subtyping Rules for $\Pi_{\le}$

$\Pi_{\le}$ is the subsumptive subtyping system specified in LF that corresponds to the system $\lambda P_{\le}$ in [AC01]. Note that there are some subtle differences between Edinburgh LF (λP) [HHP93] and the logical framework LF we use (LF has rules to derive definitional equality, including β and η rules, whereas λP leaves definitional equality and β conversion at meta-level and η rule does not hold here.), but they are irrelevant to the point of this chapter: the system with coercive subtyping in signatures can be used to faithfully represent a subsumptive subtyping system.

4.1.2 $\Pi_{S,\leq}$

Here I shall consider the system described in Section 3.2 of the previous chapter for the particular case when T_S is in fact Π_S. Π_S is the system specified in LF_S given by the rules in Figure 3.1 with the Π-type constants as in Figure 4.2 from the previous section which give the rules in Figure 4.5. A complete listing of the rules of this system can also be found in the Appendix C.

$$\frac{\Gamma \vdash_\Sigma A : \mathit{Type} \quad \Gamma, x{:}A \vdash_\Sigma B(x) : \mathit{Type}}{\Gamma \vdash_\Sigma \Pi(A, B) : \mathit{Type}}$$

$$\frac{\Gamma \vdash_\Sigma A : \mathit{Type} \quad \Gamma \vdash_\Sigma B : (A)\mathit{Type} \quad \Gamma \vdash_\Sigma f : (x{:}A)B(x)}{\Gamma \vdash_\Sigma \lambda(A, B, f) : \Pi(A, B)}$$

$$\frac{\Gamma \vdash_\Sigma g : \Pi(A, B) \quad \Gamma \vdash_\Sigma a : A}{\Gamma \vdash_\Sigma \mathit{app}(A, B, g, a) : B(a)}$$

$$\frac{\begin{array}{c}\Gamma \vdash_\Sigma A : \mathit{Type} \quad \Gamma \vdash_\Sigma B : (A)\mathit{Type} \\ \Gamma \vdash_\Sigma f : (x{:}A)B(x) \quad \Gamma \vdash_\Sigma a : A\end{array}}{\Gamma \vdash_\Sigma \mathit{app}(A, B, \lambda(A, B, f), a) = f(a) : B(a)}$$

Figure 4.5: Inference Rules for Subtyping in Π_S

In order to extend this system with subtyping, we need to add the structural subtyping rule for dependent product type to the subtyping rules from Figure 3.2. The rule is the following

$$\frac{\begin{array}{r}\Gamma \vdash_\Sigma B, B' : (A)\mathit{Type} \\ \Gamma \vdash_\Sigma A' \leq_{c_1} A : \mathit{Type} \quad \Gamma, x{:}A \vdash_\Sigma B(x) \leq_{c_2[x]} B'(x) : \mathit{Type}\end{array}}{\Gamma \vdash_\Sigma \Pi(A, B) \leq_d \Pi(A', B' \circ c_1) : \mathit{Type}} (*)$$

where $d \equiv [F : \Pi(A, B)]\lambda(A', B' \circ c_1, [x{:}A']c_2[x](\mathit{app}(A, B, F, c_1(x))))$

Let us now consider the system $\Pi^{0K}_{S,\leq}$ given by the rules in Figures 3.1, 4.5, 3.2 and 3.3 and the rule $(*)$. All these rules can also be found listed in the Appendix C. Coherence in the sense of the Definition 5 is now considered for signatures in the system $\Pi^{0K}_{S,\leq}$.

At this point, as studied by Luo and Adams [LA08], we need to make sure that we cannot produce incoherence from derivable premises. It turns out that for this particular way of adding dependent product, we don't need to do anything further. For example, if we consider the situation when we have a coercion obtained from transitivity and one from the $(*)$ rule. Let us assume we have a

$$\frac{\dfrac{\Gamma \vdash_\Sigma A'' \leq_{c_1'} A' \leq_{c_1} A : \mathit{Type}}{\Gamma \vdash_\Sigma A'' \leq_{c_1 \circ c_1'} A : \mathit{Type}} \qquad \dfrac{\Gamma \vdash_\Sigma B, B', B'' : (A)\mathit{Type} \quad \Gamma, x{:}A \vdash_\Sigma B(x) \leq_{c_2[x]} B'(x) \leq_{c_2'[x]} B''(x) : \mathit{Type}}{\Gamma, x{:}A'' \vdash_\Sigma B(x) \leq_{c_2'[x] \circ c_2[x]} B''(x) : \mathit{Type}}}{\Gamma \vdash_\Sigma \Pi(A, B) \leq_d \Pi(A'', B'' \circ (c_1 \circ c_1')) : \mathit{Type}}$$

where $d(F) = \lambda(A'', B'' \circ (c_1 \circ c_1'), [x{:}A''](c_2'[x] \circ c_2[x])(\mathit{app}(A, B, F, (c_1 \circ c_1')(x)))$ for $F : \Pi(A, B)$ and (omitting $:\mathit{Type}$)

$$\frac{\dfrac{\Gamma \vdash_\Sigma A'' \leq_{c_1'} A' \quad \Gamma, x{:}A' \vdash_\Sigma B'(x) \leq_{c_2'[x]} B''(x)}{\Gamma \vdash_\Sigma \Pi(A', B') \leq_{d_2} \Pi(A'', B'' \circ (c_1'))} \qquad \dfrac{\Gamma \vdash_\Sigma A' \leq_{c_1} A \quad \Gamma, x{:}A \vdash_\Sigma B(x) \leq_{c_2[x]} B'(x)}{\Gamma \vdash_\Sigma \Pi(A, B) \leq_{d_1} \Pi(A', B' \circ (c_1))}}{\Gamma \vdash_\Sigma \Pi(A, B) \leq_{d_2 \circ d_1} \Pi(A'', B'' \circ (c_1 \circ c_1'))}$$

where $d_1(F) = \lambda(A', B' \circ c_1, [x{:}A]c_2[x](\mathit{app}(A, B, F, c_1(x)))$ for $F : \Pi(A, B)$ and $d_2(G) = \lambda(A'', B'' \circ c_1', [x{:}A'']c_2'[x](\mathit{app}(A', B', G, c_1'(x)))$ for $G : \Pi(A', B')$.

The question is, whether, for $F : \Pi(A, B)$, $(d_2 \circ d_1)(F) = d(F)$.

$$\begin{aligned}
(d_2 \circ d_1)(F) &= d_2(d_1(F)) \\
&= d_2(\lambda(A', B' \circ c_1, [x{:}A']c_2[x](\mathit{app}(A, B, F, c_1(x)))) \\
&= \lambda(A'', B'' \circ c_1', [x{:}A'']c_2'[x](\mathit{app}(A', B', \lambda(A', B' \circ c_1, \\
&\quad [x{:}A']c_2[x](\mathit{app}(A, B, F, c_1(x))), c_1'(x))) \\
&= \lambda(A'', B'' \circ (c_1 \circ c_1'), \\
&\quad [x{:}A''](c_2'[x] \circ c_2[x])(\mathit{app}(A, B, F, (c_1 \circ c_1')(x))) \\
&= d(F)
\end{aligned}$$

Note that here I used the *app* constant instead of the eliminator discussed in Subsection 2.1.2 which is lazy and the above equality happens with η rule. It turns out that if we used the eliminator for dependent product which is strict in defining the structural coercion we would no longer be able to prove this. Let

$$\begin{aligned}
E_\Pi \;:\; & (A{:}\mathit{Type})(B{:}(A)\mathit{Type})(C{:}(\Pi(A, B))\mathit{Type})(g{:}((f : (x{:}A)B(x)) \\
& C(\lambda(A, B, f))))(z{:}\Pi(A, B))C(z)
\end{aligned}$$

and

$$\begin{aligned}
d(F) \quad = \quad & E_\Pi(A, B, [f{:}\Pi(A, B)]\Pi(A'', B''), [g{:}(x{:}A)B(x)]\lambda(A'', B'' \circ (c_1 \circ c_1'), \\
& [x{:}A'']c_2'[x](c_2[x]g((c_1 \circ c_1')(x)))), F)
\end{aligned}$$

then we have

$$\begin{aligned}
(d_2 \circ d_1)(F) \quad = \quad & d_2(d_1(F)) \\
= \quad & d_2(E_\Pi(A, B, [f_1{:}\Pi(A, B)]\Pi(A', B'), \\
& [g{:}(x{:}A)B(x)]\lambda(A', B' \circ (c_1), [x{:}A']c_2[x]g(c_1(x))), F)) \\
= \quad & E_\Pi(A', B', [f_2{:}\Pi(A', B')]\Pi(A'', B''), \\
& [h{:}(x{:}A')B'(x)]\lambda(A'', B'' \circ (c_1'), [x{:}A'']c_2'[x]h(c_1'(x))), \\
& E_\Pi(A, B, [f_1{:}\Pi(A, B)]\Pi(A', B'), [g{:}(x{:}A)B(x)]\lambda(A', B' \circ (c_1), \\
& [x{:}A']c_2[x]g(c_1(x))), F))
\end{aligned}$$

and this cannot be reduced any further.

As studied by Luo and Adams [LA08], a solution for this is the addition of a functoriality rule. In this chapter, for simplicity I will stick to the usage of *app*. Similarly, one can add structural subyping rule for dependent sum with projections.

With coherence defined for signatures of the system $\Pi^{0K}_{S,\leq}$, I form the system $\Pi_{S,\leq}$ from the rules Figures 3.1, 4.5, 3.2, 3.3 and 3.4 and the rule $(*)$. The rules for this system can also be found listed together in the Appendix C. Note that the fact that we consider Π and the structural subtyping rule for Π - type does not impact the well-behavedness proof from the Section 3.4 of the previous chapter. It carries over simply by considering $\Pi^{;}$, the type system specified in $LF^{;}$ with Π-type, $\Pi[\mathcal{C}]^{;}$ instead of $T[\mathcal{C}]^{;}$ and it should also have structural subtyping rule for dependent product type

$$\frac{\Sigma\Gamma\vdash A'\leq_{c_1}A:\mathit{Type}\quad \Sigma;\Gamma\vdash B,B':(A)\mathit{Type}\quad \Sigma;\Gamma,x:A\vdash B(x)\leq_{c_2[x]}B'(x):\mathit{Type}}{\Sigma;\Gamma\vdash\Pi(A,B)\leq_d\Pi(A',B'\circ c_1):\mathit{Type}}(*)$$

where $d \equiv [F : \Pi(A, B)]\lambda(A', B' \circ c_1, [x{:}A']c_2[x](app(A, B, F, c_1(x))))$

4.1.3 The embedding of $\Pi_{\leq}$ in $\Pi_{S,\leq}$

Once introduced $\Pi_{\leq}$ I proceed by giving an interpretation of it in the system with coercive subtyping in signatures $\Pi_{S,\leq}$, namely I will show that this calculus can be faithfully embedded in the coercive subtyping one.

In this system, an important thing to note is how placing subtyping entries in contexts interferes with abstraction, specifically, the abstraction is not allowed at the lefthand side of subtyping entries. I will give a mapping that sends the contexts with subtyping entries in the subsumptive system to signatures in the coercive system, prove that these signatures are coherent, and, finally, that we can embed the subsumptive subtyping system into the coercive subtyping system via this mapping. I am motivated by giving a coercive subtyping system in which I can represent this subsumptive system and at the same time allowing abstraction to happen freely.

