Weak Transitivity in Coercive Subtyping

Yong Luo, Zhaohui Luo and Sergei Soloviev

Department of Computer Science, University of Durham,
South Road, Durham, DH1 3LE, U.K.
E-mail: {Yong.Luo, Zhaohui.Luo}@durham.ac.uk

IRIT Univ. Paul Sabatier (Toulouse -III)
118 route de Narbonne, 31000 Toulouse, France
E-mail: soloviev@irit.fr

Abstract. Coercive subtyping is a general approach to subtyping, inheritance and abbreviation in dependent type theories. A vital requirement for coercive subtyping is that of coherence which essentially says that coercions between any two types must be unique. Another important task for coercive subtyping is to prove the admissibility or elimination of transitivity and substitution. In this paper, we propose and study the notion of *Weak Transitivity*, consider suitable subtyping rules for certain parameterised inductive types and prove its coherence and the admissibility of substitution and weak transitivity in the coercive subtyping framework.

1 Introduction

Coercive subtyping represents a general approach to subtyping and inheritance in type theory (see, for example, [Luo97,Luo99,SL02]). In particular, it provides a framework in which subtyping, inheritance, and abbreviation can be understood in dependent type theories. This paper investigates the issue of transitivity in coercive subtyping.

A problem with transitivity In the presentation of coercive subtyping in [Luo99], the transitivity rule

$$(Trans) \qquad \frac{\Gamma \vdash A <_c B : Type \;\; \Gamma \vdash B <_{c'} C : Type}{\Gamma \vdash A <_{c' \circ c} C : Type}$$

is included. Intuitively, it says that the composition of two coercions is also the coercion corresponding to transitivity. In [LL01], it has been proved that the transitivity rule is admissible for certain subtyping rules, such as those for Π-types and Σ-types.

However, the above transitivity rule is sometimes too strong (in intensional type theories). For some parameterised inductive data types together with natural subtyping rules, especially when the inductive type has more than one constructor, the above rule fails to be admissible or eliminatable. For instance,

for the inductive type of lists $List(A)$ parameterised by its element type A, if we introduce the following subtyping rule:

$$\frac{\Gamma \vdash A <_c B : Type}{\Gamma \vdash List(A) <_{d_{List}} List(B) : Type}$$

where $d_{List} = map(A, B, c)$ (see the detailed definition of d_{List} in Example 3) such that $d_{List}(nil(A)) = nil(B)$ and $d_{List}(cons(A, a, l)) = cons(B, c(a), d_{List}(l))$, then the transitivity rule $(Trans)$ fails to be admissible and, if we add it into the system, the coherence requirement fails to be satisfied. To see this, suppose we have $\Gamma \vdash F <_{c_1} E : Type$ and $\Gamma \vdash E <_{c_2} N : Type$, and by transitivity rule $(Trans)$, we also have $\Gamma \vdash F <_{c_2 \circ c_1} N : Type$. By the above subtyping rule for lists, we have respectively

$$\Gamma \vdash List(F) <_{map(F,E,c_1)} List(E) : Type$$

$$\Gamma \vdash List(E) <_{map(E,N,c_2)} List(N) : Type$$

$$\Gamma \vdash List(F) <_{map(F,N,c_2 \circ c_1)} List(N) : Type$$

By transitivity rule $(Trans)$, we also have

$$\Gamma \vdash List(F) <_{map(E,N,c_2) \circ map(F,E,c_1)} List(N) : Type$$

Now, the problem is that, in an intensional type theory,

$$\Gamma \not\vdash map(F, N, c_2 \circ c_1) = map(E, N, c_2) \circ map(F, E, c_1) : (List(F))List(N)$$

This means that we have two coercions between $List(F)$ and $List(N)$, but they are not computationally equal in an intensional type theory (and hence coherence fails), although we know that they are extensionally equal.

Remark 1. The problem showed in the above example arises when we consider subtyping *rules* for parameterised inductive types. This itself is a difficult issue, though these subtyping rules are very powerful and useful.

Weak transitivity – a proposed solution Rather than the above (strong) transitivity rule, we introduce a new concept –*Weak Transitivity,* which can informally be represented by the following rule:

$$(WTrans) \qquad \frac{\Gamma \vdash A <_c B : Type \;\; \Gamma \vdash B <_{c'} C : Type}{\Gamma \vdash A <_{c''} C : Type}$$

It says that, if $A <_c B$ and $B <_{c'} C$, then $A <_{c''} C$ for some coercion c''. Furthermore, we require that c'' be extensionally equal to $c' \circ c$ (see Section 5 for the treatment of the equality requirement). The essential difference compared with the strong transitivity rule $(Trans)$ is that we are only more concerned with the existence of c'' and such weak transitivity should be better suited to a wider application, that is, many natural subtyping rules are suitable for weak transitivity $(WTrans)$ but incompatible with the transitivity rule $(Trans)$.

Through our investigation, we also found out that weak transitivity does not necessarily hold for some parameterised inductive types such as dependent Σ-types which involve certain form of dependency between parameters. We show how such dependency can be made precise – we consider, in Section 3, a restricted form of inductive schemata, which forbids such dependency and hence enjoys weak transitivity[1]. For such inductive schemata, we develop a general method, which is also useful for implementation, to give the subtyping rules and the definition of coercions for a large class of parameterised inductive types. Then, in Section 4, we use the proof methods developed in [LL01] to prove, for these rules, the coherence and the admissibility of substitution and weak transitivity of the coercive subtyping system. Section 5 discusses the equality requirement for weak transitivity and shows that the general coercions we define satisfy the requirement of extensional equality. Discussions will be given in the last section of the paper.

In the following section, before presenting the above work, we give a brief introduction to coercive subtyping and explain some background notations to be used in latter sections.

2 Coercive subtyping and inductive schemata

2.1 Coercive subtyping

The basic idea of coercive subtyping, as studied in [Luo99], is that A is a subtype of B if there is a (unique) coercion c from A to B, and therefore, any object of type A may be regarded as an object of type B via c, where c is a functional operation from A to B in the type theory.

A coercion plays the role of abbreviation. More precisely, if c is a coercion from K_0 to K, then a functional operation f with domain K can be applied to any object k_0 of K_0 and the application $f(k_0)$ is definitionally equal to $f(c(k_0))$. Intuitively, we can view f as a context which requires an object of K; then the argument k_0 in the context f stands for its image of the coercion, $c(k_0)$. Therefore, one can use $f(k_0)$ as an abbreviation of $f(c(k_0))$.

The above simple idea, when formulated in the logical framework, becomes very powerful. The second author and his colleagues have developed the framework of coercive subtyping that covers variety of subtyping relations including those represented by parameterised coercions and coercions between parameterised inductive types. See [Luo99,Bai98,CL01,LC98,CLP01] for details of some of these development and applications of coercive subtyping.

Some important meta-theoretic aspects of coercive subtyping have been studied. In particular, the results on conservativity and on transitivity elimination for subkinding have been proved in [JLS98,SL02]. The conservativity result says, intuitively, that every judgement that is derivable in the theory with coercive

[1] Note that the harmful dependency is some form of dependency between parameters. Forbidding such dependency does not mean that all of the dependent types are excluded. See more details in Section 3 and Section 6.

subtyping and that does not contain coercive applications is derivable in the original type theory. Furthermore, for every derivation in the theory with coercive subtyping, one can always insert coercions correctly to obtain a derivation in the original type theory. The main result of [SL02] is essentially that coherence of basic subtyping rules does imply conservativity. These results not only justify the adequacy of the theory from the proof-theoretic consideration, but also provide the proof-theoretic basis for implementation of coercive subtyping.

