CHALMERS
UNIVERSITY OF TECHNOLOGY

Structural subtyping for inductive types with functorial equality rules

Math. Struct. in Comp. Science (2008), *vol.* 18, *pp.* 931–972.
doi:10.1017/S0960129508006956 Printed in the United Kingdom

Structural subtyping for inductive types with functorial equality rules[†]

ZHAOHUI LUO[‡] and ROBIN ADAMS[§]

Department of Computer Science,
Royal Holloway, University of London,
Egham, Surrey TW20 0EX, United Kingdom
[‡]*Email:* zhaohui@cs.rhul.ac.uk
[§]*Email:* robin@cs.rhul.ac.uk

Received 27 February 2006; revised 21 December 2006

In this paper we study subtyping for inductive types in dependent type theories in the framework of coercive subtyping. General structural subtyping rules for parameterised inductive types are formulated based on the notion of inductive schemata. Certain extensional equality rules play an important role in proving some of the crucial properties of the type system with these subtyping rules. In particular, it is shown that the structural subtyping rules are coherent and that transitivity is admissible in the presence of the functorial rules of computational equality.

1. Introduction

Coercive subtyping (Luo 1997; 1999) is a general framework for subtyping and abbreviation in dependent type theories. In particular, it incorporates subtyping for inductive types. In this paper, structural subtyping for inductive types is studied in the framework of coercive subtyping.

Inductive types in dependent type theories have a rich structure and include many interesting types, such as the types of natural numbers, lists, trees, ordinals and dependent pairs. In general, inductive types can be considered as generated by inductive schemata (Dybjer 1991; Paulin-Mohring 1993; Luo 1994), and this gives us a clear guide as to how to consider natural subtyping rules for the inductive types.

The idea of deriving the general structural subtyping rules from the inductive schemata was first considered in Luo (1997) and the subtyping rules for typical inductive types were given in Luo (1999). However, from the Ph.D. work of Y. Luo (Luo 2005), it has become clear that in an intensional type theory, the structural subtyping rules for some of the inductive types are not compatible with the notion of transitivity (sometimes called 'strong transitivity') as given by the following rule in coercive subtyping:

$$\text{(Trans)} \qquad \frac{A <_c B \qquad B <_{c'} C}{A <_{c' \circ c} C},$$

[†] This work is partially supported by the following research grants: UK EPSRC grant GR/R84092, Leverhulme Trust grant F/07 537/AA and EU TYPES grant 510996.

which says that the composition $c' \circ c$ of two coercions $c : (A)B$ and $c' : (B)C$ as functional operations is also a coercion. In particular, in an intensional type theory, the above transitivity rule is not admissible in the presence of the structural subtyping rules for many inductive types, although it is admissible for some (Luo and Luo 2001). For example, consider the type of lists, $List(A)$, parameterised by an element type A. Its subtyping rule is

$$\frac{A <_c B}{List(A) <_{map(c)} List(B)}$$

where $map(c) : (List(A))List(B)$ is the usual map operation on lists defined by induction. With such a subtyping rule, the above rule (`Trans`) of transitivity is not admissible in an intensional type theory: when $A <_c B$ and $B <_{c'} C$, the following two functional operations from $List(A)$ to $List(C)$ are not computationally equal (although they are extensionally equal):

— $map(c' \circ c)$, obtainable as a coercion by first applying the transitivity rule (`Trans`) and then the above subtyping rule for lists; and

— $map(c') \circ map(c)$, obtainable as a coercion by first applying the above subtyping rule for lists (twice) and then the transitivity rule (`Trans`).

Hence, if we added the transitivity rule (`Trans`), there would be more than one coercion between two types – coherence would fail.

With the motivation of dealing with this problem, a notion of 'weak transitivity' has been proposed and studied (Luo and Luo 2005; Luo 2005; Luo *et al.* 2002). Unlike the above strong notion of transitivity, weak transitivity only requires that there be a coercion that is extensionally equal to the composition and does not require that the composition itself be a coercion. Weak transitivity is admissible for intensional type theories with structural subtyping for a large class of inductive types including that of lists. However, it turns out that this is not the case for some inductive types that have a certain form of dependency between their parameters. Such inductive types include, for example, Σ-types of dependent pairs and Π-types of dependent functions. A notion of WT-schema has been developed to capture such a dependency of parameters, or more precisely, the inductive types generated by the WT-schemata exclude exactly those types that cause problems for the admissibility of weak transitivity. The structural subtyping rules for inductive types generated by WT-schemata has been formulated and weak transitivity proved to be admissible in intensional type theories with structural subtyping for such inductive types. (See Luo and Luo (2005) for more details.)

From this, it is fair to say that the introduction of the notion of weak transitivity has not solved the above problem completely – for the types uncovered by WT-schemata, we still cannot introduce their structural subtyping rules in an adequate way for transitivity to be admissible. In this paper we consider another approach: instead of introducing a weaker notion of transitivity, we consider stronger equality rules. For instance, we consider by brute force in some way that $map(c') \circ map(c)$ and $map(c' \circ c)$ are judgementally equal. In other words, we extend the underlying type theory by certain extensional rules for the computational equality. These equality rules express the functorial laws and will be called

χ-rules in this paper[†]. We shall show that in the presence of the χ-rules, the transitivity rule (`Trans`) is admissible for the type system with structural subtyping of all of the inductive types.

As we are considering inductive types generated by any inductive schema (rather than just those from the subclass generated by the WT-schemata (Luo and Luo 2005)), a more general formulation of the subtyping rules is called for that takes care of the dependency between parameters for some of the inductive types such as Σ-types and Π-types. The new formulation inserts coercions in the correct positions of the premises of the subtyping rules. This formulation is also used as the basis for the χ-rules and for proving results such as those of coherence and admissibility of transitivity.