I will assume that Δ is an arbitrary context in $\Pi_{\leq}$. We can also assume without loss of generality that $\Delta \equiv \Delta_1, \alpha_1 \leq A_1, ..., \Delta_n, \alpha_n \leq A_n, \Delta_{n+1}$, where $\{\alpha_i \leq A_i\}_{i=\overline{1,n}}$ are all of the subtyping entries of Δ. If Δ_{n+1} is free of subtyping entries we can abstract over its entries freely but the abstraction is obstructed by $\alpha_n \leq A_n$ for the entire prefix. I move this prefix, together with the obstructing entry to the signature using constant coercions $\Sigma_\Delta =$ $\Delta_1, \alpha_1{:}\mathit{Type}, c_1{:}(\alpha_1)A_1, \alpha_1 \leq_{c_1} A_1{:}\mathit{Type}, ..., \Delta_n, \alpha_n{:}\mathit{Type}, c_n{:}(\alpha_n)A_n, \alpha_n \leq_{c_n} A_n{:}\mathit{Type}$. I map the left Δ_{n+1} to a context. This way, for
$\Delta \equiv \Delta_1, \alpha_1 \leq A_1, ..., \Delta_n, \alpha_n \leq A_n, \Delta_{n+1}$, the judgement $\Delta \vdash J$ from $\Pi_{\leq}$ gets translated to $\Delta_{n+1} \vdash_{\Sigma_\Delta} J$ in $\Pi_{S,\leq}$, with Σ_Δ as above. In the rest of the section we shall prove that mapping subsumptive subtyping entries in context to constant coercions in signature is indeed adequate. For this, I first prove that

such a signature is coherent.

Lemma 24. *For any valid context Δ in $\Pi_{\leq}$, Σ_Δ is coherent w.r.t. $\Pi_{S,\leq}$.*

Proof. We need to show that, in $\Pi_{S,\leq}$, if we have $\Gamma \vdash_{\Sigma_\Delta} T_1 \leq_c T_2$ and $\Gamma \vdash_{\Sigma_\Delta} T_1 \leq_{c'} T_2$, then $c = c'{:}(T_1)T_2$. There are two cases:

1. $T_1 \equiv \alpha$ is a constant. By the validity of Δ, we have that, if $\alpha_i \leq A_i$ and $\alpha_j \leq A_i$ are two different subtyping entries in Δ, then $\alpha_i \neq \alpha_j$, therefore, if $\alpha_i \leq_{c_i} A_i$ and $\alpha_j \leq_{c_j} A_i$ are two different coercions in Σ_Δ, then necessarily, $\alpha_i \neq \alpha_j$.

2. $T_1 \equiv \Pi(A, B)$ and $T_2 \equiv \Pi(A'', B'')$. In this case the non trivial situation is:
$$\frac{\Gamma \vdash_\Sigma \Pi(A,B) \leq_{c_1} C \quad \Gamma \vdash_\Sigma C \leq_{c_2} \Pi(A'',B'')}{\Gamma \vdash_\Sigma \Pi(A,B) \leq_{c_2 \circ c_1} \Pi(A'',B'')}$$
and C is equal to dependent product too. What we need to show is that applying dependent product rule followed by transitivity leads to the same coercion as applying transitivity first and then the dependent product rule. Namely that, for some A', B' such that
$$\frac{\Gamma \vdash_{\Sigma_\Delta} A'' \leq_{c_2} A' \leq_{c_1} A \quad \Gamma \vdash_{\Sigma_\Delta} B \leq_{d_1} B' \leq_{d_2} B''}{\Gamma \vdash_{\Sigma_\Delta} \Pi(A,B) \leq_{e_1} \Pi(A',B') \leq_{e_2} \Pi(A'',B'')}$$
where, for $F{:}A \longrightarrow B$ and $G{:}\Pi(A',B')$, $e_1(F) = \lambda[x'{:}A']d_1(app(F, c_1(x')))$ and $e_2(G) = \lambda[x''{:}A'']d_2(app(G, c_2(x'')))$ applying transitivity rule, first to A, A', A'' and to B, B', B'' and then to $\Pi(A,B)$, $\Pi(A',B')$, $\Pi(A'',B'')$ results in the same coercion, that is:
$$\begin{aligned} e_2 \circ e_1 &= e_2(e_1(F)) \\ &= \lambda[x''{:}A'']d_2(app(e_1(F), c_2(x''))) \\ &=_\beta \lambda[x''{:}A'']d_2(d_1(app(F, c_1(c_2(x''))))) \\ &= d_2 \circ d_1(app(F, c_1(c_2(x'')))) \end{aligned}$$

□

Notation If $\Gamma \vdash_\Sigma k{:}K$ and $\Gamma \vdash_\Sigma K \leq_c K'$ are derivable in $\Pi_{S,\leq}$, I write $\Gamma \vdash_\Sigma k :: K'$.

Theorem 5 (Embedding Subsumptive Subtyping)**.** *Let Δ and Γ be valid contexts in $\Pi_\leq$, such that Γ does not contain any subtyping entries. Then we have:*

1. *If Δ, Γ is valid in $\Pi_\leq$ then $\vdash_{\Sigma_\Delta} \Gamma$ valid in $\Pi_{S,\leq}$.*
2. *If $\Delta, \Gamma \Vdash K$ kind is derivable in $\Pi_\leq$, then $\Gamma \vdash_{\Sigma_\Delta} K$ kind is derivable in $\Pi_{S,\leq}$.*
3. *If $\Delta, \Gamma \Vdash K = K'$ is derivable in $\Pi_\leq$, then $\Gamma \vdash_{\Sigma_\Delta} K = K'$ is derivable in $\Pi_{S,\leq}$.*
4. *If $\Delta, \Gamma \Vdash k{:}K$ is derivable in $\Pi_\leq$, then $\Gamma \vdash_{\Sigma_\Delta} k{::}K$ in $\Pi_{S,\leq}$.*
5. *If $\Delta, \Gamma \Vdash k = k'{:}K$ is derivable in $\Pi_\leq$, then $\Gamma \vdash_{\Sigma_\Delta} k = k'{::}K$ in $\Pi_{S,\leq}$.*
6. *If $\Delta, \Gamma \Vdash A \leq B{:}Type$ is derivable in $\Pi_\leq$ then $\Gamma \vdash_{\Sigma_\Delta} A \leq_c B{:}Type$, for some coercion $c{:}(A)B$, is derivable in $\Pi_{S,\leq}$.*
7. *If $\Delta, \Gamma \Vdash K \leq K'$is derivable in $\Pi_\leq$, then $\Gamma \vdash_{\Sigma_\Delta} K \leq_c K'$, for some $c{:}(K)K'$, is derivable in $\Pi_{S,\leq}$.*

Proof. The proof proceeds by induction on derivations for all the points of the theorem and I only exhibit it for the sixth point here and in particular when the last rule in the derivation tree is the one for the dependent product. We have, by induction hypothesis, that, for $\Gamma \vdash_{\Sigma_\Delta} \Pi(A, B){::}\,Type$ and $\Gamma \vdash_{\Sigma_\Delta} \Pi(A', B'){::}\,Type$ we have $\Gamma \vdash_{\Sigma_\Delta} A' \leq_c A{:}\,Type$ and
$\Gamma, x{:}A' \vdash_{\Sigma_\Delta} B \leq_{c'} B'{:}\,Type$. Note that, if $K \leq_c Type$, then $K \equiv Type$, so $\Gamma \vdash_{\Sigma_\Delta} \Pi(A, B){::}\,Type$ is equivalent to $\Gamma \vdash_{\Sigma_\Delta} \Pi(A, B){:}\,Type$, and
$\Gamma \vdash_{\Sigma_\Delta} \Pi(A', B'){::}\,Type$ is equivalent to $\Gamma \vdash_{\Sigma_\Delta} \Pi(A', B'){:}\,Type$, hence we can directly apply the rule for dependent product in $\Pi_{S,\leq}$ to obtain
$\Gamma \vdash_{\Sigma_\Delta} \Pi(A, B) \leq_d \Pi(A', B'){:}\,Type$ where
$F{:}\Pi(A, B)$, $d(F) = \lambda[x{:}A']c'(app(F, c(x)))$. □

What this proves is that we can represent the previously introduced subsumptive subtyping system in the system with coercive subtyping in signatures, meaning that we can argue about the former system with the sematic richness of the latter.

4.2 Representing Russell style universes in Tarski style universes

Universes were introduced by Martin-Löf [ML98, ML75, ML82, ML84] for his intuitionistic type theory to enable the formulation of type of types motivated by the need to have reflection principle (it is closed to formation of inductive types) and at the same time avoid the paradox of having all formulas in the system provable which is caused by having a type of all types studied by Girard [Gir72] and later presented presented by Coquand [Coq86]. Two forms of universes were introduced, Russell and Tarski style. I introduced these forms of universes in Subsection 2.1.4 and I explained how Russell style is easier to use than Tarski style. I also mentioned that Russell style universes bear a subsumptive hierarchy induced by cumulativity. I gave an example of how Tarski style can be used to form these inductive types without cumulativity but at the expense of a more complicate system to work with.

In what follows I will present two forms of subtyping for the two styles of universes, I will explain why one of them can introduce issues for meta-theory and how we can use signatures to represent it in the other style.

Russell-style Universes and Subsumptive Subtyping.

Let us extend the subsumptive subtyping system $\Pi_{\leq}$ from Section 4.1 with Russell-style universes by adding the rules in Figure 4.6.

Problems with Russell style universes were observed by Luo [Luo12b]. This straightforward formulation of universes does not satisfy the properties of canonicity or subject reduction if one adopts the standard notation of terms with full type information. For instance, the term $\lambda X{:}U_1.Nat$, where $Nat : U_0$,

for $i \in \omega$

$$\frac{\Gamma\ valid}{\Gamma \Vdash U_i : Type} \qquad \frac{\Gamma \Vdash A : U_i}{\Gamma \Vdash A : Type} \qquad \frac{\Gamma\ valid}{\Gamma \Vdash U_i : U_{i+1}} \qquad \frac{\Gamma\ valid}{\Gamma \Vdash U_i \leq U_{i+1}}$$

$$\frac{\Gamma \Vdash A : U_i \quad \Gamma \Vdash B : (A)U_i}{\Gamma \Vdash \Pi(A, B) : U_i}$$

Figure 4.6: Inference Rules for Russell Style Universes with Π-type

would be represented as $\lambda(U_1, [\_{:}U_1]U_0, [\_{:}U_1]Nat)$, but this term, which is of type $U_0 \to U_0$ (by subsumption, since $U_1 \to U_0 \leq U_0 \to U_0$ by contravariance), is not definitionally equal to any canonical term which is of the form $\lambda(U_0, ...)$.

An alternative is to use proof terms with less typing information like using (a, b) instead of $pair(A, B, a, b)$ to represent pairs, as in HoTT (see Appendix 2 of [Uni13]). The problem with this approach is that not only the property of type uniqueness fails, but a proof term may have incompatible types. For example, for $a : A$ and $A : U$, where U is a type universe, the pair (A, a) has both types $U \times A$ and $\Sigma X{:}U.X$, which are incompatible in the sense that none of them is a subtype of the other. This would lead to undecidability of type checking which is unacceptable for type theories with logics based on the propositions-as-types principle. To see the problem of type checking, it may be worth pointing out that, for a dependent type theory, type checking depends on type inference that is, in a type-checking algorithm one has to infer the type of a term in many situations, however, it has not been studied what happens in presentations where, in our case pairs are only type-checked.

Tarski Style Universes with Coercive Subtyping to represent Russell Style Universes

The Tarski-style universes are introduced into $\Pi_{S,\leq}$ by adding the rules in Figure 4.7.

Further, I annotate the lifting operators t_{i+1} as coercions, as suggested in [Luo12b]. Already the example considered in Subsection 2.1.4 is simplified with coercive application. The analogous to $\Sigma(U_i, \lambda X{:}U_i.X)$ from Russell

for $i \in \omega$

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma U_i : \mathit{Type}} \qquad \frac{\Gamma \vdash_\Sigma a : U_i}{\Gamma \vdash_\Sigma T_i(a) : \mathit{Type}} \qquad \frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma t_{i+1} : (U_i)U_{i+1}}$$

where t_{i+1} are the lifting operators,

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma u_i : U_{i+1}} \qquad \frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma T_{i+1}(u_i) = U_i : \mathit{Type}}$$

where u_i is the name of U_i in U_{i+1}
With the equation: $T_{i+1}(t_{i+1}(a)) = T_i(a){:}\mathit{Type}$

$$\frac{\Gamma \vdash_\Sigma a : U_i \quad \Gamma,\ x : T_i(a) \vdash_\Sigma b(x) : U_i}{\Gamma \vdash_\Sigma \pi_i(a,b) : U_i}$$

which satisfy the following equations:
$\Gamma \vdash_\Sigma T_i(\pi_i(a,b)) = \Pi(T_i(a), [x{:}T_i(a)]T_i(b(x))) : \mathit{Type}$
$\Gamma \vdash_\Sigma t_{i+1}(\pi_i(a,b)) = \pi_{i+1}(t_{i+1}(a), [x{:}T_i(a)]t_{i+1}(b(x))) : U_{i+1}$

Figure 4.7: Inference Rules for Tarski Style Universes with Π-type

style, which we considered then, and which was $\Sigma(U_i, [x{:}T_{i+1}(u_i)]T_{i+1}(t_{i+1}(x)))$ now becomes $\Sigma(U_i, [x{:}T_{i+1}(u_i)]T_{i+1}(x))$.

