Definitional Extension in Type Theory

Tao Xue

School of Computer Science, McGill University, Montreal, Canada
xuet.cn@hotmail.com

Abstract

When we extend a type system, the relation between the original system and its extension is an important issue we want to understand. Conservative extension is a traditional relation we study with. But in some cases, like coercive subtyping, it is not strong enough to capture all the properties, more powerful relation between the systems is required. We bring the idea definitional extension from mathematical logic into type theory. In this paper, we study the notion of definitional extension for type theories and explicate its use, both informally and formally, in the context of coercive subtyping.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Language

Keywords and phrases conservative extension, definitional extension, subtype, coercive subtyping

Digital Object Identifier 10.4230/LIPIcs.TYPES.2013.251

1 Introduction

In the studies of type theory, sometimes we extend a type system with some notions and rules. We are interested in what power the extension systems can bring to us, and we also want to know the relations between the systems. Understanding the relations between the systems tells us some of the properties the new system should hold. The most common property we always think of is conservativity, or put in anther way, whether the extension is a *conservative extension*. For example, Hofmann showed the conservativity of extensional type theory over intensional type theory with extensional concepts added [4]. Informally, conservativity means that the new system maybe more convenient than the original system but it cannot prove any new theorem within the old language. It requires that all the theorems in the old language, which are provable in new system, are also provable in the old system.

Subtypes are introduced into type theory and studied in many works [1, 2, 14, 15]. Coercive subtyping [7] is one approach of studying subtype in type theory. Unlike the traditional way of dealing subtype with subsumption rule

$$\frac{a : A \qquad A \leq B}{a : B}$$

which is very common in the study of functional programming languages [13], coercive subtyping is an abbreviation mechanism. We consider a unique coercion c between two types A and B, written as $A <_c B$. Intuitively, for every place we require a term of type B, we can use a term a of type A instead, and it is just an abbreviation of using the term $c(a)$. This simple mechanism is quite powerful, one recent use is in the study of linguistic semantics [9, 19].

Since we take coercive subtyping as an abbreviation mechanism, we don't want it to increase any power of the existing system. Soloviev and Luo [17] studied the relationship between a type system and its coercive subtyping extension and called it "conservativity". In

19th International Conference on Types for Proofs and Programs (TYPES 2013).
Editors: Ralph Matthes and Aleksy Schubert; pp. 251–269
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

fact, the relationship is not quite the same as the traditional notion of conservative extension and it turns out that it can better be characterized as an *definitional extension* in a more general sense. In this paper, we will give a definition of this notion of definitional extension and explicate it, both informally with a simple example and formally for coercive subtyping.

Soloviev and Luo's previous work [17] was based on a notion of basic subtyping rules which turns out to be unnecessarily general. It does not exclude certain "bad" subtyping rules which cannot be used normally but can be applied once we introduce coercive subtyping. This would destroy the consistency of the whole system. Recently, we fix the problem by considering coercion sets rather than coercion rules and, furthermore, the latter can be captured by the former [11, 18]. In this paper, our treatment of coercive subtyping is based on this new framework.

The paper goes in the following way. We give the motivation of introducing definitional extension in Section 2 by showing that coercion mechanism cannot be expected as a conservative extension. In Section 3, we present a definition of conservative extension and definitional extension in type theory. We use a simplified example to demonstrate the relation between a system and its coercion extension in Section 4 and give a sketch on the full study of the relation in Section 5.

2 Motivation: coercive subtyping

Coercive subtyping [7] is an approach to introducing subtypes into type theory and it considers subtyping by means of *abbreviations*.

The basic idea of coercive subtyping is that, when we consider A as a subtype of B, we choose a unique function c from A to B, and declare c to be a *coercion*, written as $A <_c B$. Intuitively, the idea means anywhere we need to write an object of type B, we can use an object a of type A instead. Actually in the context, the object a is to be seen as an *abbreviation* for the object $c(a) : B$. More precisely, if we have f from B to C, then f can be applied to any object a of type A to form $f(a)$ of type C, which is definitionally equal to $f(c(a))$. We can consider $f(a)$ to be an abbreviation for $f(c(a))$, with coercion c being inserted to fill the *gap* between f and a. The idea above could be captured by means of the following formal rules:

$$\frac{f\colon B \to C \quad a\colon A \quad A <_c B}{f(a)\colon C} \qquad\qquad \frac{f\colon B \to C \quad a\colon A \quad A <_c B}{f(a) = f(c(a))\colon C}$$

As an extension of a type theory, coercive subtyping is based on the idea that subtyping is abbreviation. On the one hand, it should not increase any expressive power of the original system. On the other hand, coercions can always be correctly inserted to obtain the abbreviated expressions as long as the basic coercions are coherent [1].

In the study coercive subtyping, Soloviev and Luo tried to think it be a conservative extension [17]. But we find that conservativity is not accurate to capture the relation. In the expressions of coercive subtyping, there are "gaps" introduced by the coercions. Given $f : B \to C$ and $a : A$, although $f(a) : C$ is well-formed with coercive subtyping $A <_c B$, we can still image that there is a "gap" in $f(a)$ between f and a. As mentioned above, we want to show that all the "gaps" in the expressions caused by coercions can be correctly inserted.

---

[1] Informally, coherence in coercive subtyping means there is unique coercion between two different types, further details are discussed in Section 5.

For example, let's consider types $Nat = 0|succ(Nat)$, $Bool = true|false$ and coercion $Bool <_c Nat$. With coercive subtyping, we can have terms:

$$succ(true), succ(false), succ(succ(true)), succ(succ(false)), \ldots$$

As we have emphasised that coercive subtyping is just abbreviation, these terms should actually be equivalent to the following terms:

$$succ(c(true)), succ(c(false)), succ(succ(c(true))), succ(succ(c(false))), \ldots$$

Such equivalence is the most important property of the extension with coercive subtyping. We want to show that all the judgements or derivations in the system with coercive subtyping can be translated into the equivalent ones in the original type system. "conservative extension" is not enough for our use, it only talks about whether the derivable judgements in new system are still derivable in the original one, it doesn't ask for such equivalence connection. We find the idea of "definitional extension" in first-order logic theories contains a translation between the formulas of the theories. Hence, we think that such notion of definitional extension is a suitable option to describe the relation between a type system and its coercive subtyping extension.

3 Conservative extension and definitional extension

To build a definition for the definitional extension, we should give definitions for the equivalence between judgements and equivalence between derivations first. Such definitions depend on the forms of judgements. In this paper, we will consider the type systems formalised in Luo's logical framework[2] [6]. For other cases, we should be able to consider them in a similar way.

3.1 Luo's logical framework

Luo's logical framework [6] is a typed version of Martin-Löf's logical framework [15]. In Luo's logical framework, the functional abstractions of the form $(x)k$ in Martin-Löf's logical framework are replaced by the typed form $[x : K]k$. We will simply call it LF in the rest part of this paper.