During the proofs of the properties of the structural subtyping rules, we use the simple but important property that the inductive type constructors are injective. For instance, for types of lists, if $List(A) = List(B)$, then $A = B$. It has not been shown whether this holds for a type theory with extensional rules such as the χ-rules, as its meta-theoretic properties such as Church–Rosser are largely unknown. A direct proof of this injectivity result in the logical framework LF (Luo 1994), which is used as a basis for formulating coercive subtyping, would require us to analyse difficult cases involving λ-abstractions. To avoid this difficulty, we prove injectivity in the lambda-free logical framework TF (Aczel 2001; Adams 2004), and then lift it to LF by using the fact that LF is a conservative extension of TF.

In contrast to an earlier proof of a more restricted result in Luo and Luo (2001), the proof of transitivity elimination is done directly by induction on derivations, but by proving the admissibility of a more general rule. This has overcome a difficulty in the earlier research effort and avoided the use of a special measure developed in Chen (1998).

In Section 2, we briefly introduce the logical frameworks LF and TF, the notion of inductive types as generated by inductive schemata, and the framework of coercive subtyping. The formulations of the structural subtyping rules and the corresponding functorial equality rules are given in Section 3. Section 4 shows that LF is a conservative extension of TF and proves the injectivity result. Proofs of properties such as coherence and admissibility of transitivity are given in Section 5. Future and related work are discussed with conclusions in Section 6.

2. Logical frameworks, inductive types and coercive subtyping

In this section we give a brief introduction to the logical frameworks LF and TF, inductive types and coercive subtyping, partly for background and to establish notational conventions.

2.1. *The logical frameworks LF and TF*

We shall introduce logical frameworks LF and TF, with more details for the latter as it is less well known. Technically, in this paper, the logical framework TF is only used in the

[†] The name comes from Barral *et al.* (2005), where such equality rules are considered in a simply-typed λ-calculus with inductive types.

proof of the injectivity result in Section 4. An understanding of its formal details is only required for the proofs in Section 4.

2.1.1. *The logical framework LF* LF (Luo 1994) is a dependent type system that can be used to specify type theories such as Martin-Löf's type theory (Nordström *et al.* 1990) and UTT (Luo 1994). It is a typed version of Martin-Löf's logical framework (see Nordström *et al.* (1990, Chapter 19) for a presentation of the latter)[†].

A presentation of LF and discussions on how it should be used for specifying type theories can be found in Luo (1994, Chapter 9) or Luo (1999, Section 2). Figure 1 gives the inference rules of LF, which include the general rules for contexts, equality and substitution, and the rules for the following kinds:

— $Type$: the kind representing the conceptual universe of types;
— $El(A)$: the kind of objects of type A; and
— $(x{:}K)K'$: the kind representing the dependent product with functional operations f (such as the abstraction $[x{:}K]k'$) as objects that can be applied to objects k of kind K to form application $f(k)$ or simply fk.

Functional operations of kind $(x{:}K)K'$ are equal if they are $\beta\eta$-equal. A kind is *small* if it is of the form $El(A)$ or $(x{:}K)K'$ such that K and K' are small. (Put another way, a kind is small if it does not contain $Type$.)

LF and the type theories specified in LF satisfy many nice properties. The following is a proposition that we shall use in this paper, the proof of which can be found in Adams (2004).

Theorem 2.1 (Subject reduction for β-reduction). Let T be a type theory specified in LF. If $\Gamma \vdash k : K$ is derivable in T and $k \to^*_\beta k'$, then $\Gamma \vdash k = k' : K$ is derivable in T.

Notation. We shall adopt the following notational conventions:

— We often omit El to write A for $El(A)$ and may write $(K_1)K_2$ for $(x{:}K_1)K_2$ when x does not occur free in K_2.
— *Equality signs*: We shall use $M \equiv N$ for syntactic equality between terms, meaning that M and N are the same terms up to α-conversion, and use $M = N$ for definitional or computational equality between terms.
— *Substitution*: We sometimes use $M[x]$ to indicate that variable x may occur free in M, and subsequently write $M[N]$ for the substitution $[N/x]M$ when no confusion may occur.
— *Functional composition*: $g \circ f =_{\mathrm{df}} [x{:}K_1]g(f(x)) : (K_1)K_3$, where $f : (K_1)K_2$ and $g : (K_2)K_3$ and x does not occur free in f or g.

We shall also use the following notation for sequences and sequence operations:

— We shall use $\bar{a}$ to represent a sequence; for instance, $\bar{a} \equiv a_1, ..., a_n$ or $\bar{a} \equiv \langle a_1, ..., a_n \rangle$.

[†] The difference between the logical frameworks considered in this paper and the Edinburgh LF (Harper *et al.* 1987; 1993) is that the former LFs are intended to be used to specify computation rules and hence type theories, while the latter is not.

Contexts and assumptions

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

General equality rules

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

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

Equality typing rules

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

Substitution rules

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

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

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

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

The kind Type

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

Dependent product kinds

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

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

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

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

Fig. 1. Inference rules of LF

— Let $\bar{a} \equiv a_1, ..., a_n$, $\bar{b} \equiv b_1, ..., b_n$ and $\bar{A} \equiv A_1, A_2[x_1], ..., A_n[x_1, ..., x_{n-1}]$.

- We write $\bar{a} :: \bar{A}$ for the sequence $a_1 : A_1,\ a_2 : A_2[a_1],\ ...,\ a_n : A_n[a_1, ..., a_{n-1}]$.
- We write $\Gamma \vdash \bar{a} :: \bar{A}$ for the sequence of judgements $\Gamma \vdash a_1 : A_1$, $\Gamma \vdash a_2 : A_2[a_1]$, ..., $\Gamma \vdash a_n : A_n[a_1, ..., a_{n-1}]$.
- We write $\Gamma \vdash \bar{a} = \bar{b} :: \bar{A}$ for the sequence of judgements $\Gamma \vdash a_1 = b_1 : A_1$, $\Gamma \vdash a_2 = b_2 : A_2[a_1]$, ..., $\Gamma \vdash a_n = b_n : A_n[a_1, ..., a_{n-1}]$.