How to prove coherence and admissibility of transitivity at the type level has been studied in [LL01] recently. In particular, the concept of *Well-defined coercions* has been developed, and the suitable subtyping rules for Π-types and Σ-types have been given as examples to demonstrate these proof techniques.

Coercion mechanisms with certain restrictions have been implemented both in the proof development system Lego [LP92] and Coq [B+00], by Bailey [Bai98] and Saibi [Sai97], respectively. Callaghan of the Computer Assisted Reasoning Group at Durham has implemented Plastic [CL01], a proof assistant that supports logical framework and coercive subtyping with a mixture of simple coercions, parameterised coercions, coercion rules for parameterised inductive types, and dependent coercions [LS99].

A formal presentation Coercive subtyping is formally formulated as an extension of (type theories specified in) the logical framework LF[2] [Luo94], whose rules are given in Appendix A. Types in LF are called kinds. The kind $Type$ represents the conceptual universe of types and a kind of the form $(x : K)K'$ represents the dependent product with functional operations f as objects (*e.g.*, abstraction $[x : K]k'$) which can be applied to objects of kind K to form application $f(k)$. For every type (an object of kind $Type$), $El(A)$ is the kind of objects of A. A kind is small if it does not contain $Type$. LF can be used to specify type theories, such as Martin-Löf's type theory [NPS90] and UTT [Luo94].

Notation We shall use the following notations:

- We often write $(K)K'$ for $(x : K)K'$ when x does not occur free in K', and A for $El(A)$ and hence $(A)B$ for $(El(A))El(B)$ when no confusion may occur.
- Substitution: We sometimes use $M[x]$ to indicate that variable x may occur free in M and subsequently write $M[N]$ for $[N/x]M$, when no confusion may occur.
- Context equality: for $\Gamma \equiv x_1 : K_1, ..., x_n : K_n$ and $\Gamma' \equiv x_1 : K'_1, ..., x_n : K'_n$, we shall write $\vdash \Gamma = \Gamma'$ for the sequence of judgements $\vdash K_1 = K'_1$, ..., $x_1 : K_1, ..., x_{n-1} : K_{n-1} \vdash K_n = K'_n$.
- Functional composition: for $f : (K_1)K_2$ and $g : (y : K_2)K_3[y]$, define $g \circ f =_{df} [x : K_1]g(f(x)) : (x : K_1)K_3[f(x)]$, where x does not occur free in f or g.

A system with coercive subtyping, $T[\mathcal{R}]$, is an extension of any type theory T specified in LF. It can be presented in two stages: first we consider the system $T[\mathcal{R}]_0$, which is an extension of T, with subtyping judgements of the form $\Gamma \vdash$

[2] The LF here is different from the Edinburgh Logical Framework [HHP87].

$A <_c B : Type$; then the system $T[\mathcal{R}]$, which is an extension of $T[\mathcal{R}]_0$, with subkinding judgements of the form $\Gamma \vdash K <_c K'$. The rules for subkinding can be found in Appendix B. Note that the substitution rule for subkinding (the last rule in Appendix B) can be eliminated under the assumption that the set of basic subtyping rules $\mathcal{R}$ is coherent [SL02]. As we are mainly concerned with the subtyping rules and their transitivity and coherence (in $T[\mathcal{R}]_0$), we shall omit the details of the kind level in this paper (details can be found in the forthcoming thesis of the first author).

$T[\mathcal{R}]_0$ is an extension of T with the following rules:

- A set $\mathcal{R}$ of subtyping rules whose conclusions are subtyping judgements of the form $\Gamma \vdash A <_c B : Type$.
- The congruence rule for subtyping judgements

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

In the presentation of coercive subtyping in [Luo99], $T[\mathcal{R}]_0$ also has the following substitution and transitivity rules:

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

$$(Trans) \qquad \frac{\Gamma \vdash A <_c B : Type \;\; \Gamma \vdash B <_{c'} C : Type}{\Gamma \vdash A <_{c' \circ c} C : Type}$$

Since we consider in this paper *Weak transitivity* and we will prove that the substitution rule is admissible, we do not include them as basic rules.

Remark 2. $T[\mathcal{R}]_0$ is obviously a conservative extension of the original type theory T, since the subtyping judgements do not contribute to any derivation of a judgement of any other form.

The most basic requirement for the subtyping rules (in $\mathcal{R}$) is that of coherence, given in the following definition, which essentially says that coercions between any two types must be unique.

Definition 1. (coherence condition) *We say that the subtyping rules are coherent if $T[\mathcal{R}]_0$ has the following coherence properties:*

1. *If $\Gamma \vdash A <_c B : Type$, then $\Gamma \vdash A : Type$, $\Gamma \vdash B : Type$, and $\Gamma \vdash c : (A)B$.*
2. *$\Gamma \not\vdash A <_c A : Type$ for any Γ, A and c.*
3. *If $\Gamma \vdash A <_c B : Type$ and $\Gamma \vdash A <_{c'} B : Type$, then $\Gamma \vdash c = c' : (A)B$.*

Remark 3. This is a weaker notion of coherence as compared with that given in [Luo99], since there the rules $(Subst)(Trans)$ are included in $T[\mathcal{R}]_0$. In general, when parameterised coercions and substitutions are present, coherence is undecidable. This is one of the reasons one needs to consider proofs of coherence in general.

2.2 Well-defined Coercions

After new subtyping rules are added into $\mathcal{R}$, we need to prove that $\mathcal{R}$ is still coherent and that the transitivity rule and substitution rule are admissible. A general strategy we adopt is to consider such proofs in a stepwise way. That is, we first suppose that some existing coercions (possibly generated by some existing rules) are coherent and have good admissibility properties, and then prove that all the good properties are kept after new subtyping rules are added. This leads us to define the following concept of *well-defined coercions* (WDC) [LL01].

Definition 2. (Well-defined coercions) *If $\mathcal{C}$ is a set of subtyping judgements of the form $\Gamma \vdash M <_d M' : Type$ which satisfies the following conditions, we say that $\mathcal{C}$ is a* well-defined set of judgements for coercions*, or briefly called* Well-Defined Coercions (WDC)*.*

1. *(Coherence)*
   (a) *$\Gamma \vdash A <_c B : Type \in \mathcal{C}$ implies $\Gamma \vdash A : Type$, $\Gamma \vdash B : Type$ and $\Gamma \vdash c : (A)B$.*
   (b) *$\Gamma \vdash A <_c A : Type \notin \mathcal{C}$ for any Γ, A, and c.*
   (c) *$\Gamma \vdash A <_{c_1} B : Type \in \mathcal{C}$ and $\Gamma \vdash A <_{c_2} B : Type \in \mathcal{C}$ imply $\Gamma \vdash c_1 = c_2 : (A)B$.*
2. *(Congruence) $\Gamma \vdash A <_c B : Type \in \mathcal{C}$, $\Gamma \vdash A = A' : Type$, $\Gamma \vdash B = B' : Type$ and $\Gamma \vdash c = c' : (A)B$ imply $\Gamma \vdash A' <_{c'} B' \in \mathcal{C}$.*
3. *(Transitivity) $\Gamma \vdash A <_{c_1} B : Type \in \mathcal{C}$ and $\Gamma \vdash B <_{c_2} A' : Type \in \mathcal{C}$ imply $\Gamma \vdash A <_{c_2 \circ c_1} A' : Type \in \mathcal{C}$.*
4. *(Substitution) $\Gamma, x : K, \Gamma' \vdash A <_c B : Type \in \mathcal{C}$ implies for any k such that $\Gamma \vdash k : K$, $\Gamma, [k/x]\Gamma' \vdash [k/x]A <_{[k/x]c} [k/x]B : Type \in \mathcal{C}$.*
5. *(Weakening) $\Gamma \vdash A <_c B : Type \in \mathcal{C}$, $\Gamma \subseteq \Gamma'$ and Γ' is valid imply $\Gamma' \vdash A <_c B : Type \in \mathcal{C}$.*