At this point we can further ask that all the signatures start with the prefix $\Sigma_i \equiv U_0 \leq_{t_0} U_1,\ \ldots,\ U_{i-1} \leq_{t_i} U_i$ where i is bigger than the largest universe index that is used in an application. If universes are specified in the Tarski-style as above with the lifting operators declared as coercions, together with several notational conventions (eg, T_i is omitted, u_i is identified with U_i, etc.), they can now be used easily in Russell-style. The lifting operators are not seen (implicit) by the users. In particular, in this setting, all the Russell-style universe rules become derivable. Theorem 5 can now be extended in such a way that the Russell-style universes are faithfully emulated by the Tarski-style universes with coercive subtyping.

4.3 Injectivity and Constructor Subtyping

In subsumptive subtyping, $A \leq B$ means that A is directly embedded in B. Intuitively, this may imply that, for a and a' in A, if they are not equal in B, then they are not equal in A, either. If we think of sets, we know that if a set A is a subset of another set B, then one can always define an injective mapping from A to B. If we consider coercive subtyping $A \leq_c B$, this would translate to the requirement that c is injective in the sense that $c(a) = c(a')$ implies that $a = a'$. Here I shall formally discuss this issue in the context of representing intuitive subtyping notions by means of coercions.

In what follows I explore a particular situation which exhibits injectivity. More precisely, I look at Leibniz equality for a system with constructor subtyping as developed by Barthe and Frade [BF99]. I introduced this example in Subsection 2.3.1. In this setting an (inductive) type is considered to be a subtype of another if the latter has more constructors than the former. Here I discuss the example they start from, namely Even Numbers(*Even*) being a subtype of Natural Numbers (*Nat*) with the argument that the constructors of *Even* are 0 and successor of *Odd*, where *Odd* is given by the constructor successor of *Even*. Then, in *Nat* the successor constructor is overloaded to a lifting of these constructors as well. Formally they write:

```
datatype Odd = S of Even and Even = 0
     |S of Odd
datatype Nat  = 0
     |S of Nat
     |S of Odd
     |S of Even
```

Leibniz equality is defined as follows: $x = y$ if for any predicate P, $P(x) \iff P(y)$. We denote by $x =_A y$ for some type A the Leibniz equality between x and y related to a certain domain. Then, we have injectivity of subtyping if, given $x =_{Nat} y$, with $x, y{:}Even$ it is the case that $x =_{Even} y$. Namely, whether for any predicate $Q{:}Even \longrightarrow Prop$, it is the case that $Q(x) \iff Q(y)$. For this it is enough to show that any predicate $Q{:}Even \longrightarrow Prop$ admits a lifting $Q'{:}Nat \longrightarrow Prop$ such that for any $x{:}Even, Q'(x) \Longrightarrow Q(x)$. We can easily define such a Q' as follows:

```
Q'(x) =  Q(0) if x = 0
         Q(S(n)) if x = S of n:Odd
          true if x = S of n:Even
          true if x = S of n:Nat
```

Injectivity of the embedding holds here but it does not transfer without additionally imposing properties for the coercion.

To represent the example above in a coercive subtyping calculus we can consider a *predicate subtyping* as in [BB08], or if we want to see even natural numbers as a dependent pair(*Sigma Type*), as described by Luo [Luo99] when talking about adjectives associated with nouns. In either case, the coercion will be the first projection. In Coq proof assistant [Coq10] for example, we can write it like this:

```
Inductive Nat : Type :=
    | O : Nat
    | S : Nat -> Nat.
Inductive even : Nat -> Prop :=
    | O1 : even O
    | S1 : forall n1, even n1 -> even (S (S n1)).
Inductive Even := pair{n:Nat; e:even n}.
Definition proj1(ev:Even) :=
    match
        ev with pair n e => n
        end.
Coercion proj1 : Even >-> Nat.
```

Note that the definition of *Even* changed and we refer to it as a feature of the natural numbers rather than as a subset. In order for a natural number to be even we require a proof of that. Also note that this discussion uses first order data types, and it would be considerably more complicated for higher order data types.

We can have two proofs that 4 is even $p_1, p_2{:}even4$, and hence, two pairs $(4, p_1), (4, p_2){:}Even$ mapped to the same $4{:}Nat$. Enforcing injectivity here is similar to enforcing proof irrelevance. This happens to hold for this example, in particular, as proved by Hedberg [Hed98]. Simply put, any two proofs that

O is even will be equal as they reduce to $O1$. Then, for any even number $n = S(S(m))$ (where the even number m can be understood as $n-2$), we do induction on n and we have by induction hypothesis that any two proofs that m is even are equal. Let such a proof be $p{:}even(S(S(m)))$, then any proof $q{:}even(S(S(n)))$ that n is even is of the form $S1(S(S(m)))p$ so any two such proofs are also equal.

An extended example that no longer proves injectivity is as follows.

```
Inductive Nat : Type :=
    | O : Nat
    | S : Nat -> Nat.
Inductive even : Nat -> Prop :=
    | O1 : even O
    | O2 : even O
    | S1 : forall n1, even n1 -> even (S (S n1)).
Inductive Even := pair{n:Nat; e:even n}.
Definition proj1(ev:Even) :=
    match
        ev with pair n e => n
        end.
Coercion proj1 : Even >-> Nat.
```

The reason this coercion is not injective is that we can have two different proofs that O is even $O1, O2{:}evenO$, and hence, two different pairs $(O, O1), (O, O2){:}Even$, both of them being mapped to the same $O{:}Nat$.

So the following definition makes sense. For functions $f{:}(x{:}A)B$ I denote

$$injective(f) = \forall x, y{:}A.f(x) =_B f(y) \longrightarrow x =_A y$$

. A function f is then injective if $\exists p{:}injective(f)$.

Definition 12. *Let Σ, A, B, c such that $\vdash_\Sigma A{:}Type$, $\vdash_\Sigma B{:}Type$, $\vdash_\Sigma A \leq_c B$ are derivable. Then c is* ***injective*** *under Σ if $\vdash_\Sigma p{:}injective(c)$ is derivable* *for some p.*

In particular, for a constant coercion (namely of the form $\vdash_{\Sigma_0, c:(A)B, \Sigma_1, A \leq_c B, \Sigma_2, \Sigma_3} A \leq_c B$) we can add the assumption that it is injective

$$\vdash_{\Sigma_0, c:(A)B, \Sigma_1, A \leq_c B, \Sigma_2, p:injective(c), \Sigma_3} A \leq_c B$$

If we embed a subsumptive subtyping that propagates an equality from a type throughout its subtypes, we represent it as a constant coercion, thus, all we need to do is add the assumption that a coercion is injective. It is obvious that the transitivity and congruence preserve the injectivity property.

Chapter 5

Conclusion and Future Topics

In this thesis I introduced a new formulation of subtyping in type theories specified in a logical framework with signatures. The logical framework considered here is a variant of Luo's logical framework [Luo94] but I expect a similar development can be done for type theories such as Martin-Löf's specified in Martin-Löf's logical framework [NPS90] as they are similar.

This formulation achieves a balance between being powerful enough to represent some practical situations from the area of subsumptive subtyping and having a reasonably easy meta-theory. In addition. it is a formulation closer to the programming model of proof assistants compared to the previous systems of coercive subtyping from [LSX13, Xue13b] but still very much related to it, a thing which I have used to prove that the system I introduced here is well-behaved as an extension of the original system without subtyping in that it is a conservative extension and every derivable judgement in it is only an abbreviation of a derivable judgement in the base system.

For the examples considered here, I chose certain inductive types but other inductive types can be considered in a similar manner. More precisely, I discussed in Subsection 4.1.2 the coherence of the system with Π - types introduced with *app* constant. I expect the same situation for a system with Σ - types introduced with projections. I also discussed why an additional functoriality rule discussed in [LA08] would be required to ensure the coherence if the eliminator was used instead of *app* constant. A functoriality rule would also ensure coherence for other inductive types introduced with eliminator.

The development in this thesis raised some questions open for future work which is what I present in the rest of this chapter.

5.1 Definitionality

The discussion in this section was also presented during a talk at Types'17 using $Sigma$ - types. Here I shall use Π - types as the system Π has been often discussed in the thesis and the rules for it can be found in the Appendix A.

I mentioned in Section 3.4 that Luo et al. [LSX13, Xue13b, Xue13a] refer to the well-behavedness in which T' is an extension of T and

1. T' is conservative: any judgement in T derivable in T' is derivable in T
2. every valid derivation tree D' in T' can be translated into a valid derivation tree D in T such that the conclusion of D' is definitionally equal to the conclusion of D

as definitional extension.

Indeed, this definition captures a generalization of the Kleene's [Kle52] idea of extension by definition by expressing that the extension is nothing more than an abbreviation for certain judgements in the base system. But how comprehensive is it for type theory?

If we consider a straightforward translation of Kleene's definition, we obtain a notion of definitional extension related to the notion of conservativity as studied in [Hof95, Lum10] where an embedding of a type theory into its extension is used and induces a particular notion of definitional extension with new symbols. In the setting of this thesis, it can be formulated as follows.

1. Let T and T' be type theories specified in LF and T' an extension of T by adding new terms and rules.
2. Let f be a mapping from the terms of T' to those of T such that
   - $f|_T = id_T$, and

- the new rules involving the new terms in T' all become admissible under f in T.

Then T' is a definitional extension of T if and only if T' is a conservative extension of T and the definition rules of the form

$$\frac{\Gamma \vdash k{:}K \quad \Gamma \vdash f(k){:}K}{\Gamma \vdash k = f(k) : K}$$

are admissible in T'.

Another way to think of this is that T' extends T with new terms and new rules, including those definition rules which correspond to the definition axiom in Kleene's setting of first-order theories.

Observe that f induces a mapping on judgements $\overline{f}$ as follows:

1. $x_1{:}K_1, ..., x_n{:}K_n \vdash El(A)\ \ kind$
   $\longmapsto x_1{:}f(K_1), ..., x_n{:}f(K_n) \vdash El(f(A))\ \ kind$

2. $x_1{:}K_1, ..., x_n{:}K_n \vdash (x{:}K)L\ \ kind$
   $\longmapsto x_1{:}f(K_1), ..., x_n{:}f(K_n) \vdash (x{:}f(K))f(L)\ \ kind$

3. $x_1{:}K_1, ..., x_n{:}K_n\ \ valid \longmapsto x_1{:}f(K_1), ..., x_n{:}f(K_n)\ \ valid$

4. $x_1{:}K_1, ..., x_n{:}K_n \vdash K = L \longmapsto x_1{:}f(K_1), ..., x_n{:}f(K_n) \vdash f(K) = f(L)$

5. $x_1{:}K_1, ..., x_n{:}K_n \vdash k{:}K \longmapsto x_1{:}f(K_1), ..., x_n{:}f(K_n) \vdash f(k){:}f(K)$

6. $x_1{:}K_1, ..., x_n{:}K_n \vdash k = l{:}K$
   $\longmapsto x_1{:}f(K_1), ..., x_n{:}f(K_n) \vdash f(k) = f(l){:}f(K)$

and because of definition rules, context replacement and equality rules we obtain that if J is derivable in T', then $\overline{f}(J)$ is also derivable in T.

