

Coercive Subtyping for the Calculus of Constructions (extended abstract)

Gang Chen^{*}

Computer Science Department, Boston University
Boston, MA 02215, USA

gangchen@types.bu.edu

ABSTRACT

We present a coercive subtyping system for the calculus of constructions. The proposed system λC_{\leq}^{co} is obtained essentially by adding coercions and η -conversion to λC_{\leq} [10], which is a subtyping extension to the calculus of constructions without coercions. Following [17, 18], the coercive subtyping $c : A \leq B$ is understood as a special case of typing in arrow type $c : A \rightarrow B$ such that the term c behaves like an identity function. We prove that, with respect to this semantic interpretation, the proposed coercive subtyping system is sound and complete, and that this completeness leads to transitivity elimination (transitivity rule is admissible). In addition, we establish the equivalence between λC_{\leq}^{co} and $CC_{\beta\eta}$, this fact implies that λC_{\leq}^{co} has confluence, subject reduction and strong normalization. We propose a formalization of coercion inference problem and present a sound and complete coercion inference algorithm.

Categories and Subject Descriptors

D.3 [Software]: Programming Languages

General Terms

Languages

Keywords

Subtyping, Coercion, Calculus of Constructions, Semantics of Coercions, Transitivity Elimination

1. INTRODUCTION

The Calculus of Constructions (generally abbreviated as CC or λC) is a dependent type theory, in which types can depend on both terms and types, and terms can take terms and types as arguments. This calculus is highly expressive

^{*}Partially supported by the NSF Grant No. CCR-9988529

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

POPL'03, January 15–17, 2003, New Orleans, Louisiana, USA.
Copyright 2003 ACM 1-58113-628-5/03/0001 ...\$5.00.

and can be used as both a logic and as a programming language. Several widely used proof assistants, e.g. Coq and LEGO, are based on extensions of CC.

Researchers have found the need to use coercive subtyping in proof assistants in order to reuse definitions and proofs [3, 21]. Coercion inference algorithms have been implemented in LEGO [2] and Coq [24]. But few formal studies exist for the moment, with the exception of the work of Alex Jones, Zhaohui Luo and Sergei Soloviev [15], which will be discussed later.

The assertion that the type A can be coerced to the type B by the coercion c can be represented by the judgment:

$$c : A \leq B$$

The study of coercion concerns the following questions:

Question 1. What is the meaning of such a coercive subtyping judgment?

The term “coercion” has been used in somewhat different senses by different researchers. For example, c can be just an encoding of the derivation of $A \leq B$ (e.g. [13]), or a term inhabited in $A \rightarrow B$ (e.g. [21]).

In this paper, we follow [17, 18] and consider the coercive subtyping as “restricted linear implication”. First of all, c should be a function from A to B ; secondly, c should behave like an identity function. Formally, these requirements can be represented as:

$$c : A \rightarrow B \wedge \text{erase}(c) \rightarrow_{\eta} \lambda x.x \quad (*)$$

where erase is a function which erases the type information of c , and \rightarrow_{η} denotes zero or more steps of η -reduction. The right part of $(*)$ formalizes the linearity of the implication.

Question 2. What is an ideal coercive subtyping system?

First, the system should be sound, meaning that, any judgment derivable from the system should satisfy $(*)$. Second, it should be complete, that is, any judgment satisfying $(*)$ should be derivable. Third, it should be coherent, that is, coercions should be unique. More precisely, if both $c : A \leq B$ and $d : A \leq B$ are derivable, then c, d should be equivalent in some sense. Fourth, it is preferable to formulate the subtyping system so that it has the transitivity elimination property. Among other things, systems with such a property are suitable for the meta-theoretical study.

All the above points can be summarized as follows:

1. Subtyping as implication:
 $\Gamma \vdash c : A \leq B \Rightarrow \Gamma \vdash c : A \rightarrow B$

2. Coercions erase to the identity:
 $erase(c) \rightarrow_{\eta} \lambda x.x$
3. Completeness of coercive subtyping:
 $\Gamma \vdash c : A \rightarrow B \wedge erase(c) \rightarrow_{\eta} \lambda x.x \Rightarrow \Gamma \vdash c : A \leq B$
4. Admissibility of transitivity
 $c : A \leq B \wedge d : B \leq C \Rightarrow \exists e. e : A \leq C$
5. Coherence of coercions:
 $\Gamma \vdash c : A \leq B \wedge \Gamma \vdash d : A \leq B \Rightarrow c =_R d$

where $=_R$ is some notion of coercion equivalence.

In [17, 18], it is shown that the subtyping system CO^+ (an extension of System-F with coercive subtyping) has all the above properties, so we will refer to this set of properties as the “LMS framework”, which provides guidance in building the coercive subtyping system. Our work will follow this framework.

Question 3. What is the coercion inference?

The idea of coercion inference has been explored by several authors [2, 24, 21] etc. under the notion of “Implicit coercion”. To understand it, consider a term, say $f(c(a))$, where c is a coercion. In most situations, users prefer to write just the coercion-free term $f(a)$; a coercion inference algorithm is expected for inserting the missing coercion c .

In this work we give a formalization of this notion of coercion inference. Roughly speaking, the coercion inference problem is stated as,

Given a context Γ and a term M , decide if there are terms M', A such that the typing $\Gamma \vdash M' : A$ is derivable and M can be obtained by removing coercions from M' .

A more formal presentation, which will be found in this paper, will make clear the procedure of “removing coercions”. Such a formulation allows us to state and to prove the soundness and completeness of the coercion inference algorithm.

In the rest of this section, we explain how to construct the coercive subtyping system in order to achieve the 5 requirements of the LMS framework.

The formal system λC_{\leq}^{co} will be presented in the next section, followed by an example of an application of coercive subtyping in formalized mathematics (Sect. 3). In Sect. 4, we show that the coercive subtyping system is sound with respect to the semantics ($*$) and we establish the equivalence between λC_{\leq}^{co} and $CC_{\beta\eta}$, the latter is CC with $\beta\eta$ -conversion[14]. From this equivalence, it follows that λC_{\leq}^{co} has all the nice properties of $CC_{\beta\eta}$ such as confluence, subject reduction, and strong normalization. In Sect. 5, we study the completeness of coercions, from which the transitivity elimination can be derived. In Sect. 6, we describe an algorithmic coercive subtyping and typing system, and define formally the problem of coercion inference. Then we give a coercion inference algorithm and show that it is sound and complete. In Sect. 7, we describe the anti-symmetry of the subtyping relation, and the coherence of coercion. The proofs for these results are only sketched, the detailed proofs can be found in [11]. Finally, we summarize this work and discuss the related works.

Now we discuss the design decision for λC_{\leq}^{co} .

1.1 Transitivity Elimination and λC_{\leq}

There are two purposes of this subsection. First, we explain how to construct a subtyping system in dependent type theory such that it enjoys the transitivity elimination. Second, we briefly recall the system λC_{\leq} [10](or Part III of [11]) which is a subtyping extension to CC without coercions. λC_{\leq}^{co} will be formed essentially by adding coercions to λC_{\leq} .

A notable obstacle to a proof theoretical study of a subtyping extension to a dependent type theory is the conflict between type conversion and transitivity elimination. A dependent type (also called a type family) is actually a type-valued function, it induces the β -conversion between types. If two types are convertible, then intuitively they denote the same set of values and, in consequence, either type is a subtype of the other. This is usually represented by a conversion subtyping rule:

$$\frac{A =_{\beta} B}{A \leq B}$$

Unfortunately, in the presence of this rule, in any non-trivial subtyping system¹, the following transitivity rule:

$$\frac{A \leq C \quad C \leq B}{A \leq B}$$

can not be eliminated².