Remark 4. One may change the third condition (transitivity) to weak transitivity, *i.e.*, $\Gamma \vdash A <_{c_1} B : Type \in \mathcal{C}$ and $\Gamma \vdash B <_{c_2} A' : Type \in \mathcal{C}$ imply $\Gamma \vdash A <_{c_3} A' : Type \in \mathcal{C}$ for some c_3 such that c_3 and $c_2 \circ c_1$ are extensionally equal. This weak condition is also sufficient for the following development in this paper, except that some lemmas and proofs require minor changes.

In this paper, we consider the system of coercive subtyping in which the set $\mathcal{R}$ of the subtyping rules includes the following rule, where $\mathcal{C}$ is a WDC:

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

2.3 Inductive schemata and parameterised inductive types

In this subsection, we lay down some notations of inductive schemata, to be used in the next section (see [Luo94] for more details). We first give some formal definitions, then give some examples to explain.

Definition 3. ***(Inductive schemata)***

- *A strictly positive operator, with respect to a type variable X and a valid context Γ, is of one of the following forms:*
  1. $\Phi \equiv X$ *, or*
  2. $\Phi \equiv (x : K)\Phi_0$ *, where K is a small kind and Φ_0 is a strictly positive operator.*
- *An inductive schema Θ, with respect to a type variable X and a valid context Γ, is of one of the following forms:*
  1. $\Theta \equiv X$ *, or*
  2. $\Theta \equiv (x : K)\Theta_0$ *, where K is a small kind and Θ_0 is an inductive schema, or*
  3. $\Theta \equiv (x : \Phi)\Theta_0$ *, where Φ is a strictly positive operator and Θ_0 is an inductive schema and $x \notin FV(\Theta_0)$.*

Any finite sequence of inductive schemata $\overline{\Theta} \equiv< \Theta_1, ..., \Theta_m >$ $(m \in \omega)$ generates an inductive type $\mathcal{M}[\overline{\Theta}]$, with its introduction, elimination and computation rules. We shall consider the following form of parameterised inductive types:

$$\mathcal{T} =_{df} [Y_1 : P_1]...[Y_n : P_n]\mathcal{M}[\overline{\Theta}]$$

where $Y_1, ..., Y_n$ are parameters (λ-abstracted bound variable) and $P_1, ..., P_n$ are kinds.

Specification of parameterised types in LF Now, we declare the following constant expressions:

$$\begin{aligned}
\mathcal{T} &: (Y_1 : P_1)...(Y_n : P_n)Type \\
l_j &: (Y_1 : P_1)...(Y_n : P_n)\Theta_j[\mathcal{T}(\overline{Y})] \quad (j = 1, ..., m) \\
\mathcal{E}_{\mathcal{T}} &: (Y_1 : P_1)...(Y_n : P_n)(C : (\mathcal{T}(\overline{Y}))Type) \\
&\quad (f_1 : \Theta_1^\circ[\mathcal{T}(\overline{Y}), C, l_1(\overline{Y})])... \\
&\quad (f_m : \Theta_m^\circ[\mathcal{T}(\overline{Y}), C, l_m(\overline{Y})]) \\
&\quad (z : \mathcal{T}(\overline{Y}))C(z)
\end{aligned}$$

and assert the following computation rules: $(j = 1, ..., m)$

$$\mathcal{E}_{\mathcal{T}}(\overline{Y}, C, \overline{f}, (l_j(\overline{Y}, \Theta_j^v)) = f_j(\Theta_j^\sharp) : C(l_j(\overline{Y}, \Theta_j^v))$$

where Θ°, Θ^v and $\Theta^\sharp$ are formally introduced in the following definition, which will also be used in latter sections.

Definition 4. *Let Φ be a strictly positive operator and Θ an inductive schema. For $A : Type$, $C : (A)Type$, $f : (x : A)C(x)$, $y : \Phi[A]$ $z : \Theta[A]$,*

- *define kind $\Phi^*[C, y]$ as follows:*

$$\begin{aligned}
(X)^*[C, y] &= C(y) \\
((x : K)\Phi_0)^*[C, y] &= (x : K)\Phi_0^*[C, y(x)]
\end{aligned}$$

- *define kind* $\Theta^\circ[A, C, z]$ *as follows:*

$$\begin{aligned}(X)^\circ[A, C, z] &= C(z)\\ ((x : K)\Theta_0)^\circ[A, C, z] &= (x : K)\Theta_0^\circ[A, C, z(x)]\\ ((x : \Phi)\Theta_0)^\circ[A, C, z] &= (x : \Phi[A])(x' : \Phi^*[C, x])\Theta_0^\circ[A, C, z(x)]\end{aligned}$$

- *define* $\Phi^\natural[f, y]$ *as follows:*

$$\begin{aligned}(X)^\natural[f, y] &= f(y)\\ ((x : K)\Phi_0)^\natural[f, y] &= [x : K]\Phi_0^\natural[f, y(x)]\end{aligned}$$

- *Assume that* Θ *be of the form* $(x_1 : M_1)...(x_s : M_s)X$ *and* $x_1, ..., x_s$ *are fresh variables. Then* $\Theta^v =< x_1, ..., x_s >$ *and* $\Theta^\sharp$ *is defined as the following sequence of arguments:*
  1. *if* $\Theta \equiv X$ *then* $\Theta^\sharp =< >$
  2. *if* $\Theta \equiv (x_t : K)\Theta_0$ *then* $\Theta^\sharp =< x_t, \Theta_0^\sharp >$ $(t = 1, ..., s)$
  3. *if* $\Theta \equiv (x_t : \Phi)\Theta_0$ *then* $\Theta^\sharp =< x_t, \Phi^\natural[\mathcal{E}_\mathcal{T}(\overline{A}, C, \overline{f}), x_t], \Theta_0^\sharp >$ $(t = 1, ..., s)$

Example 1. We give three examples of parameterised inductive types.