In this setting, some meta-theoretic properties are carried over from T to its definitional extension T'. For instance, if kind uniqueness holds for T, so does it for T'. To show this, let us consider $\Gamma \vdash k{:}K$ and $\Gamma \vdash k{:}L$ derivable in T'. Then $f(\Gamma) \vdash f(k){:}f(K)$ and $f(\Gamma) \vdash f(k){:}f(L)$ are derivable in T and because kind uniqueness holds here we know that $f(\Gamma) \vdash f(K) = f(L)$. By repeated application of definition rules and context replacement, $f(\Gamma) = \Gamma$ and $\Gamma \vdash f(K) = f(L)$ are derivable in T'. Again by definition rules $\Gamma \vdash f(K) = K$ $\Gamma \vdash f(L) = L$ are derivable in T', we have that $\Gamma \vdash K = L$ is also derivable in T'.

To exemplify the syntactic mapping justification of definitionality, we can consider Π, a type theory specified in LF with Π-types and $\Pi[\longrightarrow]$, the extension of Π with function type and the rules in Figure 5.1

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma \vdash B : \mathit{Type}}{\Gamma \vdash A \longrightarrow B : \mathit{Type}}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma \vdash B : \mathit{Type} \quad \Gamma \vdash f : (A)B}{\Gamma \vdash \lambda_{\longrightarrow}(A, B, f) : A \longrightarrow B}$$

$$\frac{\Gamma \vdash g{:}A \longrightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash \mathit{app}_{\longrightarrow}(A, B, g, a) : B}$$

$$\frac{\begin{array}{c}\Gamma \vdash A : \mathit{Type} \quad \Gamma \vdash B : \mathit{Type} \\ \Gamma \vdash f : (A)B \quad \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathit{app}_{\longrightarrow}(A, B, \lambda_{\longrightarrow}(A, B, f), a) = f(a) : B}$$

Figure 5.1: Inference Rules for $\longrightarrow$ - type

A syntactic map $f{:}Term(\Pi[\longrightarrow]) \longrightarrow Term(\Pi)$ can be defined as follows

1. $f|_{Term(\Pi)} = Id_{Term(\Pi)}$
2. $f(A \longrightarrow B) = \Pi(A, [x{:}A]B)$
3. $f(\lambda_{\longrightarrow}(A, B, f)) = \lambda(A, [x{:}A]B, f)$
4. $f(\mathit{app}_{\longrightarrow}(A, B, g, a)) = \mathit{app}(A, B, g, a)$

Then, if in $\Pi[\longrightarrow]$ the definition rules

$$\frac{\Gamma \vdash A{:}\mathit{Type} \quad \Gamma \vdash B{:}\mathit{Type}}{\Gamma \vdash A \longrightarrow B = \Pi(A, [x{:}A]B)}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma \vdash B : \mathit{Type} \quad \Gamma \vdash f : (A)B}{\Gamma \vdash \lambda_{\longrightarrow}(A, B, f) = \lambda(A, [x{:}A]B, f) : \Pi(A, [x{:}A]B}$$

$$\frac{\Gamma \vdash g{:}A \longrightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash \mathit{app}_{\longrightarrow}(A, B, g, a) = \mathit{app}(A, [x{:}A]B, g, a) : ([x{:}A]B)(a)}$$

are admissible, then $\Pi[\longrightarrow]$ is a definitional extension of Π. Note that, in this case, $([x{:}A]B)(a) = [a/x]B = B$ as B does not depend on A.

Unfortunately this straightforward way of expressing definitionality in type theory leaves out some situations in which the extension and the base system have the same set of terms, such as the extension with coercive subtyping discussed in Chapter 3 and [LSX13, Xue13b]. However, we have clearly seen that coercive subtyping gives a way to abbreviate some judgements in the base system and nothing more so the natural way to think of it is that this extension is too, a definitional extension in a sense. This was discussed in [LSX13, Xue13b, Xue13a] and led to the definition I mentioned at the beginning of this section. This definition needs to take derivation trees into consideration. This definition is in a sense a generalization of the sense that uses a syntactic mapping. To see this, it might be helpful to rephrase it so that it focuses on judgements rather than derivation trees as follows:

T' is an extension of T that adds new terms and rules then T' is definitional if and only if:

1. It is conservative.

2. For every derivable judgement J' in T', for any D a derivation tree of J', there exists J derivable in T such that $J' =_D J$.

where $J' =_D J$ is defined inductively by the fact that, if

$$D \equiv \frac{\frac{D_1}{J'_1} \ \cdots \ \frac{D_n}{J'_n}}{J'}$$

J and J' are definitionally equal and there exist $J_1, ..., J_n$ such that $J'_1 =_{D_1} J_1$, ..., $J'_n =_{D_n} J_n$ and

$$\frac{J_1 \cdots J_n}{J}$$

is admissible in T.

However this definition is still not general enough to cover new forms of judgements as we don't have definitional equality available for judgements of different forms. Because of this, I think an important step in the direction of establishing a comprehensive definition of definitionality or, so to say, what it means to be a syntactic sugar for type theories would be to generalize the meaning of definitionality to the point that it does cover new forms of judgements.

5.2 Parameterized and Dependent Coercions

Parameterized coercions, in the sense of point-wise subtyping described by Luo and Soloviev [LS99, SL02] are coercions parameterized over free variables, for example, if $A \leq_c B$ then for any n, $Vect_A(n) \leq_{\overline{c}} Vect_B(n)$ where $\overline{c}(a_1, ..., a_n) = (c(a_1), ..., c(a_n))$. Here $\overline{c}$ is parameterized by n. This kind of coercions are used for example in the study of natural language semantic (Asher and Luo [AL12]).

An interesting future work would be to study how, if possible, one can represent this in the system with coercive subtyping entries in signatures as we don't have free variables available there. Similarly, it would be interesting to study the potential of this system to represent dependent coercions in the sense of [LS99], of the form $x{:}A \leq_c B(x)$. A possibility would be to allow the substitution and not abstraction for the entries in signatures but how to do this and the consequences have yet to be studied.

Bibliography

[AC01] D. Aspinall and A. Compagnoni. Subtyping dependent types. *Theoretical Computer Science*, 266:273–309, 2001.

[Agd08] The Agda proof assistant (version 2). `http://appserv.cs.chalmers.se/users/ulfn/wiki/agda.php`, 2008.

[AL12] Nicholas Asher and Zhaohui Luo. Formalization of coercions in lexical semantics. 2012.

[BB08] B. Barras and B. Bernardo. The implicit calculus of constructions as a programming language with dependent types. *Foundations of Software Science and Computational Structures*, 4962:365–379, 2008.

[BCGS91] Valeriu Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance and explicit coercion. *Information and Computation*, 93, 1991.

[BF99] G. Barthe and M. J. Frade. Constructor subtyping. *Lecture Notes in Computer Science*, 1576:109–127, 1999.

[BT98] Gustavo Betarte and Alvaro Tasistro. Extension of Martin-Löf's type theory with record types and subtyping. Oxford University Press, 1998.

[Chu32] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *J Symbolic Logic*, (5):56–68, 1940.

[CL01] Paul Callaghan and Zhaohui Luo. An implementation of LF with coercive subtyping and universes. *Journal of Automated Reasoning*, 27(1):3–27, 2001.

[CL14] Stergios Chatzikyriakidis and Zhaohui Luo. Natural language inference in Coq. *J. of Logic, Lang. and Inf.*, 23(4):441–480, December 2014.

[CL15] S. Chatzikyriakidis and Z. Luo. Using signatures in type theory to represent situations. *T. Murata, K. Mineshima and D. Bekki (eds). New Frontiers in Artificial Intelligence - JSAI-isAI 2014 Workshops in Japan (LENLS, JURISIN and GABA), Revised Selected Papers. LNCS 9067*, 2015.

[Coq86] Thierry Coquand. An analysis of Girard's paradox. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, pages 227–236, 1986.

[Coq10] The Coq Development Team. *The Coq Proof Assistant Reference Manual (Version 8.3), INRIA*, 2010.

[CP90] Thierry Coquand and Christine Paulin. Inductively defined types. In *Proceedings of the International Conference on Computer Logic*, COLOG '88, pages 50–66, London, UK, UK, 1990. Springer-Verlag.

[Dyb91] P. Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In G. Huet and G. Plotkin, editors, *Logical Frameworks.* Cambridge University Press, 1991.

[Gir72] J. Y. Girard. *Interprètation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* PhD thesis, Universit Paris VII, 1972.

[Gog94] Healfdene Goguen. *A Typed Operational Semantics for Type Theory.* PhD thesis, University of Edinburgh, 1994.

[Hed98] Michael Hedberg. A coherence theorem for Martin-Löf's type theory. *J. Funct. Program.*, 8(4):413–436, July 1998.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40:143–184, 1993.

[Hof95] M. Hofmann. *Extensional concepts in intensional type theory.* PhD thesis, Univ of Edinburgh, 1995.

[JG94] Razvan Diaconescu Joseph Goguen. An Oxford survey of order sorted algebra. *Mathematical Structures in Computer Science*, (2):363–392, 1994.

[Kle52] S. Kleene. *Introduction to Mathematics.* North Holland, 1952.

[LA08] Zhaohui Luo and Robin Adams. Structural subtyping for inductive types with functorial equality rules. *Mathematical Structures in Computer Science*, 18(5), 2008.

[LL01] Yong Luo and Zhaohui Luo. Coherence and transitivity in coercive subtyping. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2250 of *Lecture Notes in Computer Science*, pages 249–265. Springer Berlin Heidelberg, November 2001.

[LL14] Georgiana E. Lungu and Zhaohui Luo. Monotonicity reasoning in formal semantics based on modern type theories. *Lecture Notes in Computer Science*, 8535:138–148, 2014.

[LLss] Georgiana E. Lungu and Zhaohui Luo. On subtyping in type theories with canonical objects. In *Postproceedings of Types for Proofs and Programs*, in press.

[LLS02] Yong Luo, Zhaohui Luo, and Sergei Soloviev. Weak transitivity in coercive subtyping. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs*, volume 2646 of *LNCS*, pages 220–239. Springer-Verlag, 2002.

[LP13] Zhaohui Luo and Fjodor Part. Subtyping in type theory: Coercion contexts and local coercions, TYPES 2013, Toulouse. 2013.

[LS99] Zhaohui Luo and Sergei Soloviev. Dependent coercions. *Electr. Notes Theor. Comput. Sci.*, 29:152–168, 1999.

[LSX13] Zhaohui Luo, Sergei Soloviev, and Tao Xue. Coercive subtyping: theory and implementation. *Information and Copmutation*, 223:18–42, 2013.

[Lum10] Peter LeFanu Lumsdaine. *Higher Categories from Type Theories.* PhD thesis, Carnegie Mellon University, 2010.

[Luo90] Z. Luo. *An Extended Calculus of Constructions.* PhD thesis, University of Edinburgh, 1990.

[Luo92] Zhaohui Luo. A unifying theory of dependent types: The schematic approach. In *Proceedings of the Second International Symposium on Logical Foundations of Computer Science*, TVER '92, pages 293–304, London, UK, UK, 1992. Springer-Verlag.

[Luo94] Zhaohui Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Oxford University Press, 1994.

[Luo96] Zhaohui Luo. Coercive subtyping in type theory. In *Proc. of CSL'96, the 1996 Annual Conference of the European Association for Computer Science Logic, Utrecht. LNCS 1258*, page draft., 1996.

[Luo99] Zhaohui Luo. Coercive subtyping. *Journal of Logic and Computation*, 9, 1999.

[Luo05] Yong Luo. *Coherence and Transitivity in Coercive Subtyping.* PhD thesis, University of Durham, 2005.

[Luo12a] Zhaohui Luo. Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophy*, 35(6):491–513, 2012.

[Luo12b] Zhaohui Luo. Notes on universes in type theory (for a talk given at institute of advanced studies), 2012.

[LW94] Barbara H. Liskov and Jeanette M. Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16:1811–1841, 1994.

[Mit84] John C. Mitchell. Coercion and type inference. In *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '84, pages 175–185, New York, NY, USA, 1984. ACM.

[ML73] Per Martin-Löf. An intuitionistic theory of types: Predicative part. 80, 01 1973.

[ML75] Per Martin-Löf. An intuitionistic theory of types: predicative part. *Studies in Logic and the Foundations of Mathematics*, 80:73–118, 1975.