— Let $\bar{x}$ be a sequence of variables.
  – $\bar{x} \in M$ means that *some* of the variables in $\bar{x}$ occur in M.
  – $\bar{x} \notin M$ means that *none* of the variables in $\bar{x}$ occur in M.
— We shall use $\hat{}$ for concatenation of sequences and :: for the operations attaching an element both to the head and to the tail of a sequence.

2.1.2. *The lambda-free framework TF* The logical framework TF (TF stands for 'Type Framework') was first conceived by Aczel (Aczel 2001), and its theory was developed in Adams (2004). It follows from the general results proved in Adams (2004) that LF can be considered a conservative extension of TF – this is a key property that will allow us to lift the injectivity result from TF to LF in Section 4.

TF is lambda free in the sense that the entities such as lambda-abstractions and Π-kinds are not first-class, nor is β-reduction. Its syntax is organised by *arities*, which are defined by the grammar

$$\alpha :: = (\alpha, \ldots, \alpha),$$

with the arity () in the base case (also denoted by **0**), the arity of terms and types in a specified object type theory. The intuition behind arities is that an object of arity $\alpha = (\alpha_1, \ldots, \alpha_n)$, an *$\alpha$-ary object*, is a function that takes n arguments – namely an α_1-ary object, ..., and an α_n-ary object – and returns a term or type of the object theory, the type theory specified in TF.

Objects in TF are in normal form in the traditional sense. In general, they have the form $[\bar{x}]z\bar{M}$, where z is either a variable or a constant. These are exactly the lambda terms that are in β-normal and η-long form. More formally, every variable or constant is associated with an arity and the *α-ary objects* for each arity α can be defined inductively as follows:

— If z is a variable or constant of arity $(\alpha_1, \ldots, \alpha_n)$, and M_i is an α_i-ary object, then $zM_1 ... M_n$ is a **0**-ary object.
— If N is a $(\beta_1, ..., \beta_n)$-ary object and x an α-ary variable, then $[x]N$ is an $(\alpha, \beta_1, ..., \beta_n)$-ary object.

Kinds and contexts are assigned arities as well (for example, $Type$ and $El(A)$ are the only kinds assigned arity **0**). Arities are respected in forming kinds and contexts. For instance, in any entry $x{:}K$ in a context, x and K must have the same arity.

Notation. We shall write z_η for the η-long form of a variable or constant z. More formally, if z is an $(\alpha_1, \ldots, \alpha_n)$-ary variable or constant, then $z_\eta \equiv [y_1, \ldots, y_n]z(y_1)_\eta ... (y_n)_\eta$, where each y_i is a fresh α_i-ary variable.

Instantiation and *employment* are crucial notions in TF and correspond to substitution and application, respectively. In TF, the result of a substitution $[M/x]N$ is, in general, not an object, and the application MN is not an object, either. The intention is that

the instantiation $\{M/x\}N$ is the β-normal and η-long form of $[M/x]N$, and that the employment $M\{N\}$ is the β-normal and η-long form of MN.

Definition 2.2 (Instantiation and employment). The two operations

— *Instantiation*, $\{M/x\}N$, resulting in a β-ary object by instantiating x with M in N, for any α-ary object M, α-ary variable x and β-ary object N, and
— *Employment*, $M\{N\}$, resulting in a $(\beta_1, ..., \beta_n)$-ary object by employing M on N, for any $(\alpha, \beta_1, ..., \beta_n)$-ary object M and α-ary object N,

are defined by mutual induction on the arity α and the structure of N, as follows:

$$\begin{aligned}
\{M/x\}zN_1 \cdots N_n &\equiv z(\{M/x\}N_1) \cdots (\{M/x\}N_n) \qquad (z \not\equiv x)\\
\{M/x\}xN_1 \cdots N_n &\equiv M\{\{M/x\}N_1\} \cdots \{\{M/x\}N_n\}\\
\{M/x\}[y]N &\equiv [y]\{M/x\}N\\
([x]M)\{N\} &\equiv \{N/x\}M\,.
\end{aligned}$$

We extend the operation of instantiation to kinds in the obvious manner.

There are only three primitive judgement forms in TF:

$$\Gamma \text{ valid} \qquad \Gamma \vdash M : T \qquad \Gamma \vdash M = N : T$$

where M and N are $\mathbf{0}$-ary objects and T is a $\mathbf{0}$-ary kind (that is, T is either $Type$ or of the form $El(A)$). Note that this condition on the arities of M, N and T justifies our assertion that the objects of higher arity are not first-class entities in TF. The judgements themselves only deal directly with the terms and types of the specified theory. The objects of higher arity can only occur in contexts or within an object of arity $\mathbf{0}$.

Having given the forms of primitive judgements, we can now introduce some *defined* judgement forms, each of which denotes a sequence of primitive judgements. The defined judgements help us in presenting and discussing the TF system. Some examples, to be used later in the paper, are as follows.

— Judgement $\Gamma \vdash K\ kind$ can be defined by considering the following two cases:
  – $K \equiv (\bar{x} :: \bar{K})Type$.
    Then $\Gamma \vdash K\ kind$ stands for $\Gamma, \bar{x} :: \bar{K}\ valid$.
  – $K \equiv (\bar{x} :: \bar{K})El(A)$.
    Then $\Gamma \vdash K\ kind$ stands for $\Gamma, \bar{x} :: \bar{K} \vdash A : Type$.
— Judgement $\Gamma \vdash [\bar{x}]M : (\bar{x}{:}\bar{K})T$ is defined to be the judgement $\Gamma, \bar{x}{:}\bar{K} \vdash M : T$.
— Judgement $\Gamma \vdash \bar{M} :: \bar{K}$ is defined to be the following sequence of judgements:

$$\Gamma \vdash M_1 : K_1, \quad \Gamma \vdash M_2 : \{M_1/x_1\}K_2, \quad ..., \quad \Gamma \vdash M_n : \{M_1/x_1, ..., M_{n-1}/x_{n-1}\}K_{n-1}\,.$$

A judgement form $\Gamma \vdash \bar{M} = \bar{N} : \bar{K}$ can be defined similarly.