The explicit presence of such a transitivity rule makes it difficult to check the subtyping relation. Furthermore, in a dependent type theory with subtyping, this rule is an obstacle to the proof of subject reduction. Our approach to resolving the conflict between type conversion and transitivity elimination is to replace the conversion subtyping rule with two rules S-id and S- β :

$$\overline{\Gamma \vdash A \leq A} \text{ S-id}$$

$$\frac{A =_{\beta} B \quad \Gamma \vdash B \leq C \quad C =_{\beta} D \quad \Gamma \vdash A, B, C, D : s}{\Gamma \vdash A \leq D} \text{ S-}\beta$$

where $\Gamma \vdash A, B, C, D : s$ means that A, B, C, D are well-formed and of the sort s . Such a treatment is independent of number of the β -reductions.

The remainder of this subsection completes the presentation of λC_{\leq} . The primitive subtyping declaration is introduced into the context in the form:

$$\Gamma, \alpha \leq A : \Pi x_1 : A_1 .. \Pi x_n : A_n. \star$$

where α is a new variable, \star is the sort intuitively denoting the class of all types, A is a dependent type of the kind $\Pi x_1 : A_1 .. \Pi x_n : A_n. \star$, that is, it is a function taking n parameters and return a type.

The whole subtyping system consists of the above two subtyping rules plus the rule S-II for subtyping between II-types

¹A subtyping system is a set of subtyping rules. A trivial subtyping system is a subtyping system in which $A \leq B$ implies $A =_{\beta} B$.

²Assume that there are no redundant rules in the subtyping system.

and the rule S- Γ for the application of primitive subtyping:

$$\frac{\Gamma \vdash A' \leq A \quad \Gamma, x : A' \vdash B \leq B'}{\Gamma \vdash \Pi x:A.B \leq \Pi x:A'.B'} \text{ S-II}$$

$$\frac{\Gamma \vdash PM_1..M_n \leq A \quad \alpha \leq P \in \Gamma}{\Gamma \vdash \alpha M_1..M_n \leq A} \text{ S-I}$$

A subtyping checking algorithm is obtained by changing S- β to S- β' :

$$\frac{A \rightarrow_{\beta} B \quad \Gamma \vdash B \leq C \quad C \leftarrow_{\beta} D}{\Gamma \vdash A \leq D} \text{ S-}\beta'$$

This rule does not contain a sorting condition because the well-formedness of A, D implies that of B and C . Therefore, in the subsumption rule, the well-formedness condition will ensure that all types in a subtyping derivation are well-formed. Since the algorithmic subtyping system does not have typing judgments in the assumptions, it is independent from the typing system.

λC_{\leq} enjoys many nice properties, including confluence, subject reduction, strong normalization and decidability of subtyping and typing (see [10] and Part III of [11]). λC_{\leq} is suitable to model theory reuse and proof reuse in proof assistants. For example, one can formulate and prove the assertion that ‘‘any monoid theorem is a group theorem’’ whose proof is actually a program which transforms proofs of monoid theorems into those of the corresponding group theorems.

Next we analyze how to add coercions into λC_{\leq} .

1.2 Coercion as Restricted Linear Implication

In the previous subsection, we have explained how to formulate the subtyping system so that it enjoys the transitivity elimination and we have presented λC_{\leq} . Now, we want to analyze how to construct a coercive subtyping system in which: 1. coercive subtyping becomes a special case of typing with the arrow type; 2. coercions behave like the identity function. This is also called ‘coercion as restricted linear implication’.

First, the system should be formulated in such a way that coercive subtyping corresponds to typing in arrow type:

$$c : A \leq B \quad \Rightarrow \quad c : A \rightarrow B$$

To this aim, two problems needs special consideration: one is how to make coercive subtyping between Π -types, another is how to introduce primitive coercive subtyping into contexts.

1.2.0.1 Coercive Subtyping for Π -types.

We begin by formulating a coercive subtyping rule for subtyping between arrow types, which is a special case of Π -type.

$$\frac{c : C \leq A \quad d : B \leq D}{\lambda f:(A \rightarrow B).\lambda x:C.d(f(c(x))) : A \rightarrow B \leq C \rightarrow D}$$

It is easy to verify that

$$\begin{aligned} & c : C \rightarrow A \quad \wedge \quad d : B \rightarrow D \\ \Rightarrow & \lambda f:(A \rightarrow B).\lambda x:C.d(f(c(x))) : (A \rightarrow B) \rightarrow (C \rightarrow D) \end{aligned}$$

A naive generalization from such a rule to a coercive sub-

typing rule between Π -types would be:

$$\frac{\Gamma \vdash c : C \leq A \quad \Gamma, x : C \vdash d : B \leq D}{\Gamma \vdash \lambda f:\Pi x:A.B.\lambda x:C.d(f(c(x))) : \Pi x:A.B \leq \Pi x:C.D}$$

Unfortunately, the property of subtyping as arrow type is lost:

$$\begin{aligned} & \Gamma \vdash c : C \rightarrow A \quad \wedge \quad \Gamma, x : C \vdash d : B \rightarrow D \\ \Rightarrow & \Gamma, f : \Pi x:A.B, x : C \vdash f(c(x)) : B[x := c(x)] \\ \not\Rightarrow & \Gamma \vdash \lambda f:\Pi x:A.B.\lambda x:C.d(f(c(x))) : \Pi x:A.B \rightarrow \Pi x:C.D \end{aligned}$$

A possible solution to this problem is to change the conclusion of the rule to

$$\begin{aligned} & \Gamma \vdash \lambda f:(\Pi x:A.B[x := c(x)].\lambda x:C.d(f(c(x)))) \\ & : \Pi x:A.B[x := c(x)] \leq \Pi x:C.D \end{aligned}$$

Such a rule is neither elegant nor amenable to subtyping checking. Instead, we propose

$$\frac{\Gamma \vdash c : C \leq A \quad \Gamma, x : C \vdash d : B[x := c(x)] \leq D}{\Gamma \vdash \lambda f:\Pi x:A.B.\lambda x:C.d(f(c(x))) : \Pi x:A.B \leq \Pi x:C.D}$$

This is consistent with the subtyping as arrow type principle.

1.2.0.2 Primitive Coercive Subtyping.

Now we study the problem of how to interpret the primitive coercive subtyping.

Recall that, in λC_{\leq} , primitive subtyping is introduced into the context in the form $\alpha \leq A : \Pi x_1:A_1..x_n:A_n.\star$ where α is a new variable. Primitive coercive subtyping can be formed by adding a new coercion variable to such a declaration: $c : \alpha \leq A : \Pi x_1:A_1..x_n:A_n.\star$. In this declaration, α and A are type-valued functions, so $\alpha \leq A$ can not be interpreted as arrow type:

$$c : \alpha \leq A \quad \not\Rightarrow \quad c : \alpha \rightarrow A$$

So can c still be interpreted as typable term? Assume $M_1 : A_1, M_2 : A_2[x_1:=M_1], \dots, M_n : A_n[x_1:=M_1, \dots, x_{n-1}:=M_{n-1}]$, then $\alpha M_1..M_n$ and $AM_1..M_n$ are types (terms of kind \star). We have

$$cM_1..M_n : \alpha M_1..M_n \rightarrow AM_1..M_n$$

This suggests that c can be typed as:

$$c : \Pi x_1:A_1..x_n:A_n.(\alpha x_1..x_n \rightarrow A x_1..x_n)$$

Such a typing declaration establishes a relationship between the coercive subtyping system $\lambda C_{\leq}^{\text{co}}$ and the original CC. In order to make such a relation explicit, we introduce a new rule *Sub2Imp* into the typing system:

$$\frac{c : \alpha \leq P : \Pi x_1:A_1..x_n:A_n.\star \in \Gamma}{\Gamma \vdash c : \Pi x_1:A_1..x_n:A_n.(\alpha x_1..x_n \rightarrow P x_1..x_n)}$$

Thus, we can establish the equivalence between $\lambda C_{\leq}^{\text{co}}$ and $CC_{\beta\eta}$.

1.2.0.3 η -conversion.

If η -conversion is added to a subtyping system without coercions, such as λC_{\leq} , then confluence will be lost. By comparison, a coercive subtyping system does not have this problem, see [21] for an example and discussions.