[ML82] Per Martin-Löf. Constructive mathematics and computer programming. *Stud. Logic Found. Math*, 104:153–175, 1982.

[ML84] Per Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

[ML98] Per Martin-Löf. An intuitionistic theory of types. *Oxford Logic Guides*, 36:127–172, 1998.

[NPS90] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Clarendon Press, New York, NY, USA, 1990.

[Pfe02] Frank Pfenning. Logical frameworksa brief introduction. In Helmut Schwichtenberg and Ralf Steinbrggen, editors, *Proof and System-Reliability*, volume 62 of *NATO Science Series*, pages 137–166. Springer Netherlands, 2002.

[Pie93] Benjamin C. Pierce. Bounded quantification is undecidable. In *Information and Computation*, pages 305–315, 1993.

[PM93] Christine Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, TLCA '93, pages 328–345, London, UK, UK, 1993. Springer-Verlag.

[Pol94] Robert Pollack. *The Theory of LEGO – A Proof Checker for the Extended Calculus of Constructions.* PhD thesis, 1994.

[Ran94] Aarne Ranta. *Type-Theoretical Grammar.* Monograph Collection (Matt - Pseudo), 1994.

[Rey80] John C. Reynolds. Using category theory to design implicit conversions and generic operators. *Semantics-Directed Compiler Generation 1980*, Lecture Notes in Computer Science 94, 1980.

[SL02] Sergei Soloviev and Zhaohui Luo. Coercion completion and conservativity in coercive subtyping. *Annals of Pure and Applied Logic*, 113(1-3):297–322, 2002.

[Smi88] Jan M. Smith. The independence of Peano's fourth axiom from Martin-Lof's type theory without universes. *The Journal of Symbolic Logic*, 53(3):840–845, 1988.

[Uni13] Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* Institute for Advanced Study, 2013.

[vB92] S. van Bakel. Principal type schemes for the strict type assignment system. *Journal of Logic and Computation*, 3:643–670, 1992.

[Xue13a] Tao Xue. Definitional extension in type theory. In Ralph Matthes and Aleksy Schubert, editors, *LIPIcs Proceedings 19th Internationa Conference on Types for Proofs and Programs*, pages 251–269, 2013.

[Xue13b] Tao Xue. *Theory and Implementation of Coercive Subtyping.* PhD thesis, Royal Holloway University of London, 2013.

Appendices

Appendix A

Inference Rules for LF and Π - type

This appendix puts together the rules for a system Π which is the system with dependent product type specified in LF.

Validity of Signature/Contexts, Assumptions

$$\frac{}{\vdash \langle\rangle} \qquad \frac{\Gamma \vdash K\,kind \quad x \notin dom(\Gamma)}{\vdash \Gamma, x{:}K} \qquad \frac{\vdash \Gamma, x{:}K, \Gamma'}{\Gamma, x{:}K, \Gamma' \vdash x{:}K}$$

Equality Rules

$$\frac{\Gamma \vdash K \ \ kind}{\Gamma \vdash K = K} \qquad \frac{\Gamma \vdash K = K'}{\Gamma \vdash K' = K} \qquad \frac{\Gamma \vdash K = K' \quad \Gamma \vdash K' = K''}{\Gamma \vdash K = K''}$$

$$\frac{\Gamma \vdash k{:}K}{\Gamma \vdash k = k{:}K} \qquad \frac{\Gamma \vdash k = k'{:}K}{\Gamma \vdash k' = k{:}K} \qquad \frac{\Gamma \vdash k = k'{:}K \quad \Gamma \vdash k' = k''{:}K}{\Gamma \vdash k = k''{:}K}$$

$$\frac{\Gamma \vdash k{:}K \quad \Gamma \vdash K = K'}{\Gamma \vdash k{:}K'} \qquad \frac{\Gamma \vdash k = k'{:}K \quad \Gamma \vdash K = K'}{\Gamma \vdash k = k'{:}K'}$$

Substitution Rules

$$\frac{\vdash \Gamma_0, x{:}K, \Gamma_1 \quad \Gamma_0 \vdash k{:}K}{\vdash \Gamma_0, [k/x]\Gamma_1}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash K' \ \ kind \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K' \ \ kind} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash L = L' \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]L = [k/x]L'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash k'{:}K' \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash l = l'{:}K' \quad \Gamma_0 \vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]l = [k/x]l'{:}[k/x]K'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash K' \ \ kind \quad \Gamma_0 \vdash k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K' = [k'/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash l{:}K' \quad \Gamma_0 \vdash k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]l = [k'/x]l{:}[k/x]K'}$$

Dependent Product Kinds

$$\frac{\Gamma \vdash K\,kind \quad \Gamma, x{:}K \vdash K'\,kind}{\Gamma \vdash (x{:}K)K'\,kind} \qquad \frac{\Gamma \vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash K_1' = K_2'}{\Gamma \vdash (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Gamma, x{:}K \vdash y{:}K'}{\Gamma \vdash [x{:}K]y{:}(x{:}K)K'} \qquad \frac{\Gamma \vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash k_1 = k_2{:}K}{\Gamma \vdash [x{:}K_1]k_1 = [x{:}K_2]k_2{:}(x{:}K_1)K}$$

$$\frac{\Gamma \vdash f{:}(x{:}K)K' \quad \Gamma \vdash k{:}K}{\Gamma \vdash f(k){:}[k/x]K'} \qquad \frac{\Gamma \vdash f = f'{:}(x{:}K)K' \quad \Gamma \vdash k_1 = k_2{:}K}{\Gamma \vdash f(k_1) = f'(k_2){:}[k_1/x]K'}$$

$$\frac{\Gamma, x{:}K \vdash k'{:}K' \quad \Gamma \vdash k{:}K}{\Gamma \vdash ([x{:}K]k')(k) = [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma \vdash f{:}(x{:}K)K' \quad x \notin FV(f)}{\Gamma \vdash [x{:}K]f(x) = f{:}(x{:}K)K'}$$

The kind Type

$$\frac{\vdash \Gamma}{\Gamma \vdash Type \ \ kind} \qquad \frac{\Gamma \vdash A{:}Type}{\Gamma \vdash El(A) \ \ kind} \qquad \frac{\Gamma \vdash A = B{:}Type}{\Gamma \vdash El(A) = El(B)}$$

Figure A.1: Inference Rules for *LF*

$$\frac{\Gamma \vdash A : Type \quad \Gamma, x{:}A \vdash B(x) : Type}{\Gamma \vdash \Pi(A, B) : Type}$$

$$\frac{\Gamma \vdash A : Type \quad \Gamma \vdash B : (A)Type \quad \Gamma \vdash f : (x{:}A)B(x)}{\Gamma \vdash \lambda(A, B, f) : \Pi(A, B)}$$

$$\frac{\Gamma \vdash g : \Pi(A, B) \quad \Gamma \vdash a : A}{\Gamma \vdash app(A, B, g, a) : B(a)}$$

$$\frac{\begin{array}{c}\Gamma \vdash A : Type \quad \Gamma \vdash B : (A)Type \\ \Gamma \vdash f : (x{:}A)B(x) \quad \Gamma \vdash a : A\end{array}}{\Gamma \vdash app(A, B, \lambda(A, B, f), a) = f(a) : B(a)}$$

Figure A.2: Inference Rules for Π - type specified in *LF*

Appendix B

Rules for Universes

for $i \in \omega$

$$\frac{\Gamma\ valid}{\Gamma \vdash U_i : Type} \qquad \frac{\Gamma \vdash A : U_i}{\Gamma \vdash A : Type} \qquad \frac{\Gamma\ valid}{\Gamma \vdash U_i : U_{i+1}} \qquad \frac{\Gamma \vdash A : U_i}{\Gamma \vdash A : U_{i+1}}$$

$$\frac{\Gamma \vdash A : U_i \quad \Gamma \vdash B : (A)U_i}{\Gamma \vdash \Pi(A, B) : U_i}$$

Figure B.1: Inference Rules for Russell Style Universes with Π - type

for $i \in \omega$

$$\frac{\vdash \Gamma}{\Gamma \vdash U_i : Type} \qquad \frac{\Gamma \vdash a : U_i}{\Gamma \vdash T_i(a) : Type} \qquad \frac{\vdash \Gamma}{\Gamma \vdash t_{i+1} : (U_i)U_{i+1}}$$

where t_{i+1} are the lifting operators,

$$\frac{\vdash \Gamma}{\Gamma \vdash u_i : U_{i+1}} \qquad \frac{\vdash \Gamma}{\Gamma \vdash T_{i+1}(u_i) = U_i : Type}$$

where u_i is the name of U_i in U_{i+1}

$$\frac{\Gamma \vdash a : U_i \quad \Gamma,\ x : T_i(a) \vdash b(x) : U_i}{\Gamma \vdash \pi_i(a, b) : U_i}$$

with the following equations:

1. $\Gamma \vdash T_{i+1}(\pi_i(a, b)) = \Pi(T_{i+1}(a), [x{:}T_i(a)]T_{i+1}(b(x))) : Type$
2. $\Gamma \vdash t_{i+1}(\pi_i(a, b)) = \pi_{i+1}(t_{i+1}(a), [x{:}T_i(a)]t_{i+1}(b(x))) : U_{i+1}$

Figure B.2: Inference Rules for Tarski Style Universes with Π - type

Appendix C

Inference Rules for LF_S, $T^{0K}_{S,\leq}$, $T_{S,\leq}$, $\Pi^{0K}_{S,\leq}$ and $\Pi_{S,\leq}$

In this appendix I list the rules of the system LF_S, $T^{0K}_{S,\leq}$ and $T_{S,\leq}$. All these rules are part of the systems $\Pi^{0K}_{S,\leq}$ and $\Pi_{S,\leq}$ as well. For these two systems there are additional rules for Π - type which are also presented here.

Validity of Signature/Contexts, Assumptions

$$\frac{}{\langle\rangle\ \mathit{valid}} \qquad \frac{\vdash_\Sigma K\ \mathit{kind} \quad c \notin \mathit{dom}(\Sigma)}{\Sigma, c{:}K\ \ \mathit{valid}} \qquad \frac{\vdash_{\Sigma,c:K,\Sigma'} \Gamma}{\Gamma \vdash_{\Sigma,c:K,\Sigma'} c{:}K}$$

$$\frac{\Sigma\ \ \mathit{valid}}{\vdash_\Sigma \langle\rangle} \qquad \frac{\Gamma \vdash_\Sigma K\ \mathit{kind} \quad x \notin \mathit{dom}(\Sigma) \cup \mathit{dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}K} \qquad \frac{\vdash_\Sigma \Gamma, x{:}K, \Gamma'}{\Gamma, x{:}K, \Gamma' \vdash_\Sigma x{:}K}$$

Weakening

$$\frac{\Gamma \vdash_{\Sigma,\ \Sigma'} J \quad \vdash_\Sigma K\ \mathit{kind} \quad c \notin \mathit{dom}(\Sigma, \Sigma')}{\Gamma \vdash_{\Sigma,\ c:K,\ \Sigma'} J}$$

$$\frac{\Gamma, \Gamma' \vdash_\Sigma J \quad \Gamma \vdash_\Sigma K\ \mathit{kind} \quad x \notin \mathit{dom}(\Gamma, \Gamma')}{\Gamma, x{:}K, \Gamma' \vdash_\Sigma J}$$

Equality Rules

$$\frac{\Gamma \vdash_\Sigma K\ \ \mathit{kind}}{\Gamma \vdash_\Sigma K = K} \qquad \frac{\Gamma \vdash_\Sigma K = K'}{\Gamma \vdash_\Sigma K' = K} \qquad \frac{\Gamma \vdash_\Sigma K = K' \quad \Gamma \vdash_\Sigma K' = K''}{\Gamma \vdash_\Sigma K = K''}$$

$$\frac{\Gamma \vdash_\Sigma k{:}K}{\Gamma \vdash_\Sigma k = k{:}K} \qquad \frac{\Gamma \vdash_\Sigma k = k'{:}K}{\Gamma \vdash_\Sigma k' = k{:}K} \qquad \frac{\Gamma \vdash_\Sigma k = k'{:}K \quad \Gamma \vdash_\Sigma k' = k''{:}K}{\Gamma \vdash_\Sigma k = k''{:}K}$$

$$\frac{\Gamma \vdash_\Sigma k{:}K \quad \Gamma \vdash_\Sigma K = K'}{\Gamma \vdash_\Sigma k{:}K'} \qquad \frac{\Gamma \vdash_\Sigma k = k'{:}K \quad \Gamma \vdash_\Sigma K = K'}{\Gamma \vdash_\Sigma k = k'{:}K'}$$