1. Lists: $List =_{df} [A : Type]\mathcal{M}[X, (A)(X)X]$. This is equivalent to declaring the following constants:

$$\begin{aligned}List &: (A)Type\\ nil &: (A : Type)List(A)\\ cons &: (A : Type)(a : A)(l : List(A))List(A)\\ \mathcal{E}_{List} &: (A : Type)(C : (List(A))Type)(C(nil(A)))\\ &\quad ((a : A)(l : List(A))(C(l))C(cons(A, a, l)))\\ &\quad (z : List(A))C(z)\end{aligned}$$

   with computation rules:

$$\begin{aligned}\mathcal{E}_{List}(A, C, c, f, nil(A)) &= c : C(nil(A))\\ \mathcal{E}_{List}(A, C, c, f, cons(A, a, l)) &= f(a, l, \mathcal{E}_{List}(A, C, c, f, l))\\ &\quad : C(cons(A, a, l))\end{aligned}$$

2. Function types: $(\rightarrow) =_{df} [A : Type][B : Type]\mathcal{M}[((A)B)X]$.

$$\begin{aligned}(\rightarrow) &: (A : Type)(B : Type)Type\\ \lambda &: (A : Type)(B : Type)((A)B)(A \rightarrow B)\\ \mathcal{E}_{(\rightarrow)} &: (A : Type)(B : Type)(C : (A \rightarrow B)Type)\\ &\quad ((g : (A)B)C(\lambda(A, B, g)))(z : A \rightarrow B)C(z)\end{aligned}$$

   with computation rule:

$$\mathcal{E}_{(\rightarrow)}(A, B, C, f, \lambda(A, B, g)) = f(g) : C(\lambda(A, B, g))$$

3. *Either* types (disjoint union): $Either =_{df} [A : Type][B : Type]\mathcal{M}[(A)X, (B)X]$.

$$\begin{aligned} Either &: (A : Type)(B : Type)Type \\ left &: (A : Type)(B : Type)(A)Either(A, B) \\ right &: (A : Type)(B : Type)(B)Either(A, B) \\ \mathcal{E}_{Either} &: (A : Type)(B : Type)(C : (Either(A, B))Type) \\ &\quad ((a : A)C(left(A, B, a)))((b : B)C(right(A, B, b))) \\ &\quad (z : Either(A, B))C(z) \end{aligned}$$

with computation rules:

$$\mathcal{E}_{Either}(A, B, C, f_1, f_2, left(A, B, a)) = f_1(a) : C(left(A, B, a))$$
$$\mathcal{E}_{Either}(A, B, C, f_1, f_2, right(A, B, b)) = f_2(b) : C(right(A, B, b))$$

3 WT-schemata and general subtyping rules

In this section, we define the WT-schemata and the general subtyping rules for those (parameterised) inductive types generated by the WT-schemata. The WT-schemata are those that generate (parameterised) inductive types whose subtyping rules satisfy the weak transitivity requirements.

3.1 WT-schemata

A problem with weak transitivity Weak transitivity does not hold for the subtyping rules for every parameterised inductive types. For example, it fails for the subtyping rules for Σ-types and Π-types. An important observation is that weak transitivity fails for such types because they involve certain form of dependency between parameters. For example, $\Sigma =_{df} [A : Type][B : (A)Type]\mathcal{M}[(x : A)(B(x))X]$ where $B(x)$ is dependent on the objects of parameter A. There are three subtyping rules for Σ-types, two of which are:

$$\frac{\Gamma \vdash A <_c A' : Type \quad \Gamma, x : A \vdash B(x) = B'(c(x)) : Type}{\Gamma \vdash \Sigma(A, B) <_{d_1} \Sigma(A', B') : Type}$$

$$\frac{\Gamma \vdash A <_c A' : Type \quad \Gamma, x : A \vdash B(x) <_{e[x]} B'(c(x)) : Type}{\Gamma \vdash \Sigma(A, B) <_{d_2} \Sigma(A', B') : Type}$$

We can see that the coercion c in the first premise occurs in the second premise. And hence the proof of weak transitivity cannot go through. For instance, in order to prove that $\Sigma(A_1, B_1) < \Sigma(A_2, B_2)$ and $\Sigma(A_2, B_2) < \Sigma(A_3, B_3)$ imply $\Sigma(A_1, B_1) < \Sigma(A_3, B_3)$ (coercions and some other details are omitted here), we would proceed by induction on derivations, and one of the cases is that the last step of the derivations of $\Sigma(A_1, B_1) < \Sigma(A_2, B_2)$ and $\Sigma(A_2, B_2) < \Sigma(A_3, B_3)$ is the first subtyping rule (above) for Σ-types :

$$\frac{A_1 <_{c_1} A_2 \quad x : A_1 \vdash B_1(x) = B_2(c_1(x))}{\Sigma(A_1, B_1) < \Sigma(A_2, B_2)} \quad \frac{A_2 <_{c_2} A_3 \quad y : A_2 \vdash B_2(y) = B_3(c_2(y))}{\Sigma(A_2, B_2) < \Sigma(A_3, B_3)}$$

By induction hypothesis, $A_1 <_{c_3} A_3$ is derivable for some c_3, but c_3 is not (necessarily) computationally equal to $c_2 \circ c_1$. Since $x : A_1 \vdash c_1(x) : A_2$ we have $x : A_1 \vdash B_2(c_1(x)) = B_3(c_2(c_1(x)))$ and hence $x : A_1 \vdash B_1(x) = B_3(c_2(c_1(x)))$. But $x : A_1 \vdash B_1(x) = B_3(c_3(x))$ is not necessarily derivable and how to derive $\Sigma(A_1, B_1) < \Sigma(A_3, B_3)$ becomes a problem of the proof.

In fact, the following example shows that weak transitivity can fail when we combine the subtyping rules for Σ-types and lists (in Section 1), *i.e.*, even if $M_1 <_{e_1} M_2$ and $M_2 <_{e_2} M_3$ are derivable, but $M_1 <_{e_3} M_3$ is not necessarily derivable for any e_3.

Example 2. Assume that we have some type constants A_1, A_2 and A_3, and a constant B_3 of kind $(List(A_3))Type$ (*i.e.*, $B_3 : (List(A_3))Type$), and a WDC $\mathcal{C}$ is generated by the congruence rule ($Cong$) and three coercions $A_1 <_{c_1} A_2$, $A_2 <_{c_2} A_3$ and $A_1 <_{c_2 \circ c_1} A_3$.

By the subtyping rule for lists, we have $List(A_1) <_{d_1} List(A_2)$, $List(A_2) <_{d_2} List(A_3)$ and $List(A_1) <_{d_3} List(A_3)$, where d_1, d_2 and d_3 are defined as in Section 1. Note that d_3 and $d_2 \circ d_1$ are NOT computationally equal.

Since $B_3 \circ d_2 : (List(A_2))Type$, by the first subtyping rule (above) for Σ-types, we have

$$\Sigma(List(A_1), B_3 \circ d_2 \circ d_1) <_{e_1} \Sigma(List(A_2), B_3 \circ d_2)$$

and

$$\Sigma(List(A_2), B_3 \circ d_2) <_{e_2} \Sigma(List(A_3), B_3)$$

We omit the definition of e_1 and e_2 here.

Now, is the judgement $\Sigma(List(A_1), B_3 \circ d_2 \circ d_1) <_{e_3} \Sigma(List(A_3), B_3)$ derivable for some e_3?

The answer is NO. We prove this by contradiction; if it is derivable, it can only be derived from the first subtyping rule (above) for Σ-types (except some steps of the congruence rule). By coherence and Church-Rosser property, we would have $d_3 = d_2 \circ d_1$, *i.e.*, they are computationally equal – a contradiction. Note that the method of the proof of coherence is the same as that in Section 4 and in [LL01].

Weak transitivity schemata The fact that weak transitivity fails for some parameterised inductive types has led us to introduce a restricted form of schemata, WT-schemata, which disallow that a coercion in one premise occurs in a type of another premise.