For a coercive subtyping system, η -conversion is also necessary to ensure the coherence. Consider, for example, the coercive subtyping judgment $c : (A \rightarrow B) \leq (A \rightarrow B)$. It may be derived by using the rule S-id or S-II. Then c

may be either $id_{A \rightarrow B}$ or $\lambda f:A \rightarrow B. \lambda x:A. id_B(f(id_{Ax}))$, where $id_A = \lambda x:A. x$ etc. The two terms are not β -convertible, but $\beta\eta$ -convertible.

Thus, λC_{\leq}^{co} will be based on $CC_{\beta\eta}$ [14]. Geuver has shown that $CC_{\beta\eta}$ has nice properties: confluence, subject reduction and strong normalization. By the equivalence between λC_{\leq}^{co} and $CC_{\beta\eta}$, which will be proved later, these properties are inherited by λC_{\leq}^{co} .

2. DEFINITION OF λC_{\leq}^{co}

First, types and terms of λC_{\leq}^{co} are the same as in $CC_{\beta\eta}$.

DEFINITION 2.1 (TYPES AND TERMS OF λC_{\leq}^{co}). *The set of pre-expressions of λC_{\leq}^{co} , denoted by \mathcal{T} , is defined as follows*

$$\begin{aligned} \mathcal{T} &::= \mathcal{V} \mid \mathcal{S} \mid \mathcal{T}\mathcal{T} \mid \lambda\mathcal{V}:\mathcal{T}.\mathcal{T} \mid \Pi\mathcal{V}:\mathcal{T}.\mathcal{T} \\ \mathcal{S} &::= \star \mid \square \end{aligned}$$

where \mathcal{V} is the collection of variables, and where \star and \square are called sorts.

Intuitively, \star and \square represent the set of types and kinds, respectively.

Contexts of λC_{\leq}^{co} are as λC_{\leq} except that primitive coercive subtyping declarations are added in the form $c:x \leq A:K$.

DEFINITION 2.2 (CONTEXT). *A declaration takes one of two forms: 1) $x:A$ with $A \in \mathcal{T}$ and $x \in \mathcal{V}$, and 2) $c:x \leq A:K$ with $A, K \in \mathcal{T}$ and $c, x \in \mathcal{V}$. In both forms of declarations, c, x are the subjects. c is the atomic coercion in $c:x \leq A:K$. A pre-context is a finite linearly ordered list of declarations, all with distinct subjects. The empty context is denoted by $\langle \rangle$. Γ is a well-formed context (or legal context) if $\Gamma \vdash A : B$ is derivable by the typing rules defined below (Def. 2.6).*

In the declaration $c : x \leq A : K$, both x and c are actually constants because they will not be used as quantified variables in, for example, $\Pi x:A. B$ or $\lambda x:A. M$. Sometimes we will use α instead of x .

DEFINITION 2.3 (JUDGMENTS). *There are two forms of judgments:*

$$\begin{aligned} \Gamma \vdash A : B & \quad \text{Typing judgment} \\ \Gamma \vdash c : A \leq B & \quad \text{Coercive subtyping judgment} \end{aligned}$$

We use $A, B, C, \dots, M, N, \dots, a, b, \dots$ for arbitrary pre-expressions (or pre-terms), Γ, Δ, \dots for precontexts, x, y, z, \dots for arbitrary variables. Letters s, s_1, s_2, \dots are sorts ranging over $\{\star, \square\}$.

DEFINITION 2.4 (REDUCTION). *The β -reduction and η -reduction on \mathcal{T} are the compatible closures of the following notions of reduction respectively:*

$$\begin{aligned} (\lambda x:A. B)C &\rightarrow_{\beta} B[x:=C] \\ \lambda x:A. Mx &\rightarrow_{\eta} M \quad \text{if } x \notin Fv(M) \end{aligned}$$

$$\rightarrow_{\beta\eta} \equiv \rightarrow_{\beta} \cup \rightarrow_{\eta}.$$

In the following, we will use \rightarrow_R and $=_R$ to denote reflexive, transitive closure of the reduction $\rightarrow_R, \rightarrow_R \cup \leftarrow_R$ respectively.

Conventions and abbreviations are listed in the following.

NOTATION 2.5. *Suppose that Γ is a pre-context, $M, A, B, A_i, B_i, i = 1..n$ are pre-expressions.*

$$\begin{aligned} \Gamma(\alpha) &\equiv B \quad \text{if } c : \alpha \leq B \in \Gamma \\ Fv(A) &\equiv \text{the set of free variables in } A \\ Dom_v(\Gamma) &\equiv \{x \mid x : A \in \Gamma\} \\ Dom_{co}(\Gamma) &\equiv \{c \mid c : \alpha \leq A : K \in \Gamma\} \\ Dom_{pt}(\Gamma) &\equiv \{\alpha \mid c : \alpha \leq A : K \in \Gamma\} \\ Dom(\Gamma) &\equiv Dom_v(\Gamma) \cup Dom_{co}(\Gamma) \cup Dom_{pt}(\Gamma) \\ x \in \Gamma &\equiv x \in Dom(\Gamma) \\ c \in \Gamma^{(n)} &\Leftrightarrow c : \alpha \leq A : \Pi x_1:A_1.. \Pi x_n:A_n. \star \in \Gamma \\ x \in M &\Leftrightarrow x \in Fv(M) \\ n f_R(M) &\equiv R \text{ normal form of } M \\ \Gamma \vdash A : B : C &\Leftrightarrow \Gamma \vdash A : B \wedge \Gamma \vdash B : C \\ \Gamma \vdash A_1, \dots, A_n : B &\Leftrightarrow \Gamma \vdash A_1 : B \wedge \dots \wedge \Gamma \vdash A_n : B \\ \Gamma \vdash c : A \leq B : s &\Leftrightarrow \Gamma \vdash c : A \leq B \wedge \Gamma \vdash A, B : s \\ \Gamma \vdash A =_{\beta\eta} B : C &\Leftrightarrow A =_{\beta\eta} B \wedge \Gamma \vdash A, B : C \end{aligned}$$

Variables in a context Γ can be divided into three parts. $Dom_v(\Gamma)$ are ‘‘true’’ variables which will be used in quantification; the set $Dom(\Gamma) \setminus Dom_v(\Gamma)$ is actually the set of constants. $Dom_{co}(\Gamma)$ are primitive coercions, $Dom_{pt}(\Gamma)$ are primitive subtypes.

The λC_{\leq}^{co} typing rules are as the typing rules of $CC_{\beta\eta}$ (CC with $\beta\eta$ -reductions, [14]) except for $start_{\leq}$, $weakening_{\leq}$, $subsumption$ and $Sub2Imp$, which will be explained below.

DEFINITION 2.6 (λC_{\leq}^{co} TYPING RULES). *Typing judgments are defined by the following rules. Letters s, s_1, s_2 range over $\{\star, \square\}$.*

$$\begin{aligned} \text{axiom} &\frac{}{\langle \rangle \vdash \star : \square} \\ (s_1, s_2) \text{ rule} &\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x:A. B : s_2} \\ \text{start} &\frac{\Gamma \vdash A : s \quad x \notin \Gamma}{\Gamma, x : A \vdash x : A} \\ \text{start}_{\leq} &\frac{\Gamma \vdash A : K : \square \quad c, x \notin \Gamma}{\Gamma, c : x \leq A : K \vdash x : K} \\ \text{weakening} &\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s \quad c, x \notin \Gamma}{\Gamma, x : C \vdash A : B} \\ \text{weakening}_{\leq} &\frac{\Gamma \vdash A : B \quad \Gamma \vdash D : K : \square \quad x \notin \Gamma}{\Gamma, c : x \leq D : K \vdash A : B} \\ \text{application} &\frac{\Gamma \vdash F : \Pi x:A. B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x:=a]} \\ \text{abstraction} &\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash \Pi x:A. B : s}{\Gamma \vdash \lambda x:A. b : \Pi x:A. B} \\ \text{subsumption} &\frac{\Gamma \vdash a : A \quad \Gamma \vdash c : A \leq B : s}{\Gamma \vdash c(a) : B} \\ \text{conversion} &\frac{\Gamma \vdash a : A \quad \Gamma \vdash A =_{\beta\eta} B : s}{\Gamma \vdash a : B} \end{aligned}$$

$$\text{Sub2Imp} \frac{c : \alpha \leq P : \Pi x_1:A_1.. \Pi x_n:A_n. \star \in \Gamma}{\Gamma \vdash c : \Pi x_1:A_1.. \Pi x_n:A_n. (\alpha x_1..x_n \rightarrow Px_1..x_n)}$$