The inference rules of TF are given in Figure 2, where T, M, N and P are all of arity $\mathbf{0}$. Note that the number of rules is rather small, mainly due to the fact that TF only deals with terms in β-normal and η-long form.

Some of the basic properties of TF are given in the following proposition, whose proofs can be found in Adams (2004).

Contexts

$$\frac{}{\langle\rangle \text{ valid}} \qquad \frac{\Gamma \vdash K \ kind \quad x \notin FV(\Gamma)}{\Gamma, x{:}K \text{ valid}}$$

Variables and constants ($z : (\bar{y}{::}\bar{K})T$ is either in Γ or declared)

$$\frac{\Gamma \vdash \bar{M} :: \bar{K}}{\Gamma \vdash z\bar{M} : \{\bar{M}/\bar{y}\}T} \qquad (CA)\ \frac{\Gamma \vdash \bar{M} = \bar{N} :: \bar{K}}{\Gamma \vdash z\bar{M} = z\bar{N} : \{\bar{M}/\bar{y}\}T}$$

General equality rules

$$(R)\ \frac{\Gamma \vdash M : T}{\Gamma \vdash M = M : T} \qquad (S)\ \frac{\Gamma \vdash M = N : T}{\Gamma \vdash N = M : T} \qquad (T)\ \frac{\Gamma \vdash M = N : T \quad \Gamma \vdash N = P : T}{\Gamma \vdash M = P : T}$$

$$\frac{\Gamma \vdash M : El(A) \quad \Gamma \vdash A = B : Type}{\Gamma \vdash M : El(B)} \qquad \frac{\Gamma \vdash M = N : El(A) \quad \Gamma \vdash A = B : Type}{\Gamma \vdash M = N : El(B)}$$

Fig. 2. Inference rules of TF

Proposition 2.3 (Properties of TF).

1 *Basic properties of instantiation*:
   (a) For any object M, we have $\{x_\eta/x\}M \equiv M$.
   (b) If y is not free in M, then $\{M/x\}\{N/y\}P \equiv \{\{M/x\}N/y\}\{M/x\}P$.
   (c) For any object M of arity $(\alpha, \beta_1, ..., \beta_n)$, we have $[x](M\{x_\eta\}) \equiv M$, where x has arity α.
   (d) For any object M, we have $\{M/x\}x_\eta \equiv M$.
   (e) $\{M/x\}(N\{P\}) \equiv (\{M/x\}N)\{\{M/x\}P\}$.

2 *Context validity*:
Any derivation of $\Gamma_1, x{:}K, \Gamma_2 \vdash J$ has a subderivation of $\Gamma_1 \vdash K \ kind$.

3 *Weakening*:
If $\Gamma \vdash J$, $\Gamma \subseteq \Delta$ and Δ valid, then $\Delta \vdash J$.

4 *Instantiation*:
If $\Gamma, x{:}K, \Delta \vdash J$ and $\Gamma \vdash M{:}K$, then $\Gamma, \{M/x\}\Delta \vdash \{M/x\}J$.

5 *Functionality*:
If $\Gamma, x{:}K, \Delta \vdash P{:}K'$ and $\Gamma \vdash M = N{:}K$, then

$$\Gamma, \{M/x\}\Delta \vdash \{M/x\}P = \{N/x\}P : \{M/x\}K'.$$

6 *Equation validity*:
   (a) If $\Gamma \vdash M = N{:}K$, then $\Gamma \vdash M{:}K$ and $\Gamma \vdash N{:}K$.
   (b) If $\Gamma \vdash K = K'$, then $\Gamma \vdash K \ kind$ and $\Gamma \vdash K' \ kind$.

7 *Context conversion*:
If $\Gamma, x{:}K, \Delta \vdash J$ and $\Gamma \vdash K = K'$, then $\Gamma, x{:}K', \Delta \vdash J$.

8 *Kind validity*:
If $\Gamma \vdash M{:}K$ or $\Gamma \vdash M = N{:}K$, then $\Gamma \vdash K \ kind$.

As in LF, type theories can be specified in TF by declaring a number of constants and computation rules. The declaration of a constant, where T is of arity $\mathbf{0}$,

$$c : (\bar{x} :: \bar{K})T$$

has the effect of adding the following two rules:

$$\frac{\Gamma \vdash \bar{M} :: \bar{K}}{\Gamma \vdash c\bar{M} : \{\bar{M}/\bar{x}\}T} \qquad \frac{\Gamma \vdash \bar{M} = \bar{N} :: \bar{K}}{\Gamma \vdash c\bar{M} = c\bar{N} : \{\bar{M}/\bar{x}\}T}$$

The declaration of a computation rule of the form

$$k = k' : T \quad \text{for} \quad \bar{x} : \bar{K},$$

where k, k' and T are all of arity $\mathbf{0}$, has the effect of adding the following rule:

$$\frac{\Gamma \vdash \bar{M} :: \bar{K}}{\Gamma \vdash \{\bar{M}/\bar{x}\}k = \{\bar{M}/\bar{x}\}k' : \{\bar{M}/\bar{x}\}T} \ .$$

Remark. This way of declaring type theories in TF is similar to the way type theories are declared in LF (*c.f.*, Luo (1994, Section 9.1.2)). The restriction disallowing declarations of computation rules between objects of higher arities conforms with that imposed in Luo (1999) for LF, that is, that there should be no declarations of equalities between objects of a dependent product kind.