Definition 5. ***(WT-schemata)*** *Let P be a set of parameters and Θ an inductive schema. Then Θ is a WT-schema* w.r.t. *P if the following is the case:*

- *if $(x : K)M$ is a subterm of Θ and x occurs free in M, then K does not contain any of the parameters in P.*

Remark 5. Obviously, WT-schemata can be defined inductively as done for inductive schemata, but the above definition captures directly the dependency to be excluded.

The above notion of WT-schema covers a large class of parameterised inductive types such as lists, *Maybe* types, *Either* types (disjoint union), function types, product types, types of branching trees, etc. What it excludes are those parameterised types such as Σ-types and Π-types.

3.2 Subtyping rules and coercions

Now, we consider how to define subtyping rules and the associated coercions for any parameterised types (in Γ) generated by the form:

$$\mathcal{T} =_{df} [Y_1 : P_1]...[Y_n : P_n]\mathcal{M}[\overline{\Theta}]$$

where P_i $(i \in \omega)$ are kinds, and $\overline{\Theta} \equiv < \Theta_1, ..., \Theta_m >$ $(m \in \omega)$ is a finite sequence of WT-schemata.

Before we introduce a general form of subtyping rules we give the following examples to demonstrate what the subtyping rules and associated coercions are.

Example 3. 1. We have given the subtyping rule for lists in Section 1. The coercion d_{List} can be defined as follows.

$$\begin{aligned} d_{List} =_{df} \mathcal{E}_{List}(A, &[l : List(A)]List(B), nil(B), \\ &[a : A][l : List(A)][l' : List(B)]cons(B, c(a), l')) \end{aligned}$$

2. Subtyping rules for *Either* types:

$$\frac{\Gamma \vdash A <_{c_1} A' : Type \;\; \Gamma \vdash B = B' : Type}{\Gamma \vdash Either(A, B) <_{d_{Either1}} Either(A', B') : Type}$$

$$\frac{\Gamma \vdash A = A' : Type \;\; \Gamma \vdash B <_{c_2} B' : Type}{\Gamma \vdash Either(A, B) <_{d_{Either2}} Either(A', B') : Type}$$

$$\frac{\Gamma \vdash A <_{c_1} A' : Type \;\; \Gamma \vdash B <_{c_2} B' : Type}{\Gamma \vdash Either(A, B) <_{d_{Either3}} Either(A', B') : Type}$$

where

$$\begin{aligned} d_{Either3} =_{df} \mathcal{E}_{Either}(A, B, &[z : Either(A, B)]Either(A', B'), \\ &[a : A]left(A', B', c_1(a)), [b : B]right(A', B', c_2(b))) \end{aligned}$$

such that $d_{Either3}(left(A, B, a)) = left(A', B', c_1(a))$ and $d_{Either3}(right(A, B, b)) = right(A', B', c_2(b))$. The definitions of $d_{Either1}$ and $d_{Either2}$ are similar to $d_{Either3}$.

3. Subtyping rules for function types:

$$\frac{\Gamma \vdash A' <_{c_1} A : Type \;\; \Gamma \vdash B = B' : Type}{\Gamma \vdash A \to B <_{d_{(\to)1}} A' \to B' : Type}$$

$$\frac{\Gamma \vdash A = A' : Type \;\; \Gamma \vdash B <_{c_2} B' : Type}{\Gamma \vdash A \to B <_{d_{(\to)2}} A' \to B' : Type}$$

$$\frac{\Gamma \vdash A' <_{c_1} A : Type \;\; \Gamma \vdash B <_{c_2} B' : Type}{\Gamma \vdash A \to B <_{d_{(\to)3}} A' \to B' : Type}$$

where

$$d_{(\to)3} =_{df} \mathcal{E}_{(\to)}(A, B, [z : A \to B](A' \to B'), \\ [g : (A)B]\lambda(A', B', c_2 \circ g \circ c_1))$$

such that $d_{(\to)3}(\lambda(A, B, g)) = \lambda(A', B', c_2 \circ g \circ c_1)$. The definitions of $d_{(\to)1}$ and $d_{(\to)2}$ are similar to $d_{(\to)3}$.

The general form of subtyping rules The general form of subtyping rules for the parameterised inductive types is

$$(*) \qquad \frac{premises}{\Gamma \vdash \mathcal{T}(A_1, ..., A_n) <_{d_\mathcal{T}} \mathcal{T}(B_1, ..., B_n) : Type}$$

There can be more than one rule for an parameterised inductive type generated by WT-schemata. In order to find out the premises, we need to give some notational definitions.

Notation: We shall often write $D[\overline{A}]$ for $[A_1/Y_1, ..., A_n/Y_n]D$. Also, we write $\overline{Y} \in FV(M)$ and $\overline{Y} \notin FV(M)$ to mean that 'some of the parameters occur free in M' and 'none of the parameters occurs free in M', respectively.

Definition 6. ***(premise set)***

- *For any small kind K in Γ and with the restrictions in WT-schema, we define $prem_\Gamma(K)$ as follows:*
  1. $K \equiv El(D)$
     (a) *if* $\overline{Y} \notin FV(D)$ *then* $prem_\Gamma(K) = \emptyset$
     (b) *if* $\overline{Y} \in FV(D)$ *then* $prem_\Gamma(K) = \{\Gamma \vdash D[\overline{A}] \leq D[\overline{B}] : Type\}$
  2. $K \equiv (y : K_1)K_2$
     (a) *if* $y \notin FV(K_2)$ *then* $prem_\Gamma(K) = \overline{prem_\Gamma(K_1)} \cup prem_\Gamma(K_2)$, *where*

$$\overline{prem_\Gamma(K_1)} =_{df} \{\Gamma \vdash B \leq A : Type \mid \Gamma \vdash A \leq B : Type \in prem_\Gamma(K_1)\}$$

     (b) *if* $y \in FV(K_2)$ *then* $prem_\Gamma(K) = prem_{\Gamma, y:K_1}(K_2)$. *Note that in this case,* $\overline{Y} \notin FV(K_1)$.
- *For any WT-schema Θ in Γ, we define $prem_\Gamma(\Theta)$ as follows:*
  1. $\Theta \equiv X$, *then* $prem_\Gamma(\Theta) = \emptyset$
  2. $\Theta \equiv (x : K)\Theta_0$
     (a) *if* $x \notin FV(\Theta_0)$ *then* $prem_\Gamma(\Theta) = prem_\Gamma(K) \cup prem_\Gamma(\Theta_0)$
     (b) *if* $x \in FV(\Theta_0)$ *then* $prem_\Gamma(\Theta) = prem_\Gamma(K) \cup prem_{\Gamma, x:K}(\Theta_0)$. *Note that in this case,* $\overline{Y} \notin FV(K)$.
  3. $\Theta \equiv (x : \Phi)\Theta_0$, *then* $prem_\Gamma(\Theta) = \overline{prem_\Gamma(\Phi)} \cup prem_\Gamma(\Theta_0)$
- *For any sequence of WT-schemata in Γ, $\overline{\Theta} \equiv < \Theta_1, ..., \Theta_m >$, we define*

$$prem_\Gamma(\overline{\Theta}) = \cup_{i=1}^{m} prem_\Gamma(\Theta_i)$$

We can now describe how to obtain the premises for the subtyping rules of the form $(*)$. For every $\Gamma \vdash A \leq B : Type \in prem_\Gamma(\overline{\Theta})$, we consider a new coercion c, and one of the following is a premise of a $(*)$-rule:

- $\Gamma \vdash A <_c B : Type$, or
- $\Gamma \vdash A = B : Type$

All of the different combinations give different sequences of premises and hence different rules, except that there must be at least one premise that has the form $\Gamma \vdash A <_c B : Type$.