Signature Replacement

$$\frac{\Gamma \vdash_{\Sigma_0, c:L, \Sigma_1} J \quad \vdash_{\Sigma_0} L = L'}{\Gamma \vdash_{\Sigma_0, c:L', \Sigma_1} J}$$

Context Replacement

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma J \quad \Gamma_0 \vdash_\Sigma K = K'}{\Gamma_0, x{:}K', \Gamma_1 \vdash_\Sigma J}$$

Substitution Rules

$$\frac{\vdash_\Sigma \Gamma_0, x{:}K, \Gamma_1 \quad \Gamma_0 \vdash_\Sigma k{:}K}{\vdash_\Sigma \Gamma_0, [k/x]\Gamma_1}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma K'\ \ \mathit{kind} \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]K'\ \ \mathit{kind}} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma L = L' \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]L = [k/x]L'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma k'{:}K' \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma l = l'{:}K' \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]l = [k/x]l'{:}[k/x]K'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma K'\ \ \mathit{kind} \quad \Gamma_0 \vdash_\Sigma k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]K' = [k'/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma l{:}K' \quad \Gamma_0 \vdash_\Sigma k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]l = [k'/x]l{:}[k/x]K'}$$

Dependent Product Kinds

$$\frac{\Gamma \vdash_\Sigma K\ \mathit{kind} \quad \Gamma, x{:}K \vdash_\Sigma K'\ \mathit{kind}}{\Gamma \vdash_\Sigma (x{:}K)K'\ \mathit{kind}} \qquad \frac{\Gamma \vdash_\Sigma K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash_\Sigma K_1' = K_2'}{\Gamma \vdash_\Sigma (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Gamma, x{:}K \vdash_\Sigma y{:}K'}{\Gamma \vdash_\Sigma [x{:}K]y{:}(x{:}K)K'} \qquad \frac{\Gamma \vdash_\Sigma K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash_\Sigma k_1 = k_2{:}K}{\Gamma \vdash_\Sigma [x{:}K_1]k_1 = [x{:}K_2]k_2{:}(x{:}K_1)K}$$

$$\frac{\Gamma \vdash_\Sigma f{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k{:}K}{\Gamma \vdash_\Sigma f(k){:}[k/x]K'} \qquad \frac{\Gamma \vdash_\Sigma f = f'{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k_1 = k_2{:}K}{\Gamma \vdash_\Sigma f(k_1) = f'(k_2){:}[k_1/x]K'}$$

$$\frac{\Gamma, x{:}K \vdash_\Sigma k'{:}K' \quad \Gamma \vdash_\Sigma k{:}K}{\Gamma \vdash_\Sigma ([x{:}K]k')(k) = [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma \vdash_\Sigma f{:}(x{:}K)K' \quad x \notin FV(f)}{\Gamma \vdash_\Sigma [x{:}K]f(x) = f{:}(x{:}K)K'}$$

The kind Type

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathit{Type}\ \ \mathit{kind}} \qquad \frac{\Gamma \vdash_\Sigma A{:}\mathit{Type}}{\Gamma \vdash_\Sigma \mathit{El}(A)\ \ \mathit{kind}} \qquad \frac{\Gamma \vdash_\Sigma A = B{:}\mathit{Type}}{\Gamma \vdash_\Sigma \mathit{El}(A) = \mathit{El}(B)}$$

Figure C.1: Logical Framework Inference Rules for LF_S, Π_S, $T^{0K}_{S,\leq}$, $T_{S,\leq}$, $\Pi^{0K}_{S,\leq}$ and $\Pi_{S,\leq}$

$$\frac{\Gamma \vdash_\Sigma A : \mathit{Type} \quad \Gamma, x{:}A \vdash_\Sigma B(x) : \mathit{Type}}{\Gamma \vdash_\Sigma \Pi(A,B) : \mathit{Type}}$$

$$\frac{\Gamma \vdash_\Sigma A : \mathit{Type} \quad \Gamma \vdash_\Sigma B : (A)\mathit{Type} \quad \Gamma \vdash_\Sigma f : (x{:}A)B(x)}{\Gamma \vdash_\Sigma \lambda(A,B,f) : \Pi(A,B)}$$

$$\frac{\Gamma \vdash_\Sigma g : \Pi(A,B) \quad \Gamma \vdash_\Sigma a : A}{\Gamma \vdash_\Sigma \mathit{app}(A,B,g,a) : B(a)}$$

$$\frac{\begin{array}{c}\Gamma \vdash_\Sigma A : \mathit{Type} \quad \Gamma \vdash_\Sigma B : (A)\mathit{Type} \\ \Gamma \vdash_\Sigma f : (x{:}A)B(x) \quad \Gamma \vdash_\Sigma a : A\end{array}}{\Gamma \vdash_\Sigma \mathit{app}(A,B,\lambda(A,B,f),a) = f(a) : B(a)}$$

Figure C.2: Inference Rules for Π-type in Π_S, $\Pi^{0K}_{S,\leq}$ and $\Pi_{S,\leq}$

Signature Rules for Subtyping

$$\frac{\vdash_\Sigma A : \mathit{Type} \quad \vdash_\Sigma B : \mathit{Type} \quad \vdash_\Sigma c : (A)B}{\Sigma, A \leq_c B \ \ \mathit{valid}} \qquad \frac{\vdash_{\Sigma_0, A\leq_c B:\mathit{Type}, \Sigma_1} \Gamma}{\Gamma \vdash_{\Sigma_0, A\leq_c B:\mathit{Type}, \Sigma_1} A \leq_c B : \mathit{Type}}$$

Congruence

$$\frac{\Gamma \vdash_\Sigma A \leq_c B : \mathit{Type} \quad \Gamma \vdash_\Sigma A = A' : \mathit{Type} \quad \Gamma \vdash_\Sigma B = B' : \mathit{Type} \quad \Gamma \vdash_\Sigma c = c' : (A)B}{\Gamma \vdash_\Sigma A' \leq_{c'} B' : \mathit{Type}}$$

Transitivity

$$\frac{\Gamma \vdash_\Sigma A \leq_c A' : \mathit{Type} \quad \Gamma \vdash_\Sigma A' \leq_{c'} A'' : \mathit{Type}}{\Gamma \vdash_\Sigma A \leq_{c' \circ c} A'' : \mathit{Type}}$$

Weakening

$$\frac{\Gamma \vdash_{\Sigma,\ \Sigma'} A \leq_d B : \mathit{Type} \quad \vdash_\Sigma K \ \mathit{kind}}{\Gamma \vdash_{\Sigma,\ c:K,\ \Sigma'} A \leq_d B : \mathit{Type}} \quad (c \notin \mathit{dom}(\Sigma, \Sigma'))$$

$$\frac{\Gamma, \Gamma' \vdash_\Sigma A \leq_d B : \mathit{Type} \quad \Gamma \vdash_\Sigma K \ \mathit{kind}}{\Gamma, x{:}K, \Gamma' \vdash_\Sigma A \leq_d B : \mathit{Type}} \quad (x \notin \mathit{dom}(\Gamma, \Gamma'))$$

Signature Replacement

$$\frac{\Gamma \vdash_{\Sigma_0, c:L, \Sigma_1} A \leq_d B : \mathit{Type} \quad \vdash_{\Sigma_0} L = L'}{\Gamma \vdash_{\Sigma_0, c:L', \Sigma_1} A \leq_d B : \mathit{Type}}$$

Context Replacement

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma A \leq_d B : \mathit{Type} \quad \Gamma_0 \vdash_\Sigma K = K'}{\Gamma_0, x{:}K', \Gamma_1 \vdash_\Sigma A \leq_d B : \mathit{Type}}$$

Substitution

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma A \leq_c B \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]A \leq_{[k/x]c} [k/x]B}$$

Identity Coercion

$$\frac{\Gamma \vdash_\Sigma A : \mathit{Type}}{\Gamma \vdash_\Sigma A \leq_{[x:A]x} A : \mathit{Type}}$$

Figure C.3: Inference Rules for Subtyping in $T^{0K}_{S,\leq}$, $T_{S,\leq}$, $\Pi^{0K}_{S,\leq}$ and $\Pi_{S,\leq}$ (1)

Subtyping for dependent product Rule

$$\frac{\Gamma \vdash_\Sigma A' \leq_{c_1} A : \mathit{Type} \quad \Gamma \vdash_\Sigma B, B' : (A)\mathit{Type} \quad \Gamma, x{:}A \vdash_\Sigma B(x) \leq_{c_2[x]} B'(x) : \mathit{Type}}{\Gamma \vdash_\Sigma \Pi(A,B) \leq_d \Pi(A', B' \circ c_1) : \mathit{Type}}$$

where $d \equiv [F : \Pi(A,B)]\lambda(A', B' \circ c_1, [x{:}A']c_2[x](\mathit{app}(A,B,F,c_1(x))))$

Figure C.4: Subtyping for dependent product Rule for $\Pi^{0K}_{S,\leq}$ and $\Pi_{S,\leq}$

Basic Subkinding Rule and Identity Coercion

$$\frac{\Gamma \vdash_\Sigma A \leq_c B{:}\mathit{Type}}{\Gamma \vdash_\Sigma \mathit{El}(A) \leq_c \mathit{El}(B)} \qquad \frac{\Gamma \vdash_\Sigma K \ \ \mathit{kind}}{\Gamma \vdash_\Sigma K \leq_{[x:K]x} K}$$

Structural Subkinding Rules

$$\frac{\Gamma \vdash_\Sigma K_1 \leq_c K_2 \quad \Gamma \vdash_\Sigma K_1 = K_1' \quad \Gamma \vdash_\Sigma K_2 = K_2' \quad \Gamma \vdash_\Sigma c = c'{:}(K_1)K_2}{\Gamma \vdash_\Sigma K_1' \leq_{c'} K_2'}$$

$$\frac{\Gamma \vdash_\Sigma K \leq_c K' \quad \Gamma \vdash_\Sigma K' \leq_{c'} K''}{\Gamma \vdash_\Sigma K \leq_{c' \circ c} K''}$$

$$\frac{\Gamma \vdash_{\Sigma,\ \Sigma'} K \leq_d K' \quad \vdash_\Sigma K_0 \ \mathit{kind}}{\Gamma \vdash_{\Sigma,\ c:K_0,\ \Sigma'} K \leq_d K'} \quad (c \notin \mathit{dom}(\Sigma, \Sigma'))$$

$$\frac{\Gamma, \Gamma' \vdash_\Sigma K \leq_d K' \quad \Gamma \vdash_\Sigma K_0 \ \mathit{kind}}{\Gamma, x{:}K_0, \Gamma' \vdash_\Sigma K \leq_d K'} \quad (x \notin \mathit{dom}(\Gamma, \Gamma'))$$

$$\frac{\Gamma \vdash_{\Sigma_0, c:L, \Sigma_1} K \leq_d K' \quad \vdash_{\Sigma_0} L = L'}{\Gamma \vdash_{\Sigma_0, c:L', \Sigma_1} K \leq_d K'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma L \leq_d L' \quad \Gamma_0 \vdash_\Sigma K = K'}{\Gamma_0, x{:}K', \Gamma_1 \vdash_\Sigma L \leq_d L'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \vdash_\Sigma K_1 \leq_c K_2 \quad \Gamma_0 \vdash_\Sigma k{:}K}{\Gamma_0, [k/x]\Gamma_1 \vdash_\Sigma [k/x]K_1 \leq_{[k/x]c} [k/x]K_2}$$