2.2. *Inductive types*

Inductive types are generated by inductive schemata (Dybjer 1991; Paulin-Mohring 1993; Luo 1994) and can be parameterised. We just give a brief introduction here; for more details, see, for example, Luo (1994, Chapter 9).

Intuitively, an inductive type is introduced by specifying its *constructors* (or introduction operators) and their kinds. The values (or canonical objects) of the inductive types are generated by these constructors. For instance, one can introduce the inductive type of lists of natural numbers, $List(N)$, by giving its constructors $nil(N)$ and $cons(N)$, which are of kinds $List(N)$ and $(n{:}N)(l{:}List(N))List(N)$, respectively. The values of $List(N)$ are either the empty list $nil(N)$ or of the form $cons(N, n, l)$.

Formally, the types of these constructors are given by *inductive schemata*, as given by the following definition.

Definition 2.4 (Strictly positive operator and inductive schema). Let Γ be a valid context and X be a type variable such that X does not occur free in Γ.

— A *strictly positive operator* Φ in Γ with respect to X is of one of the following forms:

  1 $\Phi \equiv X$

  2 $\Phi \equiv (x{:}K)\Phi_0$, where K is a small kind in Γ, and Φ_0 is a strictly positive operator in $\Gamma, x{:}K$ with respect to X.

— An *inductive schema* Θ in Γ with respect to X is of one of the following forms:

  1 $\Theta \equiv X$

2 $\Theta \equiv (x{:}K)\Theta_0$, where K is a small kind in Γ, and Θ_0 is an inductive schema in $\Gamma, x{:}K$ with respect to X

3 $\Theta \equiv (\Phi)\Theta_0$, where Φ is a strictly positive operator in Γ with respect to X, and Θ_0 is an inductive schema in Γ with respect to X.

In this paper, unless stated otherwise, all of the strictly positive operators and inductive schemata are with respect to X.

Remark. Note that strictly positive operators are special cases of inductive schemata. Also, the above notions of strictly positive operator and inductive schema can also be defined by the BNF notation as follows, where K stands for any small kind in which X does not occur free:

$$\Phi ::= X \mid (x{:}K)\Phi$$
$$\Theta ::= X \mid (x{:}K)\Theta \mid (\Phi)\Theta .$$

A strictly positive operator (or an inductive schema) in the context Γ with respect to X is then a kind in $\Gamma, X{:}Type$ that is generated by the above grammar.

In a valid context, any finite sequence of inductive schemata generates an inductive type, with its introduction, elimination and computation rules. In this paper, we shall consider inductive types $\mathscr{T}$ parameterised by a sequence of variables $\bar{Y}$ such that $\bar{Y} :: \bar{P}$, and generated by a sequence of schemata $\bar{\Theta} \equiv \langle \Theta_1, ..., \Theta_m \rangle$ $(m \in \omega)$ in $\bar{Y} :: \bar{P}$. For example, the parameterised inductive type of lists $List(A)$ is parameterised by type A and generated by the schemata X and $(A)(X)X$.

Notation. For any strictly positive operator Φ and inductive schema Θ with respect to X parameterised by $\bar{Y}$, we use $\Phi[\bar{A}, B]$ and $\Theta[\bar{A}, B]$ to stand for the substitutions $[\bar{A}/\bar{Y}, B/X]\Phi$ and $[\bar{A}/\bar{Y}, B/X]\Theta$, respectively. To generalise this notation further, if X does not occur free in $K[\bar{Y}]$, then $K[\bar{A}, B]$ is just $K[\bar{A}]$.

In LF, the parameterised inductive type $\mathscr{T}$ can be introduced by declaring the constant expressions:

— $\mathscr{T}$ for the type constructor of the parameterised inductive type,
— ι_j $(j = 1, ..., m)$ for the constructors (or introduction operators),
— $\mathscr{E}_{\mathscr{T}}$ for the elimination operator,

together with the associated computation rules. We will now spell out in more detail how these constants and the computation rules should be declared.

Constants The constant $\mathscr{T}$ is declared as

$$\mathscr{T} : (\bar{Y} :: \bar{P})Type .$$

For the example *List*, we have

$$List : (A{:}Type)Type .$$

Constructors The constructors for $\mathscr{T}$ are declared as

$$\iota_j : (\bar{Y} :: \bar{P})\Theta_j[\bar{Y}, \mathscr{T}(\bar{Y})] \qquad (j = 1, ..., m).$$

Note that each constructor corresponds to an inductive schema. For the example *List*, the constructors are

$$\begin{aligned} nil &: (A{:}Type)List(A) \\ cons &: (A{:}Type)(a{:}A)(l{:}List(A))List(A). \end{aligned}$$

Elimination operators The elimination operator $\mathscr{E}_{\mathscr{T}}$ expresses an induction principle stating that, if a property is true for every value of the inductive type, it is true for every object of the type. It is declared as

$$\begin{aligned} \mathscr{E}_{\mathscr{T}} \ : \ & (\bar{Y} :: \bar{P}) \\ & (C : (\mathscr{T}(\bar{Y}))Type) \\ & (f_1 : \Theta_1^\circ[\mathscr{T}(\bar{Y}), C, \iota_1(\bar{Y})]) \ ... \ (f_m : \Theta_m^\circ[\mathscr{T}(\bar{Y}), C, \iota_m(\bar{Y})]) \\ & (z : \mathscr{T}(\bar{Y}))C(z) \end{aligned}$$

where C represents the predicate (property) over $\mathscr{T}(\bar{Y})$ (which is called the *motive* in McBride and McKinna (2004)), f_j $(j = 1, ..., m)$ are the methods, each corresponding to a constructor, and the kind of each method depends on the corresponding schema, defined as follows.