LF is a type system with terms of the following forms:

$$\mathbf{Type},\ El(A),\ (x:K)K',\ [x:K]k',\ f(k)$$

The kind **Type** denotes the conceptual universe of types; $El(A)$ denotes the kind of objects of type A; $(x{:}\,K)K'$ denotes a dependent product; $[x{:}\,K]k'$ denotes an abstraction; and $f(k)$ denotes an application. The free occurrences of the variable x in K' and k' are bound by the binding operators $(x : K)$ and $[x : K]$. There are five forms of judgements in LF:

- $\Gamma \vdash$ **valid**, which asserts that Γ is a valid context.
- $\Gamma \vdash K$ **kind**, which asserts that K is a valid kind.
- $\Gamma \vdash k : K$, which asserts that k is an object of kind K.
- $\Gamma \vdash k = k' : K$, which asserts that k and k' are equal objects of kind K.
- $\Gamma \vdash K = K'$, which asserts that K and K' are two equal kinds.

Figure 7 shows the LF rules. It contains the rules for context validity and assumptions, the general equality rules, the type equality rules, the substitution rules, the rules for kind Type and the rules for dependent product kinds.

[2] It is different from the Edinburgh Logical Framework [3].

3.2 Conservative extension

In mathematical logic, when we say a logical theory S_2 is an *extension* of a theory S_1, it means that the syntax of S_2 includes all the syntax of S_1 and every theorem of S_1 is a theorem of S_2. We call S_2 a *conservative extension* of S_1, if S_2 is an extension of S_1 and we have a further condition that any theorem of S_2 in the language of S_1 is a theorem in S_1.

When we talk about such extensions, it is important to point out that the syntax of S_2 contains all the syntax of S_1. We can have two labels of the theorems, one is *proposable*, another is *provable*. Proposable means the theorem can be written down in the language, not necessary be proved. Provable means the theorem can not only be written down but also be proved. In conservative extension, we don't care those theorems which are proposable in S_2 but not proposable in S_1. However, we will see later that in definitional extension we need to think of them.

We can consider the idea similarly in type theory. Instead of thinking of the theorems, we would like to think of the judgements. If a judgement can be derived through the rules in the system, we call it a *derivable judgement*. We say type system T_2 which includes all the syntax of system T_1 is a *conservative extension* of T_1 , if for any proposable sequent (judgement) t of the system T_1, t is derivable in T_2 implies that t is derivable in T_1. If a sequent is not proposable in T_1 (only proposable in T_2), its derivability does not matter.

More precisely, let's use $\vdash_T$ for the derivable judgements in system T. T_2 is an extension of T_1 requires that, T_2 includes all the syntax of T_1 and for any judgement $\Gamma \vdash \Sigma$ in T_1(it may not be derivable):

$$\Gamma \vdash_{T_1} \Sigma \quad \Rightarrow \quad \Gamma \vdash_{T_2} \Sigma$$

For such an extension to be conservative, we also require:

$$\Gamma \vdash_{T_2} \Sigma \quad \Rightarrow \quad \Gamma \vdash_{T_1} \Sigma$$

▶ **Definition 1** (conservative extension). Type theory T_2 is a conservative extension of T_1, if T_2 includes all the syntax of T_1 and for any proposable judgement J in T_1, there's a derivation of J in T_1 if and only if there's a derivation of J in T_2.

3.3 Definitional extension

Sometimes, conservative extension is not powerful enough to describe the relation between the systems. In some cases, like the study of coercive subtyping [11], we not only want to show the conservativity, but also want the systems to keep a stronger relation. We want the formulas, judgements or derivations in one system could be translated to corresponding ones in another system. Definitional extension describes such kind of relation.

Traditionally, the notion of *definitional extension* was formulated for first-order logical theories [5]: a first-order theory is a definitional extension of another if the former is a conservative extension of the latter and any formula in the former is logically equivalent to its translation in the latter. More precisely, a definitional extension S' of a first-order theory S is obtained by successive introductions of relations(or functions) in such a way that, for example, for an n-ary relation R, the following *defining axiom* of R is added:

$$\forall x_1 ... \forall x_n.\ R(x_1, ..., x_n) \iff \phi(x_1, ..., x_n),$$

where $\phi(x_1, ..., x_n)$ is a formula in S.

For such a definitional extension S', we have:

- for any formula ψ in S', $\psi \iff \psi^*$ is provable in S', where ψ^* is the formula in S obtained from ψ by replacing any occurrence of $R(t_1, ..., t_n)$ by $\phi(t_1, ..., t_n)$ (with necessary changes of bound variables).
- S' is a conservative extension of S.

Taking the idea of definitional extension, especially the translation between formulas, we are going to consider a similar relation in type theory. The notion of definitonal extension in first-order logic is characterised in terms of translation on *formulas*. In our type theory, we have at least two options to present the translation on: *judgements* and *derivations*. Intuitively, derivable judgements and derivations are very close related to each other. In analogy to the formulas in logic, it sounds even more natural to use judgements in type theory. However, we will choose derivations to formalise our definition. Let's consider the type systems with coercive subtyping. Translating a judgement with coercive subtyping into a judgement without coercive subtyping requires us to point out all the "gaps" introduced by coercion in the judgement. They are not simply marked in the syntax, and due to the congruence rules of subtyping, the insertion might not be syntactically unique. We have to look up the derivations to find the coercions out. More generally, in intensional type theories, the non-syntax-directed use of the conversion rule makes the connection between the judgement and derivation non-structural. When we have the mechanisms like coercion, the choice of rules by which to refine a judgement becomes no more free. Based on these reasons, it is necessary to give the definition in term of derivations.

Before giving a formal definition of definitional extension, we need to define the equivalence between derivations first. The equivalence between the derivations can be defined by the equivalence between derivable judgements and the equivalence between the judgements intuitively means that the corresponding parts of two judgements are equal formulas. In LF, the judgements are of form:

$$\Gamma \vdash \textbf{valid}, \ \Gamma \vdash K \ \textbf{kind}, \ \Gamma \vdash k : K, \ \Gamma \vdash k_1 = k_2 : K \ \ \textit{and} \ \ \Gamma \vdash K_1 = K_2$$

Hence, we can define the equivalence between the judgements in the following way:

▶ **Notation 2.** In a type system S specified in LF, let Γ_1 and Γ_2 be

$$\Gamma_1 \equiv x_1 : K_1, x_2 : K_2, \cdots, x_n : K_n$$

$$\Gamma_2 \equiv x_1 : M_1, x_2 : M_2, \cdots, x_n : M_n$$

The equality $\Gamma \vdash \Gamma_1 = \Gamma_2$ is an abbreviation for the following list of n judgements:

$$\begin{array}{rcl} \Gamma & \vdash & K_1 = M_1; \\ \Gamma, \ x_1 : K_1 & \vdash & K_2 = M_2; \\ & \vdots & \\ \Gamma, x_1 : K_1, \cdots, x_{n-1} : K_{n-1} & \vdash & K_n = M_n. \end{array}$$

With the LF rules, we can proof the following propositions of our equality abbreviation in type system S specified in LF. Then, we can use them to define equality between judgements and between derivations in S.