Subkinding for Dependent Product Kind

$$\frac{\Gamma \vdash_\Sigma K_1' \leq_{c_1} K_1 \ \ \Gamma, x{:}K_1 \vdash_\Sigma K_2 \ \mathit{kind} \ \ \Gamma, x'{:}K_1' \vdash_\Sigma K_2' \ \mathit{kind} \ \ \Gamma, x{:}K_1 \vdash_\Sigma [c_1(x')/x]K_2 \leq_{c_2} K_2'}{\Gamma \vdash_\Sigma (x{:}K_1)K_2 \leq_{[f:(x:K_1)K_2][x':K_1']c_2(f(c_1(x')))} (x{:}K_1')K_2'}$$

Figure C.5: Inference Rules for Subkinding in $T^{0K}_{S,\leq}$, $T_{S,\leq}$, $\Pi^{0K}_{S,\leq}$ and $\Pi_{S,\leq}$ (2)

Coercive Application

(CA_1)
$$\frac{\Gamma \vdash_\Sigma f{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k_0{:}K_0 \quad \Gamma \vdash_\Sigma K_0 \leq_c K}{\Gamma \vdash_\Sigma f(k_0){:}[c(k_0)/x]K'}$$

(CA_2)
$$\frac{\Gamma \vdash_\Sigma f = f'{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k_0 = k_0'{:}K_0 \quad \Gamma \vdash_\Sigma K_0 \leq_c K}{\Gamma \vdash_\Sigma f(k_0) = f'(k_0'){:}[c(k_0)/x]K'}$$

Coercive Definition

(CD)
$$\frac{\Gamma \vdash_\Sigma f{:}(x{:}K)K' \quad \Gamma \vdash_\Sigma k_0{:}K_0 \quad \Gamma \vdash_\Sigma K_0 \leq_c K}{\Gamma \vdash_\Sigma f(k_0) = f(c(k_0)){:}[c(k_0)/x]K'}$$

Figure C.6: The coercive application and definition rules in $T_{S,\leq}$ and $\Pi_{S,\leq}$

Appendix D

Inference Rules for $LF^{;}$, $T[\mathcal{C}]^{;}_{0K}$ and $T[\mathcal{C}]^{;}$

In this appendix I list the rules of the system $LF^{;}$, $T[\mathcal{C}]^{;}_{0K}$ and $T[\mathcal{C}]^{;}$.

Validity of Signature/Contexts, Assumptions

$$\frac{}{\vdash \langle\rangle} \qquad \frac{\Sigma;\langle\rangle \vdash K\,kind \quad c \notin dom(\Sigma)}{\vdash \Sigma, c{:}K} \qquad \frac{\vdash \Sigma, c{:}K, \Sigma'; \Gamma}{\Sigma, c{:}K, \Sigma'; \Gamma \vdash c{:}K}$$

$$\frac{\vdash \Sigma}{\vdash \Sigma;\langle\rangle} \qquad \frac{\Sigma;\Gamma \vdash K\,kind \quad x \notin dom(\Sigma) \cup dom(\Gamma)}{\vdash \Sigma;\Gamma, x{:}K} \qquad \frac{\vdash \Sigma;\Gamma, x{:}K, \Gamma'}{\Sigma;\Gamma, x{:}K, \Gamma' \vdash x{:}K}$$

Weakening

$$\frac{\Sigma, \Sigma';\Gamma \vdash J \quad \Sigma;\langle\rangle \vdash K\ kind \quad c \notin dom(\Sigma, \Sigma')}{\Sigma,\ c{:}K,\ \Sigma';\Gamma \vdash J}$$

$$\frac{\Sigma;\Gamma, \Gamma' \vdash J \quad \Sigma;\Gamma \vdash K\ kind \quad x \notin dom(\Gamma, \Gamma')}{\Sigma;\Gamma, x{:}K, \Gamma' \vdash J}$$

Equality Rules

$$\frac{\Sigma;\Gamma \vdash K\ \ kind}{\Sigma;\Gamma \vdash K = K} \qquad \frac{\Sigma;\Gamma \vdash K = K'}{\Sigma;\Gamma \vdash K' = K} \qquad \frac{\Sigma;\Gamma \vdash K = K' \quad \Sigma;\Gamma \vdash K' = K''}{\Sigma;\Gamma \vdash K = K''}$$

$$\frac{\Sigma;\Gamma \vdash k{:}K}{\Sigma;\Gamma \vdash k = k{:}K} \qquad \frac{\Sigma;\Gamma \vdash k = k'{:}K}{\Sigma;\Gamma \vdash k' = k{:}K} \qquad \frac{\Sigma;\Gamma \vdash k = k'{:}K \quad \Sigma;\Gamma \vdash k' = k''{:}K}{\Sigma;\Gamma \vdash k = k''{:}K}$$

$$\frac{\Sigma;\Gamma \vdash k{:}K \quad \Sigma;\Gamma \vdash K = K'}{\Sigma;\Gamma \vdash k{:}K'} \qquad \frac{\Sigma;\Gamma \vdash k = k'{:}K \quad \Sigma;\Gamma \vdash K = K'}{\Sigma;\Gamma \vdash k = k'{:}K'}$$

Context Replacement

$$\frac{\Sigma_0, c{:}L, \Sigma_1;\Gamma \vdash J \quad \Sigma_0 \vdash L = L'}{\Sigma_0, c{:}L', \Sigma_1;\Gamma \vdash J} \qquad \frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash J \quad \Sigma;\Gamma_0 \vdash K = K'}{\Sigma;\Gamma_0, x{:}K', \Gamma_1 \vdash J}$$

Substitution Rules

$$\frac{\vdash \Sigma;\Gamma_0, x{:}K, \Gamma_1 \quad \Sigma;\Gamma_0 \vdash k{:}K}{\vdash \Sigma;\Gamma_0, [k/x]\Gamma_1}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash K'\ \ kind \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K'\ \ kind} \qquad \frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash L = L' \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]L = [k/x]L'}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash k'{:}K' \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]k'{:}[k/x]K'} \qquad \frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash l = l'{:}K' \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]l = [k/x]l'{:}[k/x]K'}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash K'\ \ kind \quad \Sigma;\Gamma_0 \vdash k = k'{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K' = [k'/x]K'}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash l{:}K' \quad \Sigma;\Gamma_0 \vdash k = k'{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]l = [k'/x]l{:}[k/x]K'}$$

Dependent Product Kinds

$$\frac{\Sigma;\Gamma \vdash K\,kind \quad \Sigma;\Gamma, x{:}K \vdash K'\,kind}{\Sigma;\Gamma \vdash (x{:}K)K'\,kind} \qquad \frac{\Sigma;\Gamma \vdash K_1 = K_2 \quad \Sigma;\Gamma, x{:}K_1 \vdash K_1' = K_2'}{\Sigma;\Gamma \vdash (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Sigma;\Gamma, x{:}K \vdash y{:}K'}{\Sigma;\Gamma \vdash [x{:}K]y{:}(x{:}K)K'} \qquad \frac{\Sigma;\Gamma \vdash K_1 = K_2 \quad \Sigma;\Gamma, x{:}K_1 \vdash k_1 = k_2{:}K}{\Sigma;\Gamma \vdash [x{:}K_1]k_1 = [x{:}K_2]k_2{:}(x{:}K_1)K}$$

$$\frac{\Sigma;\Gamma \vdash f{:}(x{:}K)K' \quad \Sigma;\Gamma \vdash k{:}K}{\Sigma;\Gamma \vdash f(k){:}[k/x]K'} \qquad \frac{\Sigma;\Gamma \vdash f = f'{:}(x{:}K)K' \quad \Sigma;\Gamma \vdash k_1 = k_2{:}K}{\Sigma;\Gamma \vdash f(k_1) = f'(k_2){:}[k_1/x]K'}$$

$$\frac{\Sigma;\Gamma, x{:}K \vdash k'{:}K' \quad \Sigma;\Gamma \vdash k{:}K}{\Sigma;\Gamma \vdash ([x{:}K]k')(k) = [k/x]k'{:}[k/x]K'} \qquad \frac{\Sigma;\Gamma \vdash f{:}(x{:}K)K' \quad x \notin FV(f)}{\Sigma;\Gamma \vdash [x{:}K]f(x) = f{:}(x{:}K)K'}$$

The kind Type

$$\frac{\vdash \Sigma;\Gamma}{\Sigma;\Gamma \vdash \mathit{Type}\ \ kind} \qquad \frac{\Sigma;\Gamma \vdash A{:}\mathit{Type}}{\Sigma;\Gamma \vdash El(A)\ \ kind} \qquad \frac{\Sigma;\Gamma \vdash A = B{:}\mathit{Type}}{\Sigma;\Gamma \vdash El(A) = El(B)}$$

Figure D.1: Logical Framework Inference Rules for $LF^{;}$, $T[\mathcal{C}]^{;}_{0K}$ and $T[\mathcal{C}]^{;}$

Subtyping Rules

$$\frac{\Sigma;\Gamma \vdash A \leq_c B \in \mathcal{C}}{\Sigma;\Gamma \vdash A \leq_c B}$$

Congruence

$$\frac{\Sigma;\Gamma \vdash A \leq_c B : Type \quad \Sigma;\Gamma \vdash A = A' : Type \quad \Sigma;\Gamma \vdash B = B' : Type \quad \Sigma;\Gamma \vdash c = c' : (A)B}{\Sigma;\Gamma \vdash A' \leq_{c'} B' : Type}$$

Transitivity

$$\frac{\Sigma;\Gamma \vdash A \leq_c A' : Type \quad \Sigma;\Gamma \vdash A' \leq_{c'} A'' : Type}{\Sigma;\Gamma \vdash A \leq_{c' \circ c} A'' : Type}$$

Weakening

$$\frac{\Sigma,\Sigma';\Gamma \vdash A \leq_d B : Type \quad \Sigma \vdash K\ kind}{\Sigma,\ c{:}K,\ \Sigma';\Gamma \vdash A \leq_d B : Type} \quad (c \notin dom(\Sigma,\Sigma'))$$

$$\frac{\Sigma;\Gamma,\Gamma' \vdash A \leq_d B : Type \quad \Sigma;\Gamma \vdash K\ kind}{\Sigma;\Gamma,x{:}K,\Gamma' \vdash A \leq_d B : Type} \quad (x \notin dom(\Gamma,\Gamma'))$$

Context Replacement

$$\frac{\Sigma_0,c{:}L,\Sigma_1;\Gamma \vdash A \leq_c B \quad \Sigma_0 \vdash L = L'}{\Sigma_0,c{:}L',\Sigma_1;\Gamma \vdash A \leq_c B} \qquad \frac{\Sigma;\Gamma_0,x{:}K,\Gamma_1 \vdash A \leq_c B \quad \Sigma;\Gamma_0 \vdash K = K'}{\Sigma;\Gamma_0,x{:}K',\Gamma_1 \vdash A \leq_c B}$$

Substitution

$$\frac{\Sigma;\Gamma_0,x{:}K,\Gamma_1 \vdash A \leq_c B \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0,[k/x]\Gamma_1 \vdash [k/x]A \leq_{[k/x]c} [k/x]B}$$

Identity Coercion

$$\frac{\Sigma;\Gamma \vdash A{:}\,Type}{\Sigma;\Gamma \vdash A \leq_{[x:A]x} A{:}\,Type}$$

Figure D.2: Subtyping Rules for , $T[\mathcal{C}]^{;}_{0K}$ and $T[\mathcal{C}]^{;}$ (1)

Basic Subkinding Rule and Identity

$$\frac{\Sigma;\Gamma \vdash A \leq_c B{:}\mathit{Type}}{\Sigma;\Gamma \vdash \mathit{El}(A) \leq_c \mathit{El}(B)} \qquad \frac{\Sigma;\Gamma \vdash K\ \mathit{kind}}{\Sigma;\Gamma \vdash K \leq_{[x:K]x} K}$$