Remark 6. Some rules have contradictory premises. For example, the subtyping rule for inductive type $\mathcal{T}(A) =_{df} \mathcal{M}[((A)A)X]$ parameterised by type variable A would be

$$\frac{\Gamma \vdash A <_{c_1} B : Type \quad \Gamma \vdash B <_{c_2} A : Type}{\Gamma \vdash \mathcal{T}(A) <_{d_\mathcal{T}} \mathcal{T}(B) : Type}$$

Since the premises in this rule are contradictory (and never satisfied), such rules will never be applied.

In order to give the definition of $d_\mathcal{T}$ in the rule $(*)$, we introduce the following notational definitions.

Definition 7. *We define $\kappa[K]$ for each kind K occurring in a WT-schema Θ of the form $...(x : K)...X$ and in a strictly positive operator Φ of the form $...(x : K)...X$ which occurs in a WT-schema. For the former, $\kappa[K]$ is of kind $(K[\overline{A}])K[\overline{B}]$ and for the latter, it is of kind $(K[\overline{B}])K[\overline{A}]$. The following definition is for the former case and the latter is similar.*

1. $K \equiv El(D)$
   (a) if $\overline{Y} \notin FV(D)$ then $\kappa[K] = id_K = [x : K]x$
   (b) if $\overline{Y} \in FV(D)$ then $\kappa[K] = c$, where c is in the premise of the form $\Gamma \vdash D[\overline{A}] \leq_c D[\overline{B}] : Type$
2. $K \equiv (y : K_1)K_2$
   (a) if $y \notin FV(K_2)$ then by induction, we have $\kappa[K_1] : (K_1[\overline{B}])K_1[\overline{A}]$ and $\kappa[K_2] : (K_2[\overline{A}])K_2[\overline{B}]$, so we define

$$\kappa[K] = [f : K[\overline{A}]][x : K_1[\overline{B}]]\kappa[K_2](f(\kappa[K_1](x)))$$

   (b) if $y \in FV(K_2)$ then $\overline{Y} \notin FV(K_1)$, and by induction, we have $\kappa[K_2] : (K_2[\overline{A}])K_2[\overline{B}]$, so we define $\kappa[K] = [f : K[\overline{A}]][y : K_1]\kappa[K_2](f(y))$.

Definition 8. *Let Θ be a WT-schema and Φ a positive operator.*

- $\Theta[\overline{A}] =_{df} [A_1/Y_1, ..., A_n/Y_n]\Theta$, and $\Theta[\overline{B}] =_{df} [B_1/Y_1, ..., B_n/Y_n]\Theta$, and $\Theta[\overline{B}][\mathcal{T}(\overline{B})] =_{df} [B_1/Y_1, ..., B_n/Y_n, \mathcal{T}(\overline{B})/X]\Theta$
- for any $f : \Phi[\overline{A}][\mathcal{T}(\overline{B})]$, define $\Phi^k[f]$ of kind $\Phi[\overline{B}][\mathcal{T}(\overline{B})]$ as follows:
  1. if $\Phi \equiv X$ then $\Phi^k[f] = f$

2. if $\Phi \equiv (x : K)\Phi_0$ then by Definition 7, we have $\kappa[K] : (K[\overline{B}])K[\overline{A}]$, and define

$$\Phi^k[f] = [x : K[\overline{B}]]\Phi_0^k[f(\kappa[K](x))]$$

- for any $g : \Theta[\overline{B}][\mathcal{T}(\overline{B})]$, define $\Theta^\lambda(g)$ as follows :
  1. if $\Theta \equiv X$ then $\Theta^\lambda(g) = g$
  2. if $\Theta \equiv (x : K)\Theta_0$ then we have $\kappa[K] : (K[\overline{A}])K[\overline{B}]$, and define

$$\Theta^\lambda(g) = [x : K[\overline{A}]]\Theta_0^\lambda(g(\kappa[K](x)))$$

  3. if $\Theta \equiv (x : \Phi)\Theta_0$ then

$$\Theta^\lambda(g) = [x : \Phi[\overline{A}][\mathcal{T}(\overline{A})]][x' : \Phi[\overline{A}][\mathcal{T}(\overline{B})]]\Theta_0^\lambda(g(\Phi^k[x']))$$

Then, we define

$$d_\mathcal{T} =_{df} \mathcal{E}_\mathcal{T}(\overline{A}, C, f_1, ..., f_m)$$

where $C = [z : \mathcal{T}(\overline{A})]\mathcal{T}(\overline{B})$ and $f_j = \Theta_j^\lambda(l_j(\overline{B}))$ $(j = 1, ..., m)$.

The coercion $d_\mathcal{T}$ as defined above sends the canonical objects of $\mathcal{T}(A_1, ..., A_n)$ to the corresponding canonical objects in $\mathcal{T}(B_1, ..., B_n)$, respecting the coercions in the premises. For example, the coercion d_{List} between $List(A)$ and $List(B)$ satisfies that $d_{List}(nil(A)) = nil(B)$ and $d_{List}(cons(A, a, l)) =$ $cons(B, c(a), d_{List}(l))$. The lemma below proves that this is in general the case.

Definition 9. *Let Θ be a WT-schema and assume that Θ be of the form $(x_1 : M_1)...(x_s : M_s)X$ and $x_1, ..., x_s$ are fresh variables. $\Theta^u(\overline{A}, \overline{B})$ is a sequence of arguments:*

1. if $\Theta \equiv X$ then $\Theta^u(\overline{A}, \overline{B}) =< >$
2. if $\Theta \equiv (x_t : K)\Theta_0$ $(t = 1, ..., s)$ then we have $\kappa[K] : (K[\overline{A}])K[\overline{B}]$, and define

$$\Theta^u(\overline{A}, \overline{B}) =< \kappa[K](x_t), \Theta_0^u(\overline{A}, \overline{B}) >$$

3. if $\Theta \equiv (x_t : \Phi)\Theta_0$ $(t = 1, ..., s)$ then $\Theta^u(\overline{A}, \overline{B}) =< \Phi^k[\Phi^\natural[d_\mathcal{T}, x_t]], \Theta_0^u(\overline{A}, \overline{B}) >$

Lemma 1. $d_\mathcal{T}(l_j(\overline{A}, \Theta_j^v)) = l_j(\overline{B}, \Theta_j^u(\overline{A}, \overline{B}))$.

Proof. By the definition of $d_\mathcal{T}$ and the computation rule(in Page 7), we have

$$d_\mathcal{T}(l_j(\overline{A}, \Theta_j^v)) = \mathcal{E}_\mathcal{T}(\overline{A}, C, f_1, ..., f_m, l_j(\overline{A}, \Theta_j^v)) = \Theta_j^\lambda(l_j(\overline{B}))(\Theta_j^\sharp)$$

Now, we need to prove that $\Theta_j^\lambda(l_j(\overline{B}))(\Theta_j^\sharp) = l_j(\overline{B}, \Theta_j^u(\overline{A}, \overline{B}))$. Rather than proving it directly, we generalise the problem first; for any $g : \Theta[\overline{B}][\mathcal{T}(\overline{B})]$, $\Theta_j^\lambda(g)(\Theta_j^\sharp) = g(\Theta_j^u(\overline{A}, \overline{B}))$. This can be proved by induction on the structures of schema. □

4 Coherence and Admissibility of Substitution and Weak Transitivity

In this section, we show that coherence of subtyping rules and admissibility of substitution and weak transitivity hold for the inductive types generated by WT-schemata. Details of the proofs can be found in the forthcoming thesis of the first author.