▶ **Proposition 3.** *In a type system S specified in LF.*

1. *If Γ_1 is a valid context, then $\vdash \Gamma_1 = \Gamma_1$.*
2. *If $\Gamma \vdash \Gamma_1 = \Gamma_2$, then $\Gamma \vdash \Gamma_2 = \Gamma_1$.*

3. *If* $\Gamma \vdash \Gamma_1 = \Gamma_2$ *and* $\Gamma \vdash \Gamma_2 = \Gamma_3$*, then* $\Gamma \vdash \Gamma_1 = \Gamma_3$.
4. *If* $\Gamma, \Gamma_1 \vdash J$ *and* $\Gamma \vdash \Gamma_1 = \Gamma_2$ *then* $\Gamma, \Gamma_2 \vdash J$. *(*$J$ *is of form* **valid**, K **kind**, $k\colon K$, $k_1 = k_2\colon K$ *or* $K_1 = K_2$*)*

Proof. See appendix B. ◀

▶ **Definition 4.** (equality between judgements) Let S be a type theory specified in LF. The notion of equality between judgements of the same form in S, notation $J_1 =_s J_2$, is inductively defined as follows:

1. $(\Gamma_1 \vdash \textbf{valid}) =_s (\Gamma_2 \vdash \textbf{valid})$ iff $\vdash \Gamma_1 = \Gamma_2$ is derivable in S.
2. $(\Gamma_1 \vdash K_1\ \textbf{kind}) =_s (\Gamma_2 \vdash K_2\ \textbf{kind})$ iff $\vdash \Gamma_1 = \Gamma_2$ and $\Gamma_1 \vdash K_1 = K_2$ are derivable in S.
3. $(\Gamma_1 \vdash k_1\colon K_1) =_s (\Gamma_2 \vdash k_2\colon K_2)$ iff $\vdash \Gamma_1 = \Gamma_2$, $\Gamma_1 \vdash K_1 = K_2$ and $\Gamma_1 \vdash k_1 = k_2\colon K_1$ are derivable in S.
4. $(\Gamma_1 \vdash K_1 = K_1') =_s (\Gamma_2 \vdash K_2 = K_2')$ iff $\vdash \Gamma_1 = \Gamma_2$, $\Gamma_1 \vdash K_1 = K_2$ and $\Gamma_1 \vdash K_1' = K_2'$ are derivable in S.
5. $(\Gamma_1 \vdash k_1 = k_1'\colon K_1) =_s (\Gamma_2 \vdash k_2 = k_2'\colon K_2)$ iff $\vdash \Gamma_1 = \Gamma_2$, $\Gamma_1 \vdash K_1 = K_2$, $\Gamma_1 \vdash k_1 = k_2\colon K_1$ and $\Gamma_1 \vdash k_1' = k_2'\colon K_1$ are derivable in S.

The equivalence between the derivations can be given as follows:

▶ **Definition 5.** (equality between derivations) Suppose d is a derivation in type system S, let $conc(d)$ denote the conclusion of derivation d. Given two derivations d_1 and d_2, we call d_1 and d_2 equivalent derivations in S and write $d_1 \sim_s d_2$ iff $conc(d_1) =_s conc(d_2)$ in S.

▶ **Theorem 6.** *Let* S *be a type theory specified in LF,* $=_s$ *and* $\sim_s$ *are equivalence relations.*

Proof. Straight with Proposition 3 and LF rules in Figure 7. ◀

When no confusion may occur, We will omit S and simply write $=$ and $\sim$ for the equivalence between judgements and derivations in system S.

▶ **Definition 7.** (definitional extension) We call T_2 is a definitional extension of T_1, if we have:

- for any derivation d in T_2, we can translate d into a corresponding derivation d' in T_1, d and d' are equivalent derivations in T_2.
- T_2 is a conservative extension of T_1.

4 A simple example

In Section 2, we have proposed our motivation of introducing definitional extension: conservative extension is not enough to capture the properties when we extend a system with coercive subtyping. However, we find that coercive subtyping is not a definitional extension either. The reason is that terms like $succ(true)$ are proposable but not derivable in the original system. With the help of coercive subtyping, they are derivable. It doesn't satisfy the definition of conservative extension, hence not definitional extension. To figure out what exactly the relation is, we have to employ some intermediate systems to help us.

The complete description of the relations between a type system, its coercive subtyping extension and intermediate systems is complex and includes some tedious proofs [18]. We will give a sketch of it in the next section. In this section, we try to give an example with coercive subtyping to tell such story in a simple and informal way. Through this trivial looking example, we would like to show the following points: 1) why definitional extension is still not enough (or why we introduce a intermediate system); 2) how to introduce a proper intermediate system; 3) the relations between the systems.

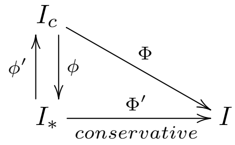

Figure 1 Relations between I_c, I_* and I.

We will consider three systems in the example, a type system I and two of its extensions. I is a very simple type system with only two constant types *Nat* and *Bool*. We extend it into system I_c with one coercion $Bool <_c Nat$. We also introduce system I_* as extension of I with $*$ calculus, such $*$ plays a role of gap holder when we apply the coercion. Through the relations between the judgements of these systems, we can draw a picture for the links between these systems as Figure 1 (definitions of Φ, Φ', ϕ and ϕ' are shown in the later parts of this section).

We will use some informal notions in this example section for the purpose of a simple description. We only have subtyping in the example, while in LF we shift them into subkinding. We will omit the contexts of judgements and use judgements to formalise translations for definitional extension. It is worth pointing out that using judgements for the translations doesn't violate our previous settings with derivations in this example. Because in the syntax of judgements, the applications of *succ* on *true* or *false* indicate the use of coercion application rule in the derivation clearly.

4.1 System I

In I, we only have two basic types *Nat* and *Bool* with their constructors, and a term c of type $Bool \to Nat$:

$Nat\colon Type,\quad 0\colon Nat,\quad succ\colon Nat \to Nat,$

$Bool\colon Type,\quad true\colon Bool,\quad false\colon Bool,\quad c\colon Bool \to Nat$

And, we have the following rules:

$$\frac{f\colon M \to N \quad a\colon M}{f(a) : N} \qquad \frac{a\colon M}{a = a\colon M} \qquad \frac{a_1 = a_2\colon M}{a_2 = a_1\colon M} \qquad \frac{a_1 = a_2\colon M \quad a_2 = a_3\colon M}{a_1 = a_3\colon M}$$

In this system, the judgements are of form:

$$a\colon M \quad \textit{and} \quad a_1 = a_2\colon M$$

We can easily list out all the derivable judgements in I, they can only be of the following cases:

$0\colon Nat,\quad succ\colon Nat \to Nat,\quad succ(...succ(0))\colon Nat,$

$true\colon Bool,\quad false\colon Bool,\quad c\colon Bool \to Nat,\quad c(true)\colon Nat,\quad c(false)\colon Nat,$

$succ(...succ(c(true)))\colon Nat,\quad succ(...succ(c(false))\colon Nat,$

$0 = 0\colon Nat,\quad succ(...succ(0)) = succ(...succ(0))\colon Nat,$

$true = true\colon Bool,\quad false = false\colon Bool,$

$succ = succ\colon Nat \to Nat,\quad c = c\colon Bool \to Nat,$

$succ(...succ(c(true))) = succ(...succ(c(true)))\colon Nat,$

$succ(...succ(c(false))) = succ(...succ(c(false)))\colon Nat$

▶ Remark. For the judgements like $succ(...succ(c(true))) = succ(...succ(c(true)))\colon Nat$, the left and right term of the equal mark have the same number of *succ*. It's the same case for the other similar judgements in rest of this section.