Structural Subkinding Rules

$$\frac{\Sigma;\Gamma \vdash K_1 \leq_c K_2 \quad \Sigma;\Gamma \vdash K_1 = K_1' \quad \Sigma;\Gamma \vdash K_2 = K_2' \quad \Sigma;\Gamma \vdash c = c'{:}(K_1)K_2}{\Sigma;\Gamma \vdash K_1' \leq_{c'} K_2'}$$

$$\frac{\Sigma;\Gamma \vdash K \leq_c K' \quad \Sigma;\Gamma \vdash K' \leq_{c'} K''}{\Sigma;\Gamma \vdash K \leq_{c' \circ c} K''}$$

$$\frac{\Sigma,\Sigma';\Gamma \vdash K \leq_d K' \quad \Sigma;\langle\rangle \vdash K_0\ \mathit{kind}}{\Sigma, c{:}K_0, \Sigma';\Gamma \vdash K \leq_d K'} \quad (c \notin \mathit{dom}(\Sigma,\Sigma'))$$

$$\frac{\Sigma;\Gamma,\Gamma' \vdash K \leq_d K' \quad \Sigma;\Gamma \vdash K_0\ \mathit{kind}}{\Sigma;\Gamma, x{:}K_0, \Gamma' \vdash K \leq_d K'} \quad (x \notin \mathit{dom}(\Gamma,\Gamma'))$$

$$\frac{\Sigma_0, c{:}L, \Sigma_1;\Gamma \vdash K \leq_d K' \quad \Sigma_0;\langle\rangle \vdash L = L'}{\Sigma_0, c{:}L', \Sigma_1;\Gamma \vdash K \leq_d K'}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash L \leq_d L' \quad \Sigma;\Gamma_0 \vdash K = K'}{\Sigma;\Gamma_0, x{:}K', \Gamma_1 \vdash L \leq_d L'}$$

$$\frac{\Sigma;\Gamma_0, x{:}K, \Gamma_1 \vdash K_1 \leq_c K_2 \quad \Sigma;\Gamma_0 \vdash k{:}K}{\Sigma;\Gamma_0, [k/x]\Gamma_1 \vdash [k/x]K_1 \leq_{[k/x]c} [k/x]K_2}$$

Subkinding for Dependent Product Kind

$$\frac{\Sigma;\Gamma \vdash K_1' \leq_{c_1} K_1\ \ \Sigma;\Gamma, x{:}K_1 \vdash K_2\ \mathit{kind}\ \ \Sigma;\Gamma, x'{:}K_1' \vdash K_2'\ \mathit{kind}\ \ \Sigma;\Gamma, x{:}K_1 \vdash [c_1(x')/x]K_2 \leq_{c_2} K_2'}{\Sigma;\Gamma \vdash (x{:}K_1)K_2 \leq_{[f:(x:K_1)K_2][x':K_1']c_2(f(c_1(x')))} (x{:}K_1')K_2'}$$

Figure D.3: Subkinding Rules for $T[\mathcal{C}]^{;}_{0K}$ and $T[\mathcal{C}]^{;}$ (2)

Coercive Application

$$(CA_1) \qquad \frac{\Sigma;\Gamma \vdash f{:}(x{:}K)K' \quad \Sigma;\Gamma \vdash k_0{:}K_0 \quad \Sigma;\Gamma \vdash K_0 \leq_c K}{\Sigma;\Gamma \vdash f(k_0){:}[c(k_0)/x]K'}$$

$$(CA_2) \qquad \frac{\Sigma;\Gamma \vdash f = f'{:}(x{:}K)K' \quad \Sigma;\Gamma \vdash k_0 = k_0'{:}K_0 \quad \Sigma;\Gamma \vdash K_0 \leq_c K}{\Sigma;\Gamma \vdash f(k_0) = f'(k_0'){:}[c(k_0)/x]K'}$$

Coercive Definition

$$(CD) \qquad \frac{\Sigma;\Gamma \vdash f{:}(x{:}K)K' \quad \Sigma;\Gamma \vdash k_0{:}K_0 \quad \Sigma;\Gamma \vdash K_0 \leq_c K}{\Sigma;\Gamma \vdash f(k_0) = f(c(k_0)){:}[c(k_0)/x]K'}$$

Figure D.4: The coercive application and definition rules in $T[\mathcal{C}]^{;}$

Appendix E

Inference rules for $\mathbb{LF}$ and $\Pi_{\leq}$

In this appendix I list the rules for the logical framework $\mathbb{LF}$, Π - type in this logical framework and subtyping rules. All these rules form the rules for system $\Pi_{\leq}$.

Validity of Signature/Contexts, Assumptions

$$\frac{}{\Vdash \langle\rangle} \qquad \frac{\Gamma \Vdash K\,\mathit{kind} \quad x \notin \mathit{dom}(\Gamma)}{\Vdash \Gamma, x{:}K} \qquad \frac{\Vdash \Gamma, x{:}K, \Gamma'}{\Gamma, x{:}K, \Gamma' \Vdash x{:}K}$$

Equality Rules

$$\frac{\Gamma \Vdash K \;\; \mathit{kind}}{\Gamma \Vdash K = K} \qquad \frac{\Gamma \Vdash K = K'}{\Gamma \Vdash K' = K} \qquad \frac{\Gamma \Vdash K = K' \quad \Gamma \Vdash K' = K''}{\Gamma \Vdash K = K''}$$

$$\frac{\Gamma \Vdash k{:}K}{\Gamma \Vdash k = k{:}K} \qquad \frac{\Gamma \Vdash k = k'{:}K}{\Gamma \Vdash k' = k{:}K} \qquad \frac{\Gamma \Vdash k = k'{:}K \quad \Gamma \Vdash k' = k''{:}K}{\Gamma \Vdash k = k''{:}K}$$

$$\frac{\Gamma \Vdash k{:}K \quad \Gamma \Vdash K = K'}{\Gamma \Vdash k{:}K'} \qquad \frac{\Gamma \Vdash k = k'{:}K \quad \Gamma \Vdash K = K'}{\Gamma \Vdash k = k'{:}K'}$$

Substitution Rules

$$\frac{\Vdash \Gamma_0, x{:}K, \Gamma_1 \quad \Gamma_0 \Vdash k{:}K}{\Vdash \Gamma_0, [k/x]\Gamma_1}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash K' \;\; \mathit{kind} \quad \Gamma_0 \Vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]K' \;\; \mathit{kind}} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash L = L' \quad \Gamma_0 \Vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]L = [k/x]L'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash k'{:}K' \quad \Gamma_0 \Vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash l = l'{:}K' \quad \Gamma_0 \Vdash k{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]l = [k/x]l'{:}[k/x]K'}$$

$$\frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash K' \;\; \mathit{kind} \quad \Gamma_0 \Vdash k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]K' = [k'/x]K'} \qquad \frac{\Gamma_0, x{:}K, \Gamma_1 \Vdash l{:}K' \quad \Gamma_0 \Vdash k = k'{:}K}{\Gamma_0, [k/x]\Gamma_1 \Vdash [k/x]l = [k'/x]l{:}[k/x]K'}$$

Dependent Product Kinds

$$\frac{\Gamma \Vdash K\,\mathit{kind} \quad \Gamma, x{:}K \Vdash K'\,\mathit{kind}}{\Gamma \Vdash (x{:}K)K'\,\mathit{kind}} \qquad \frac{\Gamma \Vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \Vdash K_1' = K_2'}{\Gamma \Vdash (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$\frac{\Gamma, x{:}K \Vdash y{:}K'}{\Gamma \Vdash [x{:}K]y{:}(x{:}K)K'} \qquad \frac{\Gamma \Vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \Vdash k_1 = k_2{:}K}{\Gamma \Vdash [x{:}K_1]k_1 = [x{:}K_2]k_2{:}(x{:}K_1)K}$$

$$\frac{\Gamma \Vdash f{:}(x{:}K)K' \quad \Gamma \Vdash k{:}K}{\Gamma \Vdash f(k){:}[k/x]K'} \qquad \frac{\Gamma \Vdash f = f'{:}(x{:}K)K' \quad \Gamma \Vdash k_1 = k_2{:}K}{\Gamma \Vdash f(k_1) = f'(k_2){:}[k_1/x]K'}$$

$$\frac{\Gamma, x{:}K \Vdash k'{:}K' \quad \Gamma \Vdash k{:}K}{\Gamma \Vdash ([x{:}K]k')(k) = [k/x]k'{:}[k/x]K'} \qquad \frac{\Gamma \Vdash f{:}(x{:}K)K' \quad x \notin FV(f)}{\Gamma \Vdash [x{:}K]f(x) = f{:}(x{:}K)K'}$$

The kind Type

$$\frac{\Vdash \Gamma}{\Gamma \Vdash \mathit{Type} \;\; \mathit{kind}} \qquad \frac{\Gamma \Vdash A{:}\mathit{Type}}{\Gamma \Vdash \mathit{El}(A) \;\; \mathit{kind}} \qquad \frac{\Gamma \Vdash A = B{:}\mathit{Type}}{\Gamma \Vdash \mathit{El}(A) = \mathit{El}(B)}$$

Figure E.1: Logical Framework Rules for $\mathbb{LF}$ and $\Pi_{\leq}$

$$\frac{\Gamma \Vdash A : \mathit{Type} \quad \Gamma, x{:}A \Vdash B(x) : \mathit{Type}}{\Gamma \Vdash \Pi(A, B) : \mathit{Type}}$$

$$\frac{\Gamma \Vdash A : \mathit{Type} \quad \Gamma \Vdash B : (A)\,\mathit{Type} \quad \Gamma \Vdash f : (x{:}A)B(x)}{\Gamma \Vdash \lambda(A, B, f) : \Pi(A, B)}$$

$$\frac{\Gamma \Vdash g : \Pi(A, B) \quad \Gamma \Vdash a : A}{\Gamma \Vdash \mathit{app}(A, B, g, a) : B(a)}$$

$$\frac{\begin{array}{c}\Gamma \Vdash A : \mathit{Type} \quad \Gamma \Vdash B : (A)\,\mathit{Type} \\ \Gamma \Vdash f : (x{:}A)B(x) \quad \Gamma \Vdash a : A\end{array}}{\Gamma \Vdash \mathit{app}(A, B, \lambda(A, B, f), a) = f(a) : B(a)}$$

Figure E.2: Inference Rules for Π - types specified in $\mathbb{LF}$

General Subtyping Rules

$$\frac{\Gamma \Vdash K = K'}{\Gamma \Vdash K \leq K'} \quad \frac{\Gamma \Vdash K \leq K' \quad \Gamma \Vdash K' \leq K''}{\Gamma \Vdash K \leq K''}$$

$$\frac{\Gamma \Vdash A = B\colon Type}{\Gamma \Vdash A \leq B\colon Type}$$

$$\frac{\Gamma \Vdash A \leq B\colon Type \quad \Gamma \Vdash B \leq C\colon Type}{\Gamma \Vdash A \leq C\colon Type}$$

Subtyping in Contexts

$$\frac{\Gamma \Vdash A\colon Type \quad \alpha \notin FV(\Gamma)}{\Gamma, \alpha \leq A \ \ valid} \quad \frac{\Gamma, \alpha \leq A, \Gamma' \ \ valid}{\Gamma, \alpha \leq A, \Gamma' \Vdash \alpha\colon Type} \quad \frac{\Gamma, \alpha \leq A, \Gamma' \ \ valid}{\Gamma, \alpha \leq A, \Gamma' \Vdash \alpha \leq A\colon Type}$$

Type Lifting and Subtyping

$$\frac{\Gamma \Vdash A \leq B\colon Type}{\Gamma \Vdash El(A) \leq El(B)} \quad \frac{\Gamma \Vdash k{:}K \quad \Gamma \Vdash K \leq K'}{\Gamma \Vdash k{:}K'} \quad \frac{\Gamma \Vdash k = k'{:}K \quad \Gamma \Vdash K \leq K'}{\Gamma \Vdash k = k'{:}K'}$$

Dependent Product

$$\frac{\begin{array}{c}\Gamma \Vdash \Pi(A,B)\colon Type \quad \Gamma \Vdash \Pi(A',B')\colon Type \\ \Gamma \Vdash A' \leq A\colon Type \quad \Gamma, x{:}A' \Vdash B \leq B'\colon Type\end{array}}{\Gamma \Vdash \Pi(A,B) \leq \Pi(A',B')\colon Type}$$

Figure E.3: Subtyping Rules for $\Pi_{\leq}$