Note that the set $\mathcal{R}$ of subtyping rules consists of the rule (C) and the subtyping rules for parameterised types and the system $T[\mathcal{R}]_0$ also includes the congruence rule $(Cong)$. Furthermore, we assume that for any judgement $\Gamma \vdash A <_c B : Type \in \mathcal{C}$, neither A nor B is computationally equal to any $\mathcal{T}_j$-type, where $\mathcal{T}_j$ is a type constructor such as $List$ and $Either$, and $\mathcal{T}_i \not\equiv \mathcal{T}_j$ if $i \neq j$ (for example, $\mathcal{T}_1 \equiv List$ and $\mathcal{T}_2 \equiv Either$). We also assume that the original type theory T has good properties, in particular the Church-Rosser property and the property of context replacement by equal kinds.

We denote by $\mathcal{C}_\mathcal{M}$ the set of the derivable subtyping judgements of the form $\Gamma \vdash M <_d M' : Type$ in $T[\mathcal{R}]_0$; that is, $\Gamma \vdash M <_d M' : Type \in \mathcal{C}_\mathcal{M}$ if and only if $\Gamma \vdash M <_d M' : Type$ is derivable in $T[\mathcal{R}]_0$.

Lemma 2. *If $\Gamma \vdash M_1 <_d M_2 : Type \in \mathcal{C}_\mathcal{M}$ then both M_1 and M_2 are computationally equal to a $\mathcal{T}_j$-type (*i.e.*, the normal forms of M_1 and M_2 have same type constructor) or $\Gamma \vdash M_1 <_d M_2 : Type \in \mathcal{C}$.*

Proof. By induction on derivations and because $\mathcal{C}$ is a WDC. □

Theorem 1. *If $\Gamma \vdash M_1 <_d M_2 : Type \in \mathcal{C}_\mathcal{M}$ then $\Gamma \not\vdash M_1 = M_2 : Type$.*

Lemma 3. ***(Context equality and weakening)***

- *If $\Gamma \vdash M_1 <_d M_2 : Type \in \mathcal{C}_\mathcal{M}$ and $\vdash \Gamma = \Gamma'$ then $\Gamma' \vdash M_1 <_d M_2 : Type \in \mathcal{C}_\mathcal{M}$.*
- *If $\Gamma \vdash M_1 <_d M_2 : Type \in \mathcal{C}_\mathcal{M}$, $\Gamma \subseteq \Gamma'$ and Γ' is valid then $\Gamma' \vdash M_1 <_d M_2 : Type \in \mathcal{C}_\mathcal{M}$.*

Theorem 2. ***(Coherence)*** *If $\Gamma \vdash M_1 <_d M_2 : Type \in \mathcal{C}_\mathcal{M}$, $\Gamma \vdash M_1' <_{d'} M_2' : Type \in \mathcal{C}_\mathcal{M}$, $\Gamma \vdash M_1 = M_1' : Type$ and $\Gamma \vdash M_2 = M_2' : Type$ then $\Gamma \vdash d = d' : (M_1)M_2$.*

Proof. By induction on derivations, Lemma 2 and analysing the derivations. □

Theorem 3. ***(Substitution)*** *If $\Gamma, x : K, \Gamma' \vdash M_1 <_d M_2 : Type \in \mathcal{C}_\mathcal{M}$ and $\Gamma \vdash k : K$ then $\Gamma, [k/x]\Gamma' \vdash [k/x]M_1 <_{[k/x]d} [k/x]M_2 : Type \in \mathcal{C}_\mathcal{M}$.*

Proof. By induction on derivations and Lemma 2. □

Theorem 4. ***(Weak Transitivity)*** *If $\Gamma \vdash M_1 <_{d_1} M_2 : Type \in \mathcal{C}_\mathcal{M}$, $\Gamma \vdash M_2' <_{d_2} M_3 : Type \in \mathcal{C}_\mathcal{M}$ and $\Gamma \vdash M_2 = M_2' : Type$ then $\Gamma \vdash M_1 <_{d_3} M_3 : Type \in \mathcal{C}_\mathcal{M}$ for some d_3.*

Proof. By induction on derivations and Lemma 2. □

Remark 7. In [LL01], we use the measure of *depth*, introduced by Chen in his PhD thesis [Che98], to prove the admissibility of transitivity rule. But in this paper, we proceed to the proof of weak transitivity simply by induction on derivations since WT-schemata disallow that a coercion in one premise occurs in another premise.

5 Equality requirement for weak transitivity

In the coercive subtyping framework, a subtyping relation between two types means that there is a (unique) coercion between them. However, such a coercion should not be an arbitrary one; in particular, if $\Gamma \vdash A <_c B$ and $\Gamma \vdash B <_{c'} C$, then the coercion from A to C must be in some sense related to $c' \circ c$, the composition of c and c'. As we have mentioned earlier, in such a case, we require that the coercion from A to C be extensionally equal to $c' \circ c$.

There are choices one may make about this notion of extensional equality. First, note that, although we have considered coercive subtyping in intensional type theories, the equality requirement for weak transitivity can be considered to be at the meta-level, and hence can be outside the intensional type theory. One of such choices, that we adopt here, is the notion of equality in extensional type theory [ML84].

In an extensional type theory, one has the following rule

$$\frac{\Gamma \vdash q : Eq(A, a, b)}{\Gamma \vdash a = b : A}$$

where A is a type, a and b are objects of type A, Eq is the propositional equality (Martin-Löf's equality type or the Leibniz equality), and $=$ is the judgemental equality. One can consider an extension of the intensional type theory (which has Eq-types) by the above rule to obtain the corresponding extensional theory. Note that the above rule makes the resulting type theory undecidable and loses its property of strong normalisation. However, it does capture the notion of extensional equality in a strong sense.

Notation Let K be a kind of the form $(x : K_0)El(A)$ and k and k' be objects of kind K. Then $\Gamma \vdash k =_{ext} k' : K$ stands for the judgement $\Gamma, x : K_0 \vdash k(x) = k'(x) : A$ in the extensional type theory.

We can now use the above notion of extensional equality and the associated notion just introduced to express our equality requirement about weak transitivity.

- **Equality requirement:** If $\Gamma \vdash A <_c B : Type$, $\Gamma \vdash B <_{c'} C : Type$, and $\Gamma \vdash A <_{c''} C : Type$, then $\Gamma \vdash c'' =_{ext} c' \circ c : (A)C$.

The following theorem says that the equality requirement is satisfied by the general subtyping rules for parameterised inductive types as defined in Section 3.

Theorem 5. *(equality requirement) If* $\Gamma \vdash A <_c B : Type$, $\Gamma \vdash B <_{c'} C : Type$ *and* $\Gamma \vdash A <_{c''} C : Type$ *and are all in* $\mathcal{C}_{\mathcal{M}}$, *then* $\Gamma \vdash c'' =_{ext} c' \circ c : (A)C$.