4.2 System I_c

Let's enrich the system I with coercive subtyping. We extend I into system I_c with coercion $Bool <_c Nat$ and coercion application rules:

$$\frac{f\colon B \to C \quad a\colon A \quad A <_c B}{f(a)\colon C} \qquad \frac{f\colon B \to C \quad a\colon A \quad A <_c B}{f(a) = f(c(a))\colon C}$$

The judgements in system I_c are of form[3]:

$$a\colon M \quad and \quad a_1 = a_2\colon M$$

We can get all the derivable judgements in system I_c. Besides all those we have in system I, we can derive the following judgements:

$$succ(...succ(true))\colon Nat,\ succ(...succ(false))\colon Nat,$$
$$succ(...succ(true)) = succ(...succ(true))\colon Nat,$$
$$succ(...succ(false)) = succ(...succ(false))\colon Nat,$$
$$succ(...succ(c(true))) = succ(...succ(true))\colon Nat,$$
$$succ(...succ(c(false))) = succ(...succ(false))\colon Nat,$$
$$succ(...succ(true)) = succ(...succ(c(true)))\colon Nat,$$
$$succ(...succ(false)) = succ(...succ(c(false)))\colon Nat$$

4.3 Relation between I and I_c

Now, let's consider the relation between I and I_c. We want to show that every derivable judgement in I_c is equivalent to a corresponding derivable judgement in I. To achieve this goal, we define a translation Φ from every derivable judgements in system I_c to derivable judgements in I. Φ inserts all the gaps caused by coercion with term c (since we only have one subtyping relation). The definition of Φ is as follows :

1. $\Phi(t) \equiv t$, if the t is the judgement in I,
2. $\Phi(t) \equiv succ(...succ(c(b)) : Nat$, if $t \equiv succ(...succ(b)) : Nat$, b is either *true* or *false*,
3. $\Phi(t) \equiv succ(...succ(c(b))) = succ(...succ(c(b))) : Nat$,
   if $t \equiv succ(...succ(b)) = succ(...succ(b)) : Nat$, b is either *true* or *false*,
4. $\Phi(t) \equiv succ(...succ(c(b) = succ(...succ(c(b))) : Nat$,
   if $t \equiv succ(...succ(b)) = succ(...succ(c(b))) : Nat$, b is either *true* or *false*,
5. $\Phi(t) \equiv succ(...succ(c(b)) = succ(...succ(c(b))) : Nat$,
   if $t \equiv succ(...succ(c(b))) = succ(...succ(b)) : Nat$, b is either *true* or *false*.

It is easy to prove that Φ is total. In order to show the equality between the judgements in I_c and their translations in I, we can prove Φ is holding the following property.

▶ **Proposition 8.** *For any derivable judgement t in system I_c, $\Phi(t)$ and t are equivalent judgements in system I_c*

[3] We do not consider subtyping relation as a judgement in this example section. But in full study of coercive subtyping in LF, we will think them as judgements. See the discussion in Section 5.

Although we have shown certain relation between system I and I_c, it just satisfies the first condition of definitional extension. We cannot say I_c is a definitional extension of I, because definitional extension requires conservativity. Unfortunately, I_c is not a conservative extension of I_c. A simple counter example is that, $succ(true) : Nat$ is a judgement but not derivable in I, however it is derivable in I_c. It doesn't satisfy the definition of conservative extension.

The reason for this problem is that the abbreviation with "gaps" mechanism of coercive subtyping makes such non-well-formed sequences to be well-formed. If we consider an intermediate system with an extra place holder for the "gaps", we may get rid of the problem.

4.4 System I_*

To make a more specific study for the relations, we will introduce another system I_*. Intuitively, I_* means that for any place we want to use a coercion, we insert a symbol $*$ to fill the gap, it equals to the term where the coercion applied. Similarly like I_c, I_* extends system I with the following rules:

$$\frac{f : B \to C \quad a : A \quad A <_c B}{f * a : C} \qquad \frac{f : B \to C \quad a : A \quad A <_c B}{f * a = f(c(a)) : C}$$

In system I_*, the judgements are also of form:

$$a : M \quad \mathit{and} \quad a_1 = a_2 : M$$

We can list all the derivable judgements in system I_* as follows, besides all those in system I:

$$succ(...succ * true) : Nat, \quad succ(...succ * false) : Nat,$$
$$succ(...succ * true) = succ(...succ * true) : Nat,$$
$$succ(...succ * false) = succ(...succ * false) : Nat,$$
$$succ(...succ(c(true)) = succ(...succ * true) : Nat,$$
$$succ(...succ(c(false))) = succ(...succ * true) : Nat,$$
$$succ(...succ * true) = succ(...succ(c(true))) : Nat,$$
$$succ(...succ * true) = succ(...succ(c(false))) : Nat$$

4.5 Relation between I and I_*

It is trivial to show that I_* is a conservative extension of I. Since judgements like $succ * true : Nat$ are not judgements in I, we don't need to consider them, all the other derivable judgements in I_* are exactly the same judgements in I.

▶ **Proposition 9.** *System I_* is a conservative extension of system I.*

Like what we have done for the relation between I_c and I. We can introduce a total translation Φ' from judgements of system I_* to judgements of system I. Intuitively, it substitutes all the appearance of $*$ with our only coercion c:

1. $\Phi'(t) \equiv t$, if the t is a derivable judgement in I,
2. $\Phi'(t) \equiv succ(...succ(c(b)) : Nat$, if $t \equiv succ(...succ * b) : Nat$, b is either $true$ or $false$,
3. $\Phi'(t) \equiv succ(...succ(c(b)) = succ(...succ(c(b))) : Nat$,
   if $t \equiv succ(...succ * b) = succ(...succ * b) : Nat$, b is either $true$ or $false$,
4. $\Phi'(t) \equiv succ(...succ(c(b) = succ(...succ(c(b))) : Nat$,
   if $t \equiv succ(...succ * b) = succ(...succ(c(b))) : Nat$, b is either $true$ or $false$,
5. $\Phi'(t) \equiv succ(...succ(c(b)) = succ(...succ(c(b))) : Nat$,
   if $t \equiv succ(...succ(c(b))) = succ(...succ * b) : Nat$, b is either $true$ or $false$.

Now we have a total translation Φ' from system I_* to system I. Again, it is easy to prove that for every derivable judgement t in system I_*, $\Phi'(t)$ and t are equal judgements in I_*. Together with the conservative property, we can conclude that I_* is a definitional extension of I.

- For any derivable judgement t in I_*, $\Phi'(t)$ is a derivable judgement in I, $\Phi'(t)$ and t are equivalent judgements in I_*.
- I_* is a conservative extension of I.

4.6 Relation between I_c and I_*

Now, let's think of the relation between, I_c and I_*. The rules and judgements are almost the same, only different in symbols. Intuitively, they should be equivalent systems. We can show their equality by introducing two more translations between the systems: ϕ from the judgement of I_c to the judgement of I_*, ϕ' from the judgement of I_* to the judgement of I_c.

- ϕ changes every place of $succ(true)$ or $succ(false)$ in system I_c into term $succ * true$ or $succ * false$.
- ϕ' simply removes every occurrence of $*$ in system I_*.