Definition 2.5. Let $\Theta \equiv (\bar{x} :: \bar{M})X$ be an inductive schema with respect to X parameterised by $\bar{Y}$, and $\langle \Phi_{i_1}, ..., \Phi_{i_k} \rangle$ be the subsequence of $\bar{M}$ that consists of all of the strictly positive operators in $\bar{M}$. Then, for $A : Type$, $C : (A)Type$ and $z : \Theta[\bar{Y}, A]$,

$$\begin{aligned} \Theta^\circ[A, C, z] \ =_{\mathrm{df}} \ & (\bar{x} :: \bar{M}[\bar{Y}, A]) \\ & (\Phi_{i_1}^\circ[A, C, x_{i_1}]) \ ... \ (\Phi_{i_k}^\circ[A, C, x_{i_k}]) \\ & C(z(\bar{x})). \end{aligned}$$

Note that in the special case when Θ is a strictly positive operator Φ (that is, when X does not occur free in $\bar{M}$), $\Phi^\circ[A, C, z] \equiv (\bar{x} :: \bar{M})C(z(\bar{x}))$.

For the example *List*, the elimination operator is

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

where its two methods (f_{nil} and f_{cons}) have kinds $C(nil(A))$ and $(a{:}A)(l{:}List(A))(C(l))$ $C(cons(A, a, l))$, respectively.

Computation rules When the elimination operator is applied to a value constructed by a constructor, the corresponding method gives the meaning of that application. This is

specified by the computation rules. Formally, for the parameterised inductive type $\mathscr{T}$, we have m computation rules, each corresponding to a constructor (or inductive schema). For $\Theta_j \equiv (\bar{x} :: \bar{M})X$ $(j = 1, ..., m)$ with $\langle \Phi_{i_1}, ..., \Phi_{i_k} \rangle$ being the subsequence of $\bar{M}$ that consists of all of the strictly positive operators, the jth computation rule is

$$\begin{aligned} \mathscr{E}_{\mathscr{T}}(\bar{Y}, C, \bar{f}, \iota_j(\bar{Y}, \bar{x})) \;=\; & f_j(\bar{x}, \Phi^{\sharp}_{i_1}[\mathscr{T}(\bar{Y}), C, \mathscr{E}_{\mathscr{T}}(\bar{Y}, C, \bar{f}), x_{i_1}], \\ & \ldots, \\ & \Phi^{\sharp}_{i_k}[\mathscr{T}(\bar{Y}), C, \mathscr{E}_{\mathscr{T}}(\bar{Y}, C, \bar{f}), x_{i_k}]) \\ : \;\; & C(\iota_j(\bar{Y}, \bar{x})) \end{aligned}$$

where $\bar{f} \equiv f_1, ..., f_m$ are the methods and the operation $\Phi^{\sharp}$ is defined as follows.

Definition 2.6. Let $\Phi \equiv (\bar{x} :: \bar{K})X$ be a strictly positive operator with respect to X. Then

$$\Phi^{\sharp}[A, C, f, z] =_{\mathrm{df}} [\bar{x}{:}\bar{K}]f(z(\bar{x}))$$

where $A : Type$, $C : (A)Type$, $f : (x{:}A)C(x)$ and $z : \Phi[A]$. Note that $\Phi^{\sharp}[A, C, f, z]$ is of kind $\Phi^{\circ}[A, C, z]$ and, when $m = 0$, $\Phi^{\sharp}[A, C, f, z] \equiv f(z)$.

For the example *List*, we have two computation rules:

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

We conclude this introduction to inductive types by giving several further examples.

Example 2.7. The following are examples of (parameterised) inductive types:

— The type of natural numbers N is generated by the schemata X and $(X)X$, with constructors $0 : N$ and $succ : (N)N$.
— The type of dependent functions $\Pi(A, B)$ parameterised by $A : Type$ and $B : (A)Type$ is generated by the schema $((x{:}A)B(x))X$, with constructor λ.
— The type of dependent pairs $\Sigma(A, B)$ parameterised by $A : Type$ and $B : (A)Type$ is generated by the schema $(x{:}A)(B(x))X$, with constructor *pair*.
— The disjoint union $Union(A, B)$ (or $A + B$ in the usual notation) parameterised by A, $B : Type$ is generated by the schemata $(A)X$ and $(B)X$, with constructors *inl* and *inr*.

2.3. *Coercive subtyping*

Coercive subtyping, as introduced in Luo (1997) and Luo (1999), is a framework for subtyping and abbreviation in dependent type theories. The formal presentation of the framework will be given after an introduction to the basic idea and an overview of previous work.

Basic idea

In coercive subtyping, a type is a subtype of another type if there is a unique coercion between them represented by the judgement

$$A <_c B : Type$$

where c has kind $(A)B$. Coercions are special functional operations (for example, declared by a user). If c is a coercion from A to B, then a function f from B to C can be applied to any object a of A and the application $f(a)$ is definitionally (or computationally) equal to $f(c(a))$. Intuitively, one may take f as a context that requires an object of B; then the argument a in the context f stands for its image of the coercion, $c(a)$.

The above simple idea, which has been studied in the literature for simple type systems (see, for instance, Mitchell (1991; 1983)), becomes very powerful when formulated in the logical framework and used in the context of dependent type theories. The framework of coercive subtyping covers a variety of subtyping relations, including those represented by coercions between parameterised inductive types (Luo 1999) and dependent coercions (Luo and Soloviev 1999). For example, see Luo (1999), Bailey (1999), Callaghan and Luo (2001) and Luo and Callaghan (1998) for details of some of these developments and for applications of coercive subtyping. Important meta-theoretic results of coercive subtyping have been studied, including those on conservativity (Soloviev and Luo 2002), and on transitivity elimination for subkinding (Soloviev and Luo 2002) and subtyping (Luo 2005; Luo and Luo 2005; Luo and Luo 2001). Coercion mechanisms have been implemented in the proof development systems Coq (Coq 2004), Lego (Luo and Pollack 1992) and Plastic (Callaghan and Luo 2001) by Saïbi (Saïbi 1997), Bailey (Bailey 1999) and Callaghan (Callaghan and Luo 2001), respectively.