$start_{\leq}$ and $weakening_{\leq}$ introduce the primitive coercive subtyping into the context. The rule $Sub2Imp$ explicitly gives the type for the primitive coercion. Thus, the subtyping declaration $c : \alpha \leq P : \prod x_1:A_1.. \prod x_n:A_n.\star$ can be viewed as two typing declarations $\alpha : \prod x_1:A_n.. \prod x_n:A_n.\star$ and $c : \prod x_1:A_1.. \prod x_n:A_n.(\alpha x_1..x_n \rightarrow P x_1..x_n)$, where c, α should be taken as constants. Therefore, the contexts are essentially the same as those in $CC_{\beta\eta}$ (with declarations of constants).

In some coercion-free subtyping systems, e.g. $\lambda P_{\leq}[1]$, $\lambda\Pi_{\leq}$ and λC_{\leq} , the subsumption rule replaces the conversion rule. In the presence of coercions, the conversion rule can not be eliminated. Consider the following declaration:

$$\frac{\Gamma \vdash M : A \quad \frac{A =_{\beta\eta} B}{\Gamma \vdash \lambda x:A.x : A \leq B}}{\Gamma \vdash (\lambda x:A.x)M : B} \text{ subsumption}$$

Without the conversion rule, the judgement $\Gamma \vdash M : B$ is not provable and subject reduction will fail.

Next, we present the coercive subtyping rules.

DEFINITION 2.7 (COERCIVE SUBTYPING RULES).

$$S-II \quad \frac{\Gamma \vdash c : A' \leq A : s' \quad \Gamma, x : A \vdash B : s \quad \Gamma, x : A' \vdash d : B[x:=c(x)] \leq B' : s}{\Gamma \vdash \lambda f:\Pi x:A.B.\lambda x:A'.d(f(c(x))) : \Pi x:A.B \leq \Pi x:A'.B'}$$

$$S-G \quad \frac{\Gamma \vdash c : PM_1..M_n \leq A : \star \quad d : \alpha \leq P \in \Gamma \quad \Gamma \vdash \alpha M_1..M_n : \star}{\Gamma \vdash \lambda x:(\alpha M_1..M_n).c((dM_1..M_n)x) : \alpha M_1..M_n \leq A}$$

$$S-id \quad \frac{\Gamma \vdash A : s}{\Gamma \vdash \lambda x:A.x : A \leq A}$$

$$S-\beta\eta \quad \frac{\Gamma \vdash A =_{\beta\eta} B : s \quad \Gamma \vdash c : B \leq C \quad \Gamma \vdash C =_{\beta\eta} D : s}{\Gamma \vdash c : A \leq D}$$

where s ranges over $\{\star, \square\}$.

Observe that a coercion is always of the form: $\lambda x:A.M$. Judgments of the form $\Gamma \vdash A : s$ are called sorting conditions. The presence of these conditions in the subtyping rules ensures that:

$$\Gamma \vdash A \leq B \Rightarrow \Gamma \vdash A, B : s$$

When it is clear from the context, the sorting conditions will be omitted.

Some important admissible rules are:

Reflexivity

$$\Gamma \vdash A, B : s \wedge A =_{\beta\eta} B \Rightarrow \Gamma \vdash \lambda x:A.x : A \leq B$$

Transitivity

$$\Gamma \vdash c : A \leq B \wedge \Gamma \vdash d : B \leq C \wedge \Gamma \vdash A, B, C : s \Rightarrow \exists e. e =_{\beta\eta} \lambda x:A.d(c(x)) \wedge \Gamma \vdash e : A \leq C$$

Application of primitive coercive subtyping

$$c : (\alpha \leq A : \prod x_1:A_1.. \prod x_n:A_n.\star) \in \Gamma \wedge \Gamma \vdash M_i[x_1 := M_1, \dots, x_{i-1} := M_{i-1}] : A_i \quad i = 1..n \Rightarrow \exists e. e =_{\beta\eta} cM_1..M_n \quad \Gamma \vdash e : \alpha M_1..M_n \leq AM_1..M_n$$

The first and the third property can be easily verified. The admissibility of transitivity will be proved later.

The system λC_{\leq}^{co} can be viewed as a reformulation of $CC_{\beta\eta}$. The precise relation between the two will be presented in Section 4.1.

3. EXAMPLE

There is a growing interest in using subtyping in proof assistants. The objective is for theory and proof reuse[20, 21]. We present here an example, adapted from Saibi[24], of using coercive subtyping for representing groups based on the representation of monoids. A coercion free version of this example has been appeared in [10] and [11](Part III).

Recall that a monoid has a carrier set, a binary operation, a unit element and three axioms. In CC, the monoid structure can be encoded as:

$$\begin{array}{lcl} \mathcal{M} & : & \star \\ S & : & \mathcal{M} \rightarrow \star \\ Op & : & \Pi M:\mathcal{M}. S(M) \rightarrow S(M) \rightarrow S(M) \\ E & : & \Pi M:\mathcal{M}. S(M) \\ IdL & : & \Pi M:\mathcal{M}.\Pi x:S(M).Op(E(M), x) = x \\ IdR & : & \dots \\ Assoc & : & \dots \end{array}$$

where \mathcal{M} is the type of monoids (intuitively, it is the set of names of monoids). Given a monoid $M : \mathcal{M}$, $S(M)$ is the carrier set of M . $Op(M)$ and $E(M)$ are binary operation and the unit member of M respectively. $Assoc, IdR, IdL$ are monoid axioms. The typing for IdR and $Assoc$ are omitted³. The theorem that the multiplication of units is equal to the unit ($e \cdot e = e$) is provable:

$$\Gamma \vdash p : \Pi M:\mathcal{M}.Op(M)(E(M), E(M)) = E(M)$$

where $p = \lambda M:\mathcal{M}.IdR(M)(E(M))$.

Similarly, the group structure can be represented in CC by introducing the type $\mathcal{G} : \star$ for groups and a corresponding set of declarations. For example, IdR should be adapted to:

$$IdR_G : \Pi G:\mathcal{G}.\Pi x:G. Op(G)(x, E_G(G)) = x$$

Obviously, there is a significant redundancy in this representation. An alternative way is to define group as the extension of monoid by certain components. In λC_{\leq} , this can be realized as:

$$\begin{array}{lcl} \mathcal{G} & \leq & \mathcal{M} \\ Inv & : & \Pi G:\mathcal{G}. S(G) \rightarrow S(G) \\ & & \dots \end{array}$$

Given a group $G : \mathcal{G}$, $S(G)$ is well typed and it denotes the carrier set of G . Monoid operators Op, E and axioms $Assoc, IdR, IdL$ are all applicable to elements of \mathcal{G} . Inv is the inverse operator, whose axioms are omitted.

In λC_{\leq}^{co} , the group structure can be represented as

$$\begin{array}{lcl} c : \mathcal{G} & \leq & \mathcal{M} \\ Inv & : & \Pi G:\mathcal{G}. S(c(G)) \rightarrow S(c(G)) \\ & & \dots \end{array}$$

This representation is not as succinct as the previous one. But if we remove the coercions in the second representation, we get exactly the first representation. In practice, we can allow user to write expressions without explicit use of coercions, then apply a coercion inference procedure to derive the corresponding expressions in the coercion system. Coercive subtyping systems have been used in LEGO and Coq because they do not need to modify the underlying theories.