Proof. By induction on derivations, Lemma 2 and Lemma 1. □

6 Discussion and related work

Extension of WT-schemata One may extend WT-schemata so that some families of inductive types can also be covered. For example, the type of vectors is defined as follows:

$$Vec =_{df} [A : Type]\mathcal{M}[X(0), (n : N)(A)(X(n))X(S(n))]$$

where X is a place holder of kind $(N)Type$, N is the type of natural numbers, 0 and S are constructors for zero and the successor respectively (see [Luo94] for more details). A common subtyping rule for vectors is the following:

$$\frac{\Gamma \vdash n : N \quad \Gamma \vdash A <_c B : Type}{\Gamma \vdash Vec(A, n) <_{d(n)} Vec(B, n) : Type}$$

where

$$d(0, vnil(A)) = vnil(B)$$
$$d(S(m), vcons(A, m, a, l)) = vcons(B, m, c(a), d(m, l))$$

and $vnil$ and $vcons$ are the constructors of vectors introduced as usual.

Adding this subtyping rule into $\mathcal{R}$, all the good properties are kept, *i.e.*, $\mathcal{R}$ is still coherent, substitution rule is admissible, weak transitivity holds and equality requirement is satisfied. Note that Vec is a dependent family. As mentioned in Section 3, WT-schemata avoid the kind of dependency between parameters such as that for Σ-types to make sure that there is no coercion in one premise that occurs in another premise. The above subtyping rule for vectors does not have such dependency.

Decidability Since we have proven coherence and admissibility of substitution and weak transitivity, and given a precise definition of coercions, we can give a sound and complete algorithm to implement coercive subtyping with weak transitivity and we can be sure that coercion searching in $\mathcal{C}_{\mathcal{M}}$ is decidable if it is decidable in $\mathcal{C}$.

Combining coercions As we pointed out, weak transitivity fails for some parameterised inductive types such as dependent Σ-types which involve certain form of dependency between parameters. For instance, if we combine the subtyping rules for Σ-types and lists, weak transitivity fails in the sense that the Theorem 4 fails to hold. An interesting issue is to study how one may combine different coercion schemes so that they can be used together. A step to this direction is studied in a paper in preparation, concerning how to combine component-wise subtyping rules for Σ-types with the subtyping rule of first projection.

Extensionality vs intensionality The issue of transitivity (and in particular, the weak transitivity) is studied in *intensional* type theories. An interesting issue to be studied is how transitivity (and subtyping in general) works in extensional type theories. Although extensional type theories are undecidable and arguably not suitable for implementation or practical use, and we are aware of some technical difficulties to work in an extensional type theory, to study coercive subtyping and its related issues in an extensional framework may provide further theoretical insights.

Although we have been working in a specific framework, the issues we are studying (e.g., transitivity in intensional theories) have a wider impact on similar studies in different frameworks.

Related work The early development of the framework of coercive subtyping is closely related to Aczel's idea in type-checking overloading methods for classes [Acz94] and the work on giving coercion semantics to lambda calculi with subtyping by Breazu-Tannen et al [BCGS91]. Bailey, Saibi, and Callaghan's respective implementations of coercions in the proof systems Lego, Coq and Plastic are important contributions [Bai98,Sai97,CL01]. Barthe and his colleagues have studied constructor subtyping and its possible applications in proof systems [BF99,BvR00]. A recent logical study of subtyping in system F can be found in [LMS95] and Chen has studied the issue of transitivity elimination in that framework [Che98]. One of Chen's proof methods was used in one of our earlier papers [LL01] to prove the admissibility of transitivity in the framework of coercive subtyping.

References

[Acz94] P. Aczel. Simple overloading for type theories. Draft, 1994.

[B+00] B. Barras et al. *The Coq Proof Assistant Reference Manual (Version 6.3.1).* INRIA-Rocquencourt, 2000.

[Bai98] A. Bailey. *The Machine-checked Literate Formalisation of Algebra in Type Theory.* PhD thesis, University of Manchester, 1998.

[BCGS91] V. Breazu-Tannen, T. Coquand, C. Gunter, and A. Scedrov. Inheritance and explicit coercion. *Information and Computation*, 93, 1991.

[BF99] G. Barthe and M.J. Frade. Constructor subtyping. *Proceedings of ESOP'99, LNCS 1576*, 1999.

[BvR00] G. Barthe and F. van Raamsdonk. Constructor subtyping in the calculus of inductive constructions. *Proceedings of FOSSACS'00, LNCS 1784*, 2000.

[Che98] G. Chen. *Subtyping, Type Conversion and Transitivity Elimination.* PhD thesis, University of Paris VII, 1998.

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

[CLP01] P. C. Callaghan, Z. Luo, and J. Pang. Object languages in a type-theoretic meta-framework. *Workshop of Proof Transformation and Presentation and Proof Complexities (PTP'01)*, 2001.

[HHP87] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Proc. 2nd Ann. Symp. on Logic in Computer Science. IEEE*, 1987.

[JLS98] A. Jones, Z. Luo, and S. Soloviev. Some proof-theoretic and algorithmic aspects of coercive subtyping. *Types for proofs and programs (eds, E. Gimenez and C. Paulin-Mohring), Proc. of the Inter. Conf. TYPES'96, LNCS 1512*, 1998.

[LC98] Z. Luo and P. Callaghan. Coercive subtyping and lexical semantics (extended abstract). *LACL'98*, 1998.

[LL01] Y. Luo and Z. Luo. Coherence and transitivity in coercive subtyping. In R. Nieuwenhuis and A. Voronkov, editors, *8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2250 of *LNAI*, pages 249–265. Springer-Verlag, 2001.

[LMS95] G. Longo, K. Milsted, and S. Soloviev. A logic of subtyping. In *Proc. of LICS'95*, 1995.

[LP92] Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Department of Computer Science, University of Edinburgh, 1992.

[LS99] Z. Luo and S. Soloviev. Dependent coercions. *The 8th Inter. Conf. on Category Theory and Computer Science (CTCS'99), Edinburgh, Scotland. Electronic Notes in Theoretical Computer Science*, 29, 1999.

[Luo94] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Oxford University Press, 1994.

[Luo97] Z. Luo. Coercive subtyping in type theory. *Proc. of CSL'96, the 1996 Annual Conference of the European Association for Computer Science Logic, Utrecht. LNCS 1258*, 1997.

[Luo99] Z. Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(1):105–130, 1999.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

[NPS90] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford University Press, 1990.

[Sai97] A. Saibi. Typing algorithm in type theory with inheritance. *Proc of POPL'97*, 1997.

[SL02] S. Soloviev and Z. Luo. Coercion completion and conservativity in coercive subtyping. Annals of Pure and Applied Logic, 2002.

Appendix A: The following gives the rules of the logical framework LF.

Contexts and assumptions

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

Equality rules

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

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

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

Substitution rules

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

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

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

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

The kind Type

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

Dependent product kinds

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

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

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

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

Appendix B: The following are the inference rules for the coercive subkinding extension $T[\mathcal{R}]$ (not including the rules for subtyping).

New rules for application

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

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

Coercive definition rule

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

Basic subkinding rule

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

Subkinding for dependent product kinds

$$\frac{\Gamma \vdash K_1' = K_1 \ \Gamma, x' : K_1' \vdash K_2 <_c K_2'}{\Gamma \vdash (x : K_1)K_2 <_{[f:(x:K_1)K_2][x':K_1']c(f(x'))} (x' : K_1')K_2'}$$

Congruence rule for subkinding

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

Substitution rule for subkinding

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