It's trivial to show that ϕ and ϕ' are total, and easy to prove that I_c and I_* are two equivalent systems by means of :

▶ **Proposition 10.**

- *For every judgement t in I_c, $\phi'(\phi(t)) \equiv t$.*
- *For every judgement t' in I_* $\phi(\phi'(t')) \equiv t'$.*

We can also show that Φ is a composition of Φ' and ϕ:

▶ **Proposition 11.** *For any derivable judgement t in I_c, $\Phi(t) \equiv \Phi'(\phi(t))$*

Finally, we can reach the conclusion for the relations between all these systems: I_c is equivalent to a system I_* which is a definitional extension of I, as shown in the graph previously (Figure 1):

- I_c is an equivalent system of I_*
- I_* is a definitional extension of I:

5 Coercive subtyping in LF

Luo formulated coercive subtyping [7] in his LF [6]. Later we find that the extension took a too general set of coercion rules which may ruin the consistency of the extension system. We solve the problem by reformulating it with some restriction [11, 18]. In this section, we give a sketch of reformulated system and proofs to show the definitional extension, further details could be found in the author's thesis [18].

We will mainly consider the following systems: an original type system T; an extension of system T with coercive subtyping ($T[\mathcal{C}]$); an extension of system T with coercive subtyping and place holder $*$ ($T[\mathcal{C}]^*$); an intermediate system without coercion application rules ($T[\mathcal{C}]_{0K}$).

We introduce coercive subtyping in type level (rules in Figure 2) and then move them into kind level (rules in Figure 3). The symbol $*$ is introduced as a place holder to fill the gaps left by the coercions. We call it $*$-calculus. Following the idea in Section 4, we should be able to show that $T[\mathcal{C}]^*$ is a definitional extension over T. Unfortunately, we can not reach this conclusion yet, because we need to consider the derivations of subtyping and subkinding judgements ($\Gamma \vdash A <_c B : \mathbf{Type}$ or $\Gamma \vdash K <_c K'$). We didn't consider them in the simplified

Base Coercion

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

Congruence

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

Transitivity

$$\frac{\Gamma \vdash A <_{c_1} B : \mathbf{Type} \quad \Gamma \vdash B <_{c_2} C : \mathbf{Type}}{\Gamma \vdash A <_{c_2 \circ c_1} C : \mathbf{Type}}$$

Substitution

$$\frac{\Gamma, x : K, \Gamma' \vdash A <_c B : \mathbf{Type} \qquad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]A <_{[k/x]c} [k/x]B : \mathbf{Type}}$$

Weakening

$$\frac{\Gamma, \Gamma' \vdash A <_c B : \mathbf{Type} \quad \Gamma \vdash K \ \ \mathbf{kind} \quad x \notin FV(\Gamma) \cup FV(\Gamma')}{\Gamma, x : K, \Gamma' \vdash A <_c B : \mathbf{Type}}$$

Context Retyping

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

Figure 2 The structural subtyping rules of $T[\mathcal{C}]_0$.

example in the previous section, there was only one coercion taken as axiom. In a complete description in LF, we have derivations of these subtyping and subkinding judgements, we can hardly match them to any equivalent derivations in T. To fill this gap, we have to involve the intermediate system $T[\mathcal{C}]_{0K}$ into the relations between T, $T[\mathcal{C}]$ and $T[\mathcal{C}]^*$. $T[\mathcal{C}]_{0K}$ extends T as $T[\mathcal{C}]$ but without the coercion application rules (rules in Figure 4). We will show that $T[\mathcal{C}]^*$ is a definitional extension of $T[\mathcal{C}]_{0K}$, $T[\mathcal{C}]_{0k}$ is a conservative extension of T and $T[\mathcal{C}]$ is an equivalent system of $T[\mathcal{C}]^*$.

5.1 System $T[\mathcal{C}]$

Let T be a type system specified in LF such as Martin-Löf's type theory [12] or UTT [6]. With a set $\mathcal{C}$ of coercive subtyping judgements (judgements of form $\Gamma \vdash A <_c B : \mathbf{Type}$), the following basic coercion rules in Figure 2, 3 and coercion application rules in Figure 4, we can extend T into a type system $T[\mathcal{C}]$ with coercive subtyping[4].

5.2 Coherence

Coherence is an important issue in coercive subtyping. Informally, it means there's a unique coercion between two types. To give a formal definition in our structure, we need to introduce an intermediate system $T[\mathcal{C}]_0$.

[4] Rules in Figures 2, 3, 4 and 5 are only the subtyping and subkinding rules. Figure 7 contains the rest LF rules.

Basic subkinding rule

$$\frac{\Gamma \vdash A <_c B : \mathbf{Type}}{\Gamma \vdash El(A) <_c El(B)}$$

Subkinding for dependent product kinds

$$\frac{\Gamma \vdash K_1' <_{c_1} K_1 \qquad \Gamma, x' : K_1' \vdash [c_1(x')/x]K_2 = K_2' \qquad \Gamma, x : K_1 \vdash K_2 \ \mathbf{kind}}{\Gamma \vdash (x : K_1)K_2 <_c (x' : K_1')K_2'}$$

where $c \equiv [f : (x : K_1)K_2][x' : K_1']f(c_1(x'))$;

$$\frac{\Gamma \vdash K_1' = K_1 \qquad \Gamma, x' : K_1' \vdash K_2 <_{c_2} K_2' \qquad \Gamma, x : K_1 \vdash K_2 \ \mathbf{kind}}{\Gamma \vdash (x : K_1)K_2 <_c (x' : K_1')K_2'}$$

where $c \equiv [f : (x : K_1)K_2][x' : K_1']c_2 f(x')$;

$$\frac{\Gamma \vdash K_1' <_{c_1} K_1 \qquad \Gamma, x' : K_1' \vdash [c_1(x')/x]K_2 <_{c_2} K_2' \qquad \Gamma, x : K_1 \vdash K_2 \ \mathbf{kind}}{\Gamma \vdash (x : K_1)K_2 <_c (x' : K_1')K_2'}$$

where $c \equiv [f : (x : K_1)K_2][x' : K_1']c_2(f(c_1(x')))$.

Congruence for subkinding

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

Transitivity for subkinding

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

Substitution for subkinding

$$\frac{\Gamma, x : K, \Gamma' \vdash K_1 <_c K_2 \qquad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K_1 <_{[k/x]c} [k/x]K_2}$$

Weakening for subkinding

$$\frac{\Gamma, \Gamma' \vdash K_1 <_c K_2 \quad \Gamma \vdash K \ \mathbf{kind} \quad x \notin FV(\Gamma) \cup FV(\Gamma')}{\Gamma, x : K, \Gamma' \vdash K_1 <_c K_2}$$

Context Retyping for subkinding

$$\frac{\Gamma, x : K, \Gamma' \vdash K_1 <_c K_2 \quad \Gamma \vdash K = K'}{\Gamma, x : K', \Gamma' \vdash K_1 <_c K_2}$$

Figure 3 The subkinding rules of $T[\mathcal{C}]_{0K}$.

Coercive application rule

$$(CA1)\frac{\Gamma \vdash f : (x:K)K' \qquad \Gamma \vdash k_0 : K_0 \qquad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f(k_0) : [c(k_0)/x]K'}$$

$$(CA2)\frac{\Gamma \vdash f = f' : (x:K)K' \qquad \Gamma \vdash k_0 = k_0' : K_0 \qquad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f(k_0) = f'(k_0') : [c(k_0)/x]K'}$$

Coercive definition rule

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

Figure 4 The coercive application and definition rules of $T[\mathcal{C}]$.