³The construction of concrete monoids can be implemented by using a constructor, whose arguments are of the types of the components of monoid and whose returned value is of the type \mathcal{M} . The technique is the same as in [24].

The coercion inference problem will be formalized in Sect. 6, where a coercion inference algorithm is presented along with the proofs of its soundness and completeness.

In [10] and [11](Part III), we have also shown how to turn the monoid theorem $e \cdot e = e$ to a group theorem and we prove a meta-theorem “any monoid theorem is a group theorem”. These results can be easily adapted in this coercive subtyping system.

4. SOUNDNESS OF COERCION

The soundness of coercions has two aspects: 1. coercive subtyping should correspond to typing in arrow type; 2. coercions should behave like the identity function. This section studies these two problems in λC_{\leq}^{co} . For the former, we show a stronger result: λC_{\leq}^{co} is equivalent to $CC_{\beta\eta}$ in typing.

4.1 Equivalence between λC_{\leq}^{co} and $CC_{\beta\eta}$

In this subsection, we will establish the equivalence between λC_{\leq}^{co} and the variant of $CC_{\beta\eta}$ with constants. The set of constants, denoted by \mathbf{C} , is a subset of variables: $\mathbf{C} \subseteq \mathcal{V}$, such that constants are never used in quantifications. The intention is that the set \mathbf{C} corresponds to the union of $Dom_{co}(\Gamma)$ and $Dom_{pt}(\Gamma)$ in λC_{\leq}^{co} . Formally, this can be realized by the following modified (s_1, s_2) rule and *abstraction* rule:

$$(s_1, s_2) \text{ rule} \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad x \notin \mathbf{C}}{\Gamma \vdash \Pi x : A. B : s_2}$$

$$\text{abstraction} \quad \frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash \Pi x : A. B : s \quad x \notin \mathbf{C}}{\Gamma \vdash \lambda x : A. b : \Pi x : A. B}$$

In the sequel we shall adopt the convention that $CC_{\beta\eta}$ means CC with $\beta\eta$ -conversion and constants.

Observe that, if $c : \alpha \leq P : K$ is a subtyping declaration in a context, then K must be of the form $\Pi x_1 : A_1 \dots \Pi x_n : A_n. \star$. The transformation function \mathcal{F} from contexts of λC_{\leq}^{co} to those of $CC_{\beta\eta}$ is defined as follows.

DEFINITION 4.1 (TRANSFORMATION \mathcal{F}). *The function \mathcal{F} which transforms a λC_{\leq}^{co} context to a $CC_{\beta\eta}$ context is recursively defined as follows, let $K \equiv \Pi x_1 : A_1 \dots \Pi x_n : A_n. \star$:*

$$\begin{aligned} \mathcal{F}(\langle \rangle) &= \langle \rangle \\ \mathcal{F}(\Gamma, x : A) &= \mathcal{F}(\Gamma), x : A \\ \mathcal{F}(\Gamma, c : \alpha \leq P : K) &= \\ &\mathcal{F}(\Gamma), \alpha : K, c : \Pi x_1 : A_1 \dots \Pi x_n : A_n. (\alpha x_1 \dots x_n \rightarrow P x_1 \dots x_n) \end{aligned}$$

Let $\mathcal{F}(\Gamma) \vdash_C A : B$ denote the $CC_{\beta\eta}$ judgment, in which elements of the set $\{c, \alpha \mid c : \alpha \leq P : \Pi x_1 : A_1 \dots \Pi x_n : A_n. \star \in \Gamma\}$ are taken as constants.

We will show that, under this transformation, λC_{\leq}^{co} is equivalent to $CC_{\beta\eta}$.

First, λC_{\leq}^{co} can be viewed as a subsystem of $CC_{\beta\eta}$.

PROPOSITION 4.2 (λC_{\leq}^{co} INTO $CC_{\beta\eta}$).

$$\begin{aligned} \Gamma \vdash c : A \leq B &\Rightarrow \mathcal{F}(\Gamma) \vdash_C c : A \rightarrow B \\ \Gamma \vdash A : B &\Rightarrow \mathcal{F}(\Gamma) \vdash_C A : B \end{aligned}$$

PROOF. Simultaneous induction on the derivation of $\Gamma \vdash c : A \leq B$ and $\Gamma \vdash A : B$. \square

On the other hand, it is obvious that

$$\mathcal{F}(\Gamma) \vdash_C A : B \Rightarrow \Gamma \vdash A : B$$

Thus we have the equivalence between λC_{\leq}^{co} and $CC_{\beta\eta}$

$$\Gamma \vdash A : B \Leftrightarrow \mathcal{F}(\Gamma) \vdash_C A : B$$

It follows that, subtyping in λC_{\leq}^{co} is a special case of arrow type

$$\Gamma \vdash c : A \leq B \Rightarrow \Gamma \vdash c : A \rightarrow B$$

In his thesis[14], Geuvers has shown that $CC_{\beta\eta}$ enjoys many nice properties. By means of the typing equivalence between λC_{\leq}^{co} and $CC_{\beta\eta}$ and the natural correspondence of $\beta\eta$ -reduction between the two systems, these properties are valid for λC_{\leq}^{co} . The most important results concerning $\beta\eta$ -reduction are: confluence, subject reduction and strong normalization.

4.2 Coercion as identity function

The intuition that coercions should behave like the identity function is usually formalized in the following way: 1. define an *erase* function which, intuitively, removes all type information of a term; 2. show that the erasure of a coercion will be η -reduced to the type-free function $\lambda x.x$.

For the first task, we define an *erase* function. Its definition is based on the same principle as that of Co^+ . One difference is that, in Co^+ , a term of form MA , where A is a type, will be erased to $erase(M)$. In this system, it will only be erased to $erase(M)erase(A)$. This is because there is actually no strict difference between types and terms in CC and types can have reductions. The definition of *erase* function in λC_{\leq}^{co} is thus defined as:

NOTATION 4.3 (ERASE). *Given a well-formed context Γ , a variable x , the function “ $erase_{\Gamma, x}$ ” is inductively defined as follows,*

$$\begin{aligned} erase_{\Gamma, x}(y) &= y \\ erase_{\Gamma, x}(dM_1 \dots M_n N) &= erase_{\Gamma, x}(N) \\ &\text{if } x \notin Fv(M_1, \dots, M_n) \wedge d \in \Gamma^{(n)} \\ erase_{\Gamma, x}(MN) &= erase_{\Gamma, x}(M)erase_{\Gamma, x}(N) \\ &\text{if } MN \text{ is not of the above form} \\ erase_{\Gamma, x}(\lambda y : A. M) &= \lambda y. erase_{\Gamma, x}(M) \\ &\text{if } x \notin A \wedge erase_{\Gamma, y}(M) \neq \perp \\ erase_{\Gamma, x}(M) &= \perp \text{ otherwise} \end{aligned}$$

where it is assumed that on the right sides of all equalities, we have $erase_{\Gamma, x}(N) \neq \perp$ and $erase_{\Gamma, x}(M) \neq \perp$.

$erase_{\Gamma, x}(M) = \perp$ implies that x appears in a type label or in a primitive subtyping. In such a case, M can not be considered as a legal coercion. The conditions $x \notin M_i$ and $x \notin A$ ensure that the erasure of M is essentially linear:

LEMMA 4.4 (LINEARITY OF ERASURE).

$$\begin{aligned} erase_{\Gamma, x}(M) &\twoheadrightarrow_{\eta} x \\ \Rightarrow x \in Fv(M) \wedge x \text{ appears in } M \text{ exactly once} \end{aligned}$$

This property is essential in the proof of the completeness of coercions.