Systems with coercive subtyping

Formally, a system of coercive subtyping $T[\mathscr{R}]$, with a set $\mathscr{R}$ of basic subtyping rules, is an extension of a type theory T specified in the logical framework LF. It can be presented in two stages:

— First consider the system $T[\mathscr{R}]_0$, which is an extension of T with subtyping judgements of the form $\Gamma \vdash A <_c B : Type$.

— Then consider the system $T[\mathscr{R}]$, which is an extension of $T[\mathscr{R}]_0$, with subkinding judgements of the form $\Gamma \vdash K <_c K'$.

The rules for subkinding include, for example, the following coercive definition rule:

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

This paper is mainly about subtyping rules and their properties in $T[\mathscr{R}]_0$, so the treatment of the kind level is omitted (see, for example, Luo (1999) for more details.)

Notation. As we are not going to talk about subkinding judgements in this paper, we shall often write $\Gamma \vdash A <_c B$ for the subtyping judgement $\Gamma \vdash A <_c B : Type$.

$T[\mathscr{R}]_0$ is the extension of T with the *congruence rule*

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

and a set $\mathscr{R}$ of 'basic subtyping rules' whose conclusions are subtyping judgements of the form $\Gamma \vdash A <_c B$. Examples of such rules include the structural subtyping rules for parameterised inductive types to be considered in this paper.

The set $\mathscr{R}$ of basic subtyping rules is required to be *coherent* in the sense that coercions between any two types must be unique. Put another way, the subtyping rules are coherent if the following rule is admissible in $T[\mathscr{R}]_0$:

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

For example, if we consider subtyping rules for inductive types, we want to show that they are coherent in the above sense. Furthermore, given a set of subtyping rules, we would like to prove that transitivity is admissible, that is, the following rule is admissible:

$$\frac{\Gamma \vdash A <_c B \quad \Gamma \vdash B <_{c'} C}{\Gamma \vdash A <_{c' \circ c} C}.$$

Well-defined coercions

We are going to prove, for example, admissibility results of the subtyping rules for inductive types *when there are other coercions already existing in the system*. We assume that those coercions (possibly generated by some other rules) are coherent and have good admissibility properties, that is, they form a set of well-defined coercions (Luo and Luo 2001), as defined below.

Definition 2.8 (Well-defined coercions). Let $\mathscr{C}$ be a set of subtyping judgements of the form $\Gamma \vdash A <_c B$. If $\mathscr{C}$ satisfies the following conditions, we say that $\mathscr{C}$ is a *well-defined set of subtyping judgements*, or *Well-Defined Coercions (WDC)* for short.

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

4 *Transitivity*:
If $\Gamma \vdash A <_c B \in \mathscr{C}$ and $\Gamma \vdash B <_{c'} C \in \mathscr{C}$, then $\Gamma \vdash A <_{c' \circ c} C \in \mathscr{C}$.

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

$$(\mathscr{C}) \qquad \frac{\Gamma \vdash A <_c B \ \in \ \mathscr{C}}{\Gamma \vdash A <_c B}.$$

3. Structural subtyping rules and functorial equality rules

In this section, we give a formulation of the structural subtyping rules for parameterised inductive types generated by inductive schemata, and the corresponding χ-rules describing the functorial laws for computational equality.

Given a parameterised inductive type $\mathscr{T}$ parameterised by n parameters $\bar{Y} :: \bar{P}$ $(n \in \omega)$ and generated by m schemata $\bar{\Theta} \equiv \Theta_1, ..., \Theta_m$ $(m \in \omega)$ in $\bar{Y} :: \bar{P}$, we consider the following form of structural subtyping rules for $\mathscr{T}$:

$$(*) \qquad \frac{premises(\bar{c})}{\Gamma \vdash \mathscr{T}(\bar{A}) <_{\mathrm{map}_{\mathscr{T}}} \mathscr{T}(\bar{B})}$$

where $\bar{A} \equiv A_1, ..., A_n$, $\bar{B} \equiv B_1, ..., B_n$ and $\bar{c} \equiv c_1, ..., c_k$ are fresh and distinct schematic letters. $\bar{A} :: \bar{P}$ and $\bar{B} :: \bar{P}$ correspond to the parameters of $\mathscr{T}$, $\bar{c}$ are the coercions assumed in the premises, and $\mathrm{map}_{\mathscr{T}}$ is the coercion specified by the rule, mapping canonical objects of $\mathscr{T}(\bar{A})$ to the corresponding canonical objects of $\mathscr{T}(\bar{B})$.

For example, for the type of lists $List(Y)$ parameterised by the type parameter Y and generated by the schemata X and $(Y)(X)X$, the structural subtyping rule is

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

where the coercion map_{List} takes the canonical objects $nil(A)$ to $nil(B)$ and $cons(A, a, l)$ to $cons(B, c(a), \mathrm{map}_{List}(l))$, respectively.

As another example, the type of dependent pairs $\Sigma(A, B)$ parameterised by $A : Type$ and $B : (A)Type$, is generated by the schema $(x{:}A)(B(x))X$. One of its three structural subtyping rules is

$$\frac{\Gamma \vdash A <_{c_1} A' \quad \Gamma, x{:}A \vdash B(x) <_{c_2[x]} B'(c_1(x))}{\Gamma \vdash \Sigma(A, B) <_{\mathrm{map}_{\Sigma}} \Sigma(A', B')}$$

where we have omitted the judgements for well-typedness and map_{Σ} maps $pair(A, B, a, b)$ to $pair(A', B', c_1(a), c_2[a](b))$. Note that, in this example for Σ-types, there is a dependency between parameters (the parameter B is dependent on the objects of A), which leads to the occurrence of c_1 in the second premise of the rule. (Such cases are excluded by the WT-schemata, and, therefore, not covered in Luo and Luo (2005)). It is because of such dependencies between parameters that a more general formulation of the subtyping rules, which inserts coercions in correct positions, is called for.