$T[\mathcal{C}]_0$ is a system extending T with set $\mathcal{C}$ of coercion subtyping judgements, subtyping judgements $\Gamma \vdash A <_c B : \mathbf{Type}$ and basic subtyping rules (Figure 2).

▶ **Definition 12.** (coherence) $\mathcal{C}$ is called a coherent set of coercive subtyping judgement, if in $T[\mathcal{C}]_0$ we have:

1. $\Gamma \vdash A <_c B : \mathbf{Type}$ implies $\Gamma \vdash A : \mathbf{Type}$, $\Gamma \vdash B : \mathbf{Type}$, $\Gamma \vdash c : (A)B$ are derivable in T.
2. We cannot derive $\Gamma \vdash A <_c A : \mathbf{Type}$, for any Γ, A, c.
3. $\Gamma \vdash A <_{c_1} B : \mathbf{Type}$ and $\Gamma \vdash A <_{c_2} B : \mathbf{Type}$ imply that $\Gamma \vdash c_1 = c_2 : (A)B$ is derivable in T.

In fact, we can prove that any two coercions between two given kinds are equal in $T[\mathcal{C}]$. Let c and c' be two different coercion from K to K', $K <_c K'$ and $K <_{c'} K'$:

$$\begin{aligned}
\Gamma \vdash c &= [x:K](c(x)) && (\eta \text{ rule})\\
&= [x:K]([y:K']y)c(x) && (\beta \text{ rule})\\
&= [x:K]([y:K']y)(x) && (\xi \text{ and coercive definition})\\
&= [x:K]([y:K']y)(c'(x)) && (\xi \text{ and coercive definition})\\
&= [x:K](c'(x)) && (\beta \text{ rule})\\
&= c' : (K)K' && (\eta \text{ rule})
\end{aligned}$$

This fact implies that without the coherence condition, in $T[\mathcal{C}]$ we can prove some result that we can't get in T. That's the reason why we define coherence before introducing the coercion application rule. And we have to use a coherent set of $\mathcal{C}$, otherwise the conservativity cannot hold.

5.3 Relation between $T[\mathcal{C}]$ and T

Now, we would like to consider the relation between system $T[\mathcal{C}]$ and T. The example in Section 4 gives us the basic idea of dealing their relation. However, it is more complicated in LF, there are several extra things we need to consider.

We need to extend the form of judgements. As we have rules of subtyping and subkinding and derivations of them, we consider the subtyping and subkinding as judgements as well.

Coercive application rule

$$(CA^*1)\frac{\Gamma \vdash f : (x:K)K' \qquad \Gamma \vdash k_0 : K_0 \qquad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f * k_0 : [c(k_0)/x]K'}$$

$$(CA^*2)\frac{\Gamma \vdash f = f' : (x:K)K' \qquad \Gamma \vdash k_0 = k_0' : K_0 \qquad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f * k_0 = f' * k_0' : [c(k_0)/x]K'}$$

Coercive definition rule

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

Figure 5 The coercive application and definition rules of $T[\mathcal{C}]^*$.

So we introduce two new forms of judgements:

$$\Gamma \vdash A <_c B : \mathbf{Type} \quad \textit{and} \quad \Gamma \vdash K_1 <_c K_2$$

Since we have two new forms of judgements, we need to consider the equivalence between these judgements as well. We can extend the Definition 4 with the following two cases:

▶ **Definition 13.** (equality between the subtyping and subkinding judgements) Let S be a type theory specified in LF:

1. $(\Gamma_1 \vdash A_1 <_{c_1} B_1 : \mathbf{Type}) =_s (\Gamma_2 \vdash A_2 <_{c_2} B_2 : \mathbf{Type})$ iff $\vdash \Gamma_1 = \Gamma_2$, $\Gamma_1 \vdash A_1 = A_2 : \mathbf{Type}$, $\Gamma_1 \vdash B_1 = B_2 : \mathbf{Type}$, $\Gamma \vdash c_1 = c_2 : (A_1)B_1$ are derivable in S.
2. $(\Gamma_1 \vdash K_1 <_{c_1} K_1') =_s (\Gamma_2 \vdash K_2 <_{c_2} K_2')$ iff $\vdash \Gamma_1 = \Gamma_2$, $\Gamma_1 \vdash K_1 = K_2$, $\Gamma_1 \vdash K_1' = K_2'$ and $\Gamma \vdash c_1 = c_2 : (K_1)K_1'$ are derivable in S.

It is straight to show the relation $=_s$ and $\sim_s$ are still equivalence relations in coercive subtyping extensions.

▶ **Theorem 14.** *Let S be a type theory with coercive subtyping specified in LF, $=_s$ and $\sim_s$ are equivalence relations.*

5.3.1 System $T[\mathcal{C}]_{0K}$

The system $T[\mathcal{C}]_{0K}$ is an intermediate system which extends T with subtyping and subkinding rules but no coercion application and definition rules. It is obtained from T by adding the new judgement form $\Gamma \vdash A <_c B : \mathbf{Type}$, $\Gamma \vdash K <_c K'$ and the inference rules in Figure 2 and 3. Since we don't have any coercion application rule in $T[\mathcal{C}]_{0K}$, the coercion judgements cannot be applied, $T[\mathcal{C}]_{0K}$ can be trivially proved as a conservative extension of T.

▶ **Proposition 15.** *System $T[\mathcal{C}]_{0K}$ is a conservative extension of system T.*

5.3.2 System $T[\mathcal{C}]^*$

We can think $T[\mathcal{C}]$ as a system obtained from $T[\mathcal{C}]_{0K}$ by adding the *coercive application* and *coercive definition* rules in Figure 4. We will extend $T[\mathcal{C}]_{0K}$ into another system $T[\mathcal{C}]^*$ with $*$ as gap holder when we apply coercive subtyping.

$T[\mathcal{C}]^*$ is the system obtained from $T[\mathcal{C}]_{0K}$ by adding the *coercive application* and *coercive definition* rules in Figure 5.

It is easy to find out that all the judgements with $*$ are not judgements in $T[\mathcal{C}]_{0K}$. It means that $T[\mathcal{C}]^*$ is conservative over $T[\mathcal{C}]_{0K}$

▶ **Proposition 16.** *$T[\mathcal{C}]^*$ is a conservative extension of $T[\mathcal{C}]_{0K}$.*

5.3.3 Relation between the systems

To describe the relation between the type system $T[\mathcal{C}]$, $T[\mathcal{C}]^*$ and $T[\mathcal{C}]_{0K}$, we introduce four algorithms Θ, Θ^*, θ_1 and θ_2 between the systems.

For two type systems T_1 and T_2, we write

$$f : T_1 \to T_2$$

if f is a function from the T_1-derivations to T_2-derivations.