The erasure of a coercion in λC_{\leq}^{co} may not be η -reducible to an identity function because coercions might contain β -redexes. Consider the following coercion derived by using S-II:

$$\lambda f:\Pi x:A.B.\lambda x:A'.(\lambda y:A.y)f(\lambda y:B.y) : \Pi x:A.B \leq \Pi x:A.B$$

The β -redexes of the coercion are preserved in its erasure: $\lambda f.\lambda x.((\lambda y.y)f)(\lambda y.y)$, which can not be η -reduced to the identity term $\lambda f.f$.

However these β -redexes are the only applications of the identity function. So we introduce *id*-reduction:

$$(\lambda x.x)N \rightarrow_{id} N$$

The ηid -reduction is defined as $\rightarrow_{\eta id} = \rightarrow_{\eta} \cup \rightarrow_{id}$. The erasure of a coercion can be ηid -reduced to an identity⁴:

PROPOSITION 4.5 (COERCION AS IDENTITY FUNCTION).

$$\Gamma \vdash \lambda x:A'.M : A \leq B \Rightarrow \text{erase}_{\Gamma,x}(M) \rightarrow_{\eta id} x$$

5. COMPLETENESS OF COERCIONS

This section investigates the inverse of the previous result, that is, if $\Gamma \vdash c : A \rightarrow B$ can be derived, and the erasure of c is η -reducible to $\lambda x.x$, then $\Gamma \vdash c : A \leq B$ is derivable. This result is called the completeness of coercions, intuitively it means that all semantically acceptable coercions are derivable by the coercive subtyping rules. The whole proof of this fact is long, an outline of the proof is given below. We refer the interested reader to the Part IV of [11] for more details.

One difficulty of studying the completeness of λC_{\leq}^{co} coercions is that coercions may not be in β -normal form. So it is difficult to analyze the form of a coercion. To get around of this problem, we change to the LMS style coercion judgment and construct a corresponding coercion system. In [17, 18], a coercion judgment takes the form:

$$x : A \vdash^{co} M : B$$

It reads: A is a subtype of B with coercion M . This judgment corresponds to a λC_{\leq}^{co} subtyping judgment:

$$\Gamma \vdash c : A \leq B \quad s.t. \quad c =_{\beta\eta} \lambda x:A.M$$

The advantage of a coercive subtyping system in LMS style is that a coercion term is almost in β -normal form. This facilitates the proof of completeness of coercions. The proof of λC_{\leq}^{co} completeness proceeds in following steps:

1. Define a LMS style coercion system
2. Show that it is equivalent to λC_{\leq}^{co}
3. Prove that it enjoys the completeness of coercion
4. Prove the completeness of λC_{\leq}^{co} coercion

First, the LMS style coercive subtyping system for λC_{\leq}^{co} is defined as:

$$\text{L-id} \quad \frac{}{\Gamma, x : A \vdash^{co} x : A}$$

$$\text{L-}\beta\eta \quad \frac{A =_{\beta\eta} B \quad \Gamma, x : B \vdash^{co} M : C \quad C =_{\beta\eta} D}{\Gamma, x : A \vdash^{co} M : D}$$

$$\text{L-II} \quad \frac{\Gamma, x : A' \vdash^{co} M : A \quad \Gamma, x : A', y : B[x:=M] \vdash^{co} N : B'}{\Gamma, f : \Pi x:A.B \vdash^{co} \lambda x:A'.N[y:=f(M)] : \Pi x:A'.B'}$$

$$\text{L-I} \quad \frac{\Gamma, y : PM_1..M_n \vdash^{co} N : A \quad d \in \Gamma^{(n)}}{\Gamma, x : \alpha M_1..M_n \vdash^{co} N[y:=dM_1..M_n x] : A}$$

⁴Observe that a subtyping judgement is always of the form $\Gamma \vdash \lambda x:A'.M : A \leq B$ where $A' =_{\beta\eta} A$.

where sorting conditions are omitted.

Coercions are almost in β -normal form. More precisely, the only β -redex in a coercion appears either in type labels or in terms of form $cM_1..M_n$ where $c \in \Gamma^{(n)}$. On the other hand, given a normalized term M in $CC_{\beta\eta}$, if its erasure is η -reduced to the identity then typically M will take the form:

$$M \equiv \lambda y_1:A_1..\lambda y_n:A_n.(c_n(c_{n-1}..(c_1(c_0x)M_1)..M_{n-1})M_n)$$

where c_i is of the form $dN_1..N_k \wedge d \in \Gamma^{(k)}$. This allows to prove the completeness of coercions for LMS style subtyping system:

LEMMA 5.1.

$$\begin{aligned} & \Gamma, x : A \vdash M : B \wedge \text{erase}_{\Gamma,x}(M) \rightarrow_{\eta} x \\ & \wedge M \equiv n f_{\beta}(M) \wedge x \notin Fv(B) \\ \Rightarrow & \Gamma, x : A \vdash^{co} M : B \end{aligned}$$

By means of the equivalence between λC_{\leq}^{co} and LMS style system:

$$\begin{aligned} \Gamma \vdash c:A \leq B & \Rightarrow \exists M. \Gamma, x:A \vdash^{co} M:B \wedge c =_{\beta\eta} \lambda x:A.M \\ \Gamma, x:A \vdash^{co} M:B & \Rightarrow \exists c. \Gamma \vdash c:A \leq B \wedge c =_{\beta\eta} \lambda x:A.M \end{aligned}$$

we obtain the completeness of coercions for λC_{\leq}^{co} :

THEOREM 5.2 (COMPLETENESS OF COERCIONS IN λC_{\leq}^{co}).

$$\begin{aligned} & \Gamma \vdash \lambda x:A'.M : A \rightarrow B \wedge \text{erase}_{\Gamma,x}(M) \rightarrow_{\eta} x \\ & \wedge M \equiv n f_{\beta}(M) \\ \Rightarrow & \exists c'. \Gamma \vdash c' : A \leq B \wedge \lambda x:A'.M =_{\beta\eta} c' \end{aligned}$$

By means of completeness, we can give an indirect proof of transitivity elimination. The idea is again from [17, 18].

THEOREM 5.3 (ADMISSIBILITY OF TRANSITIVITY).

$$\Gamma \vdash c:A \leq B:s \wedge \Gamma \vdash d:B \leq C:s \Rightarrow \exists e. \Gamma \vdash e:A \leq C$$

where $e =_{\beta\eta} \lambda x:A.d(c(x))$.

6. COERCION INFERENCE

One of the main objectives of coercive subtyping is to allow the user to write coercion-free terms so that the program could be succinct. For example, given a function $f : A \rightarrow B$, a term $a : A'$ and a coercion from A' to A : $c : A' \leq A$, the user is allowed to write $f(a)$. To type check this term, the coercion inference algorithm should insert a coercion c into $f(a)$ to get $f(c(a)) : B$.

Intuitively, the coercion inference problem can be stated as:

Given a context Γ and a term M , decide whether there exist terms M', A such that the judgment $\Gamma \vdash M' : A$ is derivable and M can be obtained by removing coercions from M' .

The fact that a well-formed term M' is obtained from M by inserting coercions is represented by the judgment:

$$\Gamma \vdash M \ll M' : A$$

We introduce the coercion reduction $\rightarrow_{co}^{\Gamma}$ which removes primitive coercion applications:

$$dM_1..M_n N \rightarrow_{co}^{\Gamma} N \quad \text{where } d \in \Gamma^{(n)}$$

The meaning of the Coercion Inference Judgment can be formally stated as:

$$\Gamma \vdash M \ll M' : A \Leftrightarrow M' \xrightarrow{\Gamma}_{co} M \wedge \Gamma \vdash M' : A$$

The Coercion Inference Problem is thus:

$$\text{Given } \Gamma, M, \text{ find } M', A \text{ such that } \Gamma \vdash M \ll M' : A$$

The first step of coercion inference is to check coercive subtyping, which is realized by the algorithmic coercive subtyping system. Similar to λC_{\leq} , this system is formed by replacing in the subtyping system the S- $\beta\eta$ rule by:

$$\text{SA-}\beta\eta \quad \frac{A \xrightarrow{\beta\eta} B \quad \Gamma \vdash_{\mathcal{A}} c : B \leq C \quad C \xleftarrow{\beta\eta} D}{\Gamma \vdash_{\mathcal{A}} c : A \leq D}$$

where $\Gamma \vdash_{\mathcal{A}} c : A \leq B$ denotes the algorithmic coercive subtyping judgment.