We describe four algorithms, which are such functions:

$$\begin{aligned} \Theta &: T[\mathcal{C}] \to T[\mathcal{C}]_{0K} \\ \Theta^* &: T[\mathcal{C}]^* \to T[\mathcal{C}]_{0K} \\ \theta_1 &: T[\mathcal{C}] \to T[\mathcal{C}]^* \\ \theta_2 &: T[\mathcal{C}]^* \to T[\mathcal{C}] \end{aligned}$$

The algorithms behave in the following way:

- The algorithm Θ replaces the derivations of $\Gamma \vdash K_1 <_c K_2$ in the premises of coercive rules (CA1)(CA2)(CD) by derivations of $\Gamma \vdash c : (K_1)K_2$ and replaces the coercive applications by several ordinary applications.
- The algorithm Θ^* replaces the derivations of $\Gamma \vdash K_1 <_c K_2$ in the premises of coercive rules (CA*1)(CA*2)(CD*) by derivations of $\Gamma \vdash c : (K_1)K_2$ and replaces the coercive applications by several ordinary applications.
- The algorithm θ_1 replaces coercive applications in $T[\mathcal{C}]$ derivations by coercive applications in $T[\mathcal{C}]^*$, by inserting $*$ into appropriate places.
- The algorithm θ_2 replaces coercive applications of the form $f * a$ in $T[\mathcal{C}]^*$ by coercive applications $f(a)$ in $T[\mathcal{C}]$.

We need to show that our algorithms behave in the right way, they insert the coercions into where they should be. The following property guarantees that all the coercions are inserted correctly by the algorithms:

▶ **Proposition 17.**
1. *For any derivation t in $T[\mathcal{C}]$, t and $\Theta(t)$ are equivalent derivations in $T[\mathcal{C}]$.*
2. *For any derivation t' in $T[\mathcal{C}]^*$, t' and $\Theta^*(t')$ are equivalent derivations in $T[\mathcal{C}]^*$.*

With the proposition below, we can show that $T[\mathcal{C}]$ and $T[\mathcal{C}]^*$ are equivalent systems.

▶ **Proposition 18.**
1. *For any derivation t in $T[\mathcal{C}]$, t and $\theta_2(\theta_1(t))$ are equivalent derivations in $T[\mathcal{C}]$.*
2. *For any derivation t' in $T[\mathcal{C}]^*$, t' and $\theta_1(\theta_2(t'))$ are equivalent derivations in $T[\mathcal{C}]^*$.*

Finally, with the propositions above we can conclude the relations between our systems and intermediate systems. Their relations can be drawn as Figure 6.

- $T[\mathcal{C}]$ is a equivalent system of $T[\mathcal{C}]^*$.
- $T[\mathcal{C}]^*$ is a definitional extension of $T[\mathcal{C}]_{0K}$.
- $T[\mathcal{C}]_{0K}$ is a conservative extension of T.

$T[\mathcal{C}]^*$ is a definitional extension of $T[\mathcal{C}]_{0K}$ and $T[\mathcal{C}]_{0K}$ is a conservative extension of T, we would like to call that $T[\mathcal{C}]^*$ is a *D-conservative extension*[5] of T.

[5] There is a notion of *D-conservativity* in Luo's note [8], we have a different meaning with that.

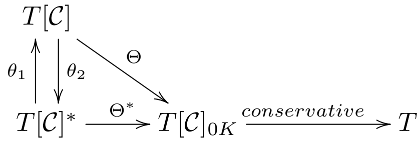

Figure 6 Relations between $T[\mathcal{C}]$, $T[\mathcal{C}]^*$, $T[\mathcal{C}]_{0K}$ and T.

▶ Remark. Although we have shown that $T[\mathcal{C}]^*$ with $*$-calculus has a more nature relationship with T, we still use $T[\mathcal{C}]$ as for description of coercive subtyping. $T[\mathcal{C}]$ itself is directly connected to important themes in the study of subtyping: *implicit coercions* and *subtyping as abbreviation*.

6 Conclusion and discussion

During the study of coercive subtyping, we find that conservativity is not enough to capture the relation between the systems. We borrow the idea of definitional extension from mathematical logic to describe the relation and formulate it in type theory. With a simple example, we demonstrate the relations and properties between a type system and its coercive subtyping extension. Although the example only consists of two basic types and one coercion, it's a nice shot containing the idea and key elements of the whole coercive subtyping extension story. We also give a sketch of the study on coercive subtyping in LF.

We hope this work presents a clear description of extending a type system with coercive subtyping and wish the notion of *definitional extension* can help with studies on other extensions in type theory. For example, *implicit syntax* of Pollack [16] is a good candidate. It starts from LEGO [10] and widely used on today's systems. We write terms with implicit arguments omitted and they are not well-typed in the system until the missing arguments have been inserted. It is not a conservative extension and we wish our notion could help to figure the exact relation out. More broadly, we can think of *elaboration*. An elaboration process maps surface language features to underlying constructions. We would like to see if elaboration is definitional extension or something more.

Acknowledgments. I would like to thank Zhaohui Luo for discussions regarding this topic and also thank to the anonymous reviewers for their valuable comments.

References

1 David Aspinall and Adriana Compagnoni. Subtyping dependent types. *Theoretical Computer Science*, 266(1-2):273–309, 2001.

2 Gilles Barthe and Maria João Frade. Constructor subtyping. In S. Doaitse Swierstra, editor, *Proceedings of Programming Languages and Systems, 8 conf. (ESOP'99)*, volume 1576 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 1999.

3 Robert Harper, Furio Honsell, and Gordon D. Plotkin. A framework for defining logics. In *Proceedings of Symposium on Logic in Computer Science 1987*, pages 194–204. IEEE Computer Society, 1987.

4 Martin Hofmann. *Extensional Concepts in Intensional Type Theory*. PhD thesis, University of Edinburgh, 1995.

5 Stephen Kleene. *Introduction to Metamathematics*. North Holland, 1952.

6 Zhaohui Luo. *Computation and Reasoning: A Type Theory for Computer Science*. Oxford University Press, 1994.
7 Zhaohui Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(1):105–130, 1999.
8 Zhaohui Luo. D-conservativity. Notes, January 2012.
9 Zhaohui Luo. Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophye*, 35(6):491–513, 2012.
10 Zhaohui Luo and Robert Pollack. Lego proof development system: User manual, 1992.
11 Zhaohui Luo, Sergei Soloviev, and Tao Xue. Coercive subtyping: Theory and implementation. *Information and Computation*, 223:18–42, February 2013.
12 Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.
13 Robin Milner. A theory of type polymorphism in programming. *Journal of Computer Systems and Sciences*, 17:348–375, 1978.
14 John C. Mitchell. Type inference with simple subtypes. *Journal of Functional Programming*, 1(3):245–285, 1991.
15 Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*. Oxford University Press, Oxford, 1990.
16 Robert Pollack. Implicit syntax. In the preliminary Proceedings of the 1st Workshop on Logical Frameworks, 1990.
17 Sergei Soloviev and Zhaohui Luo. Coercion completion and conservativity in coercive subtyping. *Annals of Pure and Applied Logic*, 113(1–3):297–322, 2002.
18 Tao Xue. *Theory and Implementation of Coercive Subtyping*. PhD thesis, Royal Holloway, University of London, 2013.
19 Tao Xue and Zhaohui Luo. Dot-types and their implementaion. *Logical Aspects of Computational Linguistics (LACL'12)*. *LNCS*, 7351:234–249, 2012.