It is routine to verify that this algorithmic subtyping system is sound and complete with respect to the subtyping system and it is terminating.

The coercion inference algorithm is realized by a set of rules which derives the coercion inference judgment $\Gamma \vdash M \ll M' : A$. This set of rules are modified from typing rules. A typical example of a coercion inference rule is the rule for the application term:

$$\text{I-App} \quad \frac{\Gamma \vdash F \ll F' : C \quad \Gamma \vdash d : C \leq_{\Pi L_{ub}} \Pi x:A.B \quad \Gamma \vdash a \ll a' : A' \quad \Gamma \vdash_{\mathcal{A}} c : A' \leq A}{\Gamma \vdash Fa \ll d(F')(c(a')) : B[x:=c(a')]}$$

Given a coercion term Fa , we first derive the coercion expansions of its two subterms F' , a' and their types C , A' , then, use the judgment $\leq_{\Pi L_{ub}}$ to find the least Π type upper bound for C , which is assumed to be $\Pi x:A.B$, and derive the coercion $c : A' \leq A$. We obtain that the coercion expansion of Fa is $d(F')(c(a'))$ and its type is $B[x:=c(a')]$.

Because of space limitation we do not present the whole coercive subtyping system and coercion inference system here; the interested readers are invited to consult[11]. The coercion inference system is sound and complete:

PROPOSITION 6.1 (SOUNDNESS).

$$\Gamma \vdash M \ll M' : A \Rightarrow \Gamma \vdash M' : A \wedge M' \xrightarrow{\Gamma}_{co} M$$

THEOREM 6.2 (COMPLETENESS OF COERCION INFERENCE).

$$\begin{aligned} & \Gamma \vdash M : A \wedge M \xrightarrow{\Gamma}_{co} M' \\ \Rightarrow & \exists A_a, M_a, c \text{ s.t. } \Gamma \vdash M' \ll M_a : A_a \\ & \wedge \Gamma \vdash c : A_a \leq A \wedge cM_a =_{\beta\eta} M \end{aligned}$$

The completeness implies that the result (M') obtained from the coercion inference is minimal.

PROPOSITION 6.3 (DECIDABILITY).

Given M , Γ , the proposition

$$\exists N, A \text{ s.t. } \Gamma \vdash M \ll N : A$$

is decidable.

7. COHERENCE AND ANTI-SYMMETRY

The coherence of coercion means the uniqueness of subtyping derivation, formally, it asserts that all coercions for the same subtyping judgment $A \leq B$ are $\beta\eta$ -convertible. The proof of this result is tricky and takes several pages,

we only sketch the main steps here. We refer the interested reader to the Part IV of [11] for more details.

First, it is necessary to prove a special case of coherence when A and B are convertible:

$$A =_{\beta\eta} B \wedge \Gamma \vdash c : A \leq B \Rightarrow c =_{\beta\eta} \lambda x:A.x$$

The hard part of its proof is to show that any judgment of the following form can not be derived:

$$\Gamma \vdash c : \Gamma(\alpha)M_1..M_n \leq \alpha M_1..M_n$$

To prove this fact, we use the algorithmic coercive subtyping system. First, we observe that a proof for such a judgment could be characterized by a sequence:

$$\begin{aligned} \Gamma(\alpha)M_1..M_n \xrightarrow{\beta\eta} \alpha_1 N_1^1..N_{k_1}^1 & \leq \Gamma(\alpha_1)N_1^1..N_{k_1}^1 \xrightarrow{\beta\eta} \\ \alpha_2 N_1^2..N_{k_2}^2 \leq \dots \Gamma(\alpha_m)N_1^m..N_{k_m}^m & \xleftarrow{\beta\eta} \alpha M_1..M_n \end{aligned}$$

That is, the derivation is started with S-id and followed by alternative applications of S- Γ and SA- $\beta\eta$. Then, we show that there should be at least one α_i in this sequence such that α_i occurs before α in the context Γ . Finally, by induction on the position of α in the context Γ , it is proved that such a sequence can never be constructed.

Having proved the coherence for convertible subtyping, the anti-symmetry and coherence can be established.

PROPOSITION 7.1 (ANTI-SYMMETRY AND COHERENCE).

$$\begin{aligned} \Gamma \vdash e : A \leq B \wedge \Gamma \vdash e' : B \leq A & \Rightarrow A =_{\beta\eta} B \\ \Gamma \vdash e_1 : A \leq B \wedge \Gamma \vdash e_2 : A \leq B & \Rightarrow e_1 =_{\beta\eta} e_2 \end{aligned}$$

8. RELATED WORKS

8.0.0.4 Coercions.

The notion of coercion was first introduced in [5] as the semantical interpretation of the subtyping relation in the PER model of Bounded Fun[7], a subtyping extension of second order lambda calculus. Intuitively, in their work, the meaning of $A \leq B$ is an identity function (more precisely a morphism in PER) from the interpretation of A (a per) to that of B .

In [13] coercions are used for proving the coherence of subtyping, that is, the uniqueness of subtyping derivation. As coercions are used to encode subtyping derivation, so that the coherence is proved via the coercion conversions. This technique of proving coherence has been followed by several authors, for example, [8].

By means of coercions, in [4], the meaning of terms in Bounded Fun is given by their translations in the original second order lambda calculus (without subtyping). In their approach, the subtyping relation $A \leq B$ is interpreted as the typing relation $c : A \rightarrow B$ where c is the coercion term. They have also proved the uniqueness of the translation, that is, the uniqueness of coercions.

[22] also considers coercions to be a kind of type manipulation operations, which do not change the functional behavior of its argument. This is characterized by an *erase* function such that *erase*(c) is the result of erasing all type information of c . “ c is a coercion” is thus formally expressed as “*erase*(c) η -reduces to untyped identity”.

The subtyping system Co^{\perp} in [17, 18] contains most essential coercion properties: subtyping as implication; coercion erasing to identity; completeness of coercions; transitivity elimination and coherence of coercions. The adaptation of LMS framework to the present work is not straightforward

due to the type conversions which are not present in system F. In particular, the proof of anti-symmetry requires new technique. Another important issue which has not been addressed in their work is the problem of coercion inference. We have not only formalized this problem but given a sound and complete coercion inference algorithm.

The researches cited above concern only coercions in second order λ -calculus. Motivated by the need of using coercions in proof assistants, several authors have proposed to add coercions to dependent type systems. [3] studied a form of coercions over PTS. [24] and [2] implemented coercions in Coq and LEGO respectively. Recently, significant advancements have been achieved in the study of the the coercive subtyping system for the Unified Type Theory(UTT)[20, 21, 19, 25]. An important feature of this system is that it contains inductive types. Many meta-theoretic properties, including coherence and transitivity elimination, are proved. UTT[20, 21]. In a recent work, Alex Jones, Zhaohui Luo and Sergei Soloviev have proved a set of important meta-theoretical properties on a new version of such a system, including the coherence of coercions and the soundness of a coercion inference algorithm[15]. A basic idea behind coercive subtyping in the sense of these authors is that subtyping provides a mechanism for notational abbreviation in type theory. Compared with their work, ours have, among others, two notable results: 1. the coercive subtyping rules are semantically complete; 2. the coercion inference algorithm is complete.

8.0.0.5 Transitivity elimination.

The importance of transitivity elimination has been discussed in Subsection 1.1. Often, a subtyping system may contain several rules, which cause difficulty to the elimination of the transitivity rule. An effective approach to overcome this problem is to reformulate an equivalent subtyping system such that these “culprit” rules are replaced by some rules which do not prevent the elimination of transitivity. In [13], the rule of application of primitive subtyping has been reformulated. [17, 18] proposed a rule to replace the rule $\forall x:A.B \leq B[x := C]$. In dependent type theory, there is a difficulty due to subtyping between applications of dependent types. To overcome it, [23] has adapted the technique of [13] and realized a subtyping checking algorithm. [1] used the similar idea but they formulated a rule, which influenced the formulation of S- Γ in this work. Another “culprit” rule is what we called “conversion subtyping rule” in Subsection 1.1. There has been two main approaches towards this problem. The first is to restrict the use of the transitivity rule, e.g.[3, 24, 16]. The second is to use an intermediate subtyping system (often called algorithmic system) which are defined on types in normal forms, e.g. F_{\leq}^{ω} [23, 12] and λP_{\leq} [1]. Such an approach needs to separate different β -reduction, so the method is difficult to extend to systems with more reductions, such as CC.

[9] (or Part I of [11]) proposed a reformulation of λP_{\leq} , named $\lambda \Pi_{\leq}$. Among subtyping systems with dependent types, it is the first system defined on all types (not just on normalized types) such that the transitivity rule is not explicitly present but is admissible. This work is extended in [10] and Part III of [11] where a subtyping extension to CC (λC_{\leq}) is studied. As discussed in Subsection 1.1, in this system, the transitivity elimination is achieved by introducing the rule S- β .

8.0.0.6 Subtyping dependent types.

There are a few subtyping extensions to dependent types without coercions. The idea of extending dependent type theory with subtyping has first appeared in [6]. F^{ω} with bounded quantification has been studied in [23] and [12]. A subtyping extension to first order dependent type, called λP_{\leq} , has been first proposed and investigated in [1]. Our system is influenced by their works in several aspects, notably, the introduction of primitive subfamily relation, the rule S- Γ (which contributes to the transitivity elimination for primitive subtyping), the derivation of Π -type upper bound etc.

9. CONCLUSIONS AND FURTHER WORK

The coercive subtyping system λC_{\leq}^{co} is presented. We have investigated in the following aspects:

1. In the semantic aspect, coercive subtyping has been interpreted as typing in arrow type and coercions behave like identity function; By the formalization provided in this paper, λC_{\leq}^{co} is equivalent to $CC_{\beta\eta}$ with constants and the coercive subtyping system is semantically sound and complete.
2. In the proof theoretic aspect, λC_{\leq}^{co} is shown to be equivalent to $CC_{\beta\eta}$ with constants. Therefore, λC_{\leq}^{co} has nice properties like confluence, subject reduction and strong normalization. Besides, it enjoys properties specific to coercive subtyping: transitivity elimination, anti-symmetry and coherence.
3. In the algorithmic aspect, the notion of coercion inference is first formalized and a coercion inference algorithm is proposed. It is shown that the algorithm is sound, complete and terminating.

This system provides a basis for using coercions in proof assistants to realize theory reuse.

We have not addressed the problem of bounded quantification. For the application of subtyping in proof assistants, this feature does not seem to be necessary; but for the modeling of object-oriented programming languages it is indispensable. Another interesting line of investigation is to add inductive types to λC_{\leq}^{co} because proof assistants like LEGO and Coq are based on extensions of CC with inductive types.

Acknowledgment I am grateful to Zhaohui Luo for many helpful discussions, especially for his idea of definition reuse and proof reuse, and for his insight on the problem of η -reduction. Thanks to Stefano Berardi for his comments on the formalization of the problem of coercion inference. Many thanks to anonymous referees and to my wife Ping Hu. I am grateful to Paul Johnson at Berlitz for his help with my English. Finally, special thanks to my Ph.D supervisors Giuseppe Longo whose ideas on coercions provide the guidance of this work, and Giuseppe Castagna who has given me many helps during the preparation of my thesis, which contains the technical part of this work.

10. REFERENCES

- [1] D. Aspinall and A. Compagnoni. Subtyping dependent types. In *Proc. 11th Annual Symposium on Logic in Computer Science, IEEE*, pages 86–97, 1996.

- [2] A. Bailey. Lego with implicit coercions, 1996. draft.
- [3] G. Barthe. Implicit coercions in type systems. In S. Berardi and M. Coppo, editors, *Proceedings of Types'95, LNCS 1128*, pp 1–15., 1995.
- [4] V. Breazu-Tannen, T. Coquand, C. Gunter, and A. Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–221, July 1991.
- [5] K. B. Bruce and G. Longo. A modest model of records, inheritance, and bounded quantification. *Information and Computation*, 87:196–240, 1990.
- [6] L. Cardelli. Typechecking dependent types and subtypes. In *Proc. of the Workshop on Foundations of Logic and Functional Programming*, December 1987.
- [7] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, December 1985.
- [8] G. Castagna. *Object-Oriented Programming: A Unified Foundation*. Progress in Theoretical Computer Science. Birkäuser, Boston, 1997. ISBN 3-7643-3905-5.
- [9] G. Chen. Dependent type system with subtyping. Technical Report LIENS-96-27, Laboratoire d'Informatique, Ecole Normale Supérieure - Paris, 12 1996. Revised version in *Journal of Computer Science and Technology*, vol. 14, no. 1 (1999).
- [10] G. Chen. Subtyping calculus of construction, extended abstract. In *The 22nd International Symposium on Mathematical Foundation of Computer Science*, volume 1295. Springer-Verlag LNCS, August 1997. Bratislava, Slovakia.
- [11] G. Chen. *Subtyping, type conversions and elimination of transitivity*. PhD thesis, Université Paris 7, December 1998.
- [12] A. B. Compagnoni. Subtyping in F_{λ}^{ω} is decidable. Technical Report ECS-LFCS-94-281, LFCS University of Edinburgh, January 1994. and in CSL'94.
- [13] P.-L. Curien and G. Ghelli. Coherence of subsumption, minimum typing and the type checking in F_{\leq} . *Mathematical Structures in Computer Science*, 2(1):55–91, 1992.
- [14] H. Geuvers. *Logics and Type Systems*. PhD thesis, University of Nijmegen, Netherlands, 1993.
- [15] A. Jones, Z. Luo, and S. Soloviev. Some algorithmic and proof-theoretical aspects of coercive subtyping. In *Workshop on Subtyping, Inheritance and Modular Development of Proofs*, September 1997. Durham, U.K.
- [16] M. Lillibridge. *Translucent Sums: A Foundation for Higher-Order Module Systems*. PhD thesis, CMU, May 1997. CMU-CS-97-122.
- [17] G. Longo, K. Milsted, and S. Soloviev. A Logic of Subtyping, extended abstract. In *Logic in Computer Science (LICS)*, pages 292–300. IEEE, 1995. San Diego, June 1995.
- [18] G. Longo, K. Milsted, and S. Soloviev. Coherence and transitivity of subtyping as entailment, 1998. *Journal of Logic and Computation.*, vol. 10: 4, pp. 493–526, August 2000.
- [19] Y. Luo and Z. Luo. Coherence and transitivity of subtyping in coercive subtyping. In *Proc. of the 8th Inter. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'01), Havana, Cuba.*, number 2250 in LNAI, 2001.
- [20] Z. Luo. Coercive subtyping in type theory. In *CSL'96, the 1996 Annual Conference of the European Association for Computer Science Logic, Utrecht*, volume 1258 of LNCS, 1996.
- [21] Z. Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(97-13), 1997.
- [22] J. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76:211–249, 1988.
- [23] B. Pierce and M. Steffen. Higher-order subtyping. In *IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET)*, 1994. Full version in *Theoretical Computer Science*, vol. 176, no. 1–2, pp. 235–282, 1997 (with a corrigendum in TCS vol. 184 (1997), p. 247).
- [24] A. Saibi. Typing algorithm in type theory with inheritance. In *the 24th Annual SIGPLAN-SIGACT Symposium on principles of Programming Languages*, January 1997. Paris, France.
- [25] S. Soloviev and Z. Luo. Coercion completion and conservativity in coercive subtyping. *Annals of Pure and Applied Logic*, 113(1-3):297–322, 2002.