A LF inference rules

Contexts and assumptions

$$\frac{}{<>\vdash \textbf{valid}} \qquad \frac{\Gamma \vdash K \;\textbf{kind} \quad x \notin FV(\Gamma)}{\Gamma, x : K \;\vdash \textbf{valid}} \qquad \frac{\Gamma, x : K, \Gamma' \vdash \textbf{valid}}{\Gamma, x : K, \Gamma' \vdash x : K}$$

$$\frac{\Gamma, \Gamma' \vdash J \quad \Gamma \vdash K \;\textbf{kind} \quad x \notin FV(\Gamma) \cup FV(\Gamma')}{\Gamma, x : K, \Gamma' \vdash J}$$

General equality rules

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

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

Equality typing rules

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

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

where J is of form: **valid**, K_0 **kind**, $k : K_0$, $K_1 = K_2$ *or* $k_1 = k_2 : K_0$

Substitution rules

$$\frac{\Gamma, x : K, \Gamma' \;\vdash \textbf{valid} \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \;\vdash \textbf{valid}}$$

$$\frac{\Gamma, x : K, \Gamma' \vdash K' \;\textbf{kind} \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' \;\textbf{kind}} \qquad \frac{\Gamma, x : K, \Gamma' \vdash k' : K' \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' : [k/x]K'}$$

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

$$\frac{\Gamma, x : K, \Gamma' \vdash K' \;\textbf{kind} \quad \Gamma \vdash k = k' : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' = [k'/x]K'} \qquad \frac{\Gamma, x : K, \Gamma' \vdash k' : K' \quad \Gamma \vdash k_1 = k_2 : K}{\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]k' = [k_2/x]k' : [k_1/x]K'}$$

The kind Type

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

Dependent product kinds

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

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

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

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

Figure 7 The inference rules of LF.

B Proof of Proposition 3

In a type system S specified in LF.

1. If Γ_1 is a valid context, $\vdash \Gamma_1 = \Gamma_1$
2. If $\Gamma \vdash \Gamma_1 = \Gamma_2$, then $\Gamma \vdash \Gamma_2 = \Gamma_1$.
3. If $\Gamma \vdash \Gamma_1 = \Gamma_2$ and $\Gamma \vdash \Gamma_2 = \Gamma_3$, then $\Gamma \vdash \Gamma_1 = \Gamma_3$.
4. If $\Gamma, \Gamma_1 \vdash J$ and $\Gamma \vdash \Gamma_1 = \Gamma_2$ then $\Gamma, \Gamma_2 \vdash J$. ($J$ is of form **valid**, K **kind**, $k\colon K$, $k_1 = k_2 \colon K$ or $K_1 = K_2$)

Proof. Suppose

$$\Gamma_1 \equiv x_1 : K_1, x_2 : K_2, \cdots, x_n : K_n$$
$$\Gamma_2 \equiv x_1 : M_1, x_2 : M_2, \cdots, x_n : M_n$$
$$\Gamma_3 \equiv x_1 : N_1, x_2 : N_2, \cdots, x_n : N_n$$

1. Straight by definition.
2. Since $\Gamma \vdash \Gamma_1 = \Gamma_2$, by definition we have:

$$\begin{aligned} \Gamma &\vdash K_1 = M_1; \\ \Gamma, x_1\colon K_1, \cdots, x_{i-1}\colon K_{i-1} &\vdash K_i = M_i \quad (i = 2, \cdots, n) \end{aligned}$$

For any $1 < i \leq n$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x_1\colon K_1, \cdots, x_{i-2}\colon K_{i-2}, x_{i-1}\colon K_{i-1} \vdash K_i = M_i \quad \Gamma, x_1\colon K_1, \cdots, x_{i-2}\colon K_{i-2} \vdash K_{i-1} = M_{i-1}}{\Gamma, x_1\colon K_1, \cdots, x_{i-2}\colon K_{i-2}, x_{i-1}\colon M_{i-1} \vdash K_i = M_i}}{\vdots}}{\Gamma, x_1 : K_1, x_2 : M_2, \cdots, x_{i-1} : M_{i-1} \vdash K_i = M_i} \quad \Gamma \vdash K_1 = M_1}{\Gamma, x_1 : M_1, x_2 : M_2, \cdots, x_{i-1} : M_{i-1} \vdash K_i = M_i}}{\Gamma, x_1 : M_1, x_2 : M_2, \cdots, x_{i-1} : M_{i-1} \vdash M_i = K_i}$$

and $i = 1$ is trivial with $\dfrac{\Gamma \vdash K_1 = M_1}{\Gamma \vdash M_1 = K_1}$. Hence, we have $\Gamma \vdash \Gamma_2 = \Gamma_1$ by definition.

3. Since $\Gamma \vdash \Gamma_2 = \Gamma_3$, by definition we have:

$$\begin{aligned} \Gamma &\vdash M_1 = N_1 \\ \Gamma, x_1\colon M_1, \cdots, x_{i-1}\colon M_{i-1} &\vdash M_i = N_i \quad (i = 2, \cdots, n) \end{aligned}$$

We have $\Gamma \vdash K_1 = M_1$, and from case 2:

$$\Gamma, x_1\colon M_1, \cdots, x_{i-1}\colon M_{i-1} \vdash K_i = M_i \quad (i = 2, \cdots, n)$$

In the LF, we have transitivity rules for equal kinds, so we can get:

$$\begin{aligned} \Gamma &\vdash K_1 = N_1 \\ \Gamma, x_1\colon M_1, \cdots, x_{i-1}\colon M_{i-1} &\vdash K_i = N_i \quad (i = 2, \cdots, n) \end{aligned}$$

For any $1 < i \leq n$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x_1\colon M_1, \cdots, x_{i-2}\colon M_{i-2}, x_{i-1}\colon M_{i-1} \vdash K_i = N_i \quad \Gamma, x_1\colon M_1, \cdots, x_{i-2}\colon M_{i-2} \vdash M_{i-1} = K_{i-1}}{\Gamma, x_1\colon M_1, \cdots, x_{i-2}\colon M_{i-2}, x_{i-1}\colon K_{i-1} \vdash K_i = N_i}}{\vdots}}{\Gamma, x_1 : M_1, x_2 : K_2, \cdots, x_{i-1} : K_{i-1} \vdash K_i = N_i} \quad \Gamma \vdash K_1 = M_1}{\Gamma, x_1 : K_1, x_2 : K_2, \cdots, x_{i-1} : K_{i-1} \vdash K_i = N_i}$$

We have $\Gamma \vdash \Gamma_1 = \Gamma_3$ by definition.

4.

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x_1\colon K_1, \cdots, x_{n-1}\colon K_{n-1}, x_n\colon K_n \vdash J \quad \Gamma, x_1\colon K_1, \cdots, x_{n-1}\colon K_{n-1} \vdash K_n = M_n}{\Gamma, x_1\colon K_1, \cdots, x_{n-1}\colon K_{n-1}, x_n\colon M_n \vdash J}}{\vdots}}{\Gamma, x_1 : K_1, x_x : M_2, \cdots, x_n : M_n \vdash J} \quad \Gamma \vdash K_1 = M_1}{\Gamma, x_1 : M_1, x_x : M_2, \cdots, x_n : M_n \vdash J}$$

Hence we have $\Gamma, \Gamma_2 \vdash J$.