The following subsections are arranged as follows:

— Section 3.1 defines the 'components' that form the premises of the structural subtyping rules, declaring that certain operations are coercions between types.
— Section 3.2 defines the mapping functor given the premises, the coercion from $\mathscr{T}(\bar{A})$ to $\mathscr{T}(\bar{B})$.
— Section 3.3 describes the structural subtyping rules.

Finally, after defining the mapping functor, the composition of the mapping functors can be defined, so:

— Section 3.4 defines the functorial equality rules (χ-rules) for the parameterised inductive types.

3.1. *Components in premises*

Given a parameterised inductive type $\mathscr{T}$, we intend to show how to construct the structural subtyping rules of the form (*) (see above). More concretely, we want to construct the premises that assume certain coercions between appropriate types and, given these coercions, the mapping functor $\mathrm{map}_{\mathscr{T}}$ from $\mathscr{T}(\bar{A})$ to $\mathscr{T}(\bar{B})$, the coercion in the conclusion.

To this end, we define the *components* of a parameterised inductive type, each of which is a quadruple

$$(\Gamma;\ A;\ B;\ c)$$

consisting of a context, two types and a schematic letter c. The quadruple is intended to denote the fact that one of the premises in each of the structural subtyping rules is

$$\Gamma \vdash A <_c B \quad \text{or} \quad \Gamma \vdash A = B : Type\,.$$

For example, the components for the Σ-types constitute the sequence

$$(\Gamma;\ A;\ A';\ c_1),\ (\Gamma, x{:}A;\ B(x);\ B'(c_1(x));\ c_2)\,,$$

which corresponds to the two premises of the subtyping rules for Σ-types as presented above.

The definition of the notion of a component is better given in two stages, first defining it for small kinds and then for inductive schemata and parameterised inductive types.

Definition 3.1 (Components of small kinds). Let $K[\bar{x}]$ be a small kind in $(\Gamma, \bar{x}{::}\bar{P})$, $\Gamma \vdash \bar{a} :: \bar{P}$ and $\Gamma \vdash \bar{b} :: \bar{P}$. We shall define a sequence of quadruples, the *components* of K (with respect to $\bar{x}$, $\bar{a}$ and $\bar{b}$):

$$C_\Gamma(K;\bar{x};\bar{a};\bar{b}) = \langle(\Gamma_1;A_1;B_1;c_1),\ldots,(\Gamma_n;A_n;B_n;c_n)\rangle$$

where each A_i and B_i is a type in context Γ_i and c_i is a schematic letter. We shall simultaneously define an object c_K, the '*schematic coercion*' corresponding to K, such that $\Gamma \vdash c_K : (K[\bar{a}])K[\bar{b}]$ if $\Gamma_i \vdash c_i : (A_i)B_i$ $(i = 1,...,n)$.

The definitions are given by simultaneous induction on the structure of K:

— If $\bar{x} \notin K$, then

$$\begin{aligned} C_\Gamma(K;\bar{x};\bar{a};\bar{b}) &=_{\mathrm{df}} \langle\rangle \\ c_K &=_{\mathrm{df}} id_K \end{aligned}$$

where $id_K \equiv [x{:}K]x$ is the identity on K.

— If $K \equiv El(A[\bar{x}])$ with $\bar{x} \in A$, then

$$\begin{aligned} C_\Gamma(K;\bar{x};\bar{a};\bar{b}) &=_{\mathrm{df}} \langle(\Gamma;A[\bar{a}];A[\bar{b}];c)\rangle \\ c_K &=_{\mathrm{df}} c \end{aligned}$$

where c is a fresh schematic letter (and, for different occurrences of A, the schematic letters are different[†].)

— If $K \equiv K[\bar{x}] \equiv (y{:}K_1[\bar{x}])K_2$ with $\bar{x} \in K$, then

$$C_\Gamma(K;\bar{x};\bar{a};\bar{b}) =_{\mathrm{df}} \begin{cases} C_\Gamma(K_2;\bar{x};\bar{a};\bar{b}) & \text{if } \bar{x} \notin K_1 \text{ and } y \notin K_2 \\ C_{\Gamma,x':K_1}(K_2;\bar{x}::y;\bar{a}::x';\bar{b}::x') & \text{if } \bar{x} \notin K_1 \text{ and } y \in K_2 \\ C_\Gamma(K_1;\bar{x};\bar{b};\bar{a})\,\hat{}\,C_\Gamma(K_2;\bar{x};\bar{a};\bar{b}) & \text{if } \bar{x} \in K_1 \text{ and } y \notin K_2 \\ C_\Gamma(K_1;\bar{x};\bar{b};\bar{a}) & \\ \quad \hat{}\,C_{\Gamma,x':K_1[\bar{b}]}(K_2;\bar{x}::y;\bar{a}::c_{K_1}(x');\bar{b}::x') & \text{if } \bar{x} \in K_1 \text{ and } y \in K_2 \end{cases}$$

and $c_K =_{\mathrm{df}} [f{:}K[\bar{a}]][x'{:}K_1[\bar{b}]]c_{K_2}(f(c_{K_1}(x')))$. (Recall that, as defined in the first case above, $c_{K_1} \equiv id_{K_1}$ if $\bar{x} \notin K_1$.)

Example 3.2. The following examples, where N is the constant type of natural numbers (see Example 2.7), illustrate the above definition:

— In the following three examples, $\bar{x} \equiv Y$, $\bar{a} \equiv A$ and $\bar{b} \equiv B$:

- $K \equiv El(Y)$. $C_\Gamma(K;\bar{x};\bar{a};\bar{b}) = \langle(\Gamma;\ A;\ B;\ c)\rangle$.
- $K \equiv (N)Y$. $C_\Gamma(K;\bar{x};\bar{a};\bar{b}) = C_\Gamma(Y;\bar{x};\bar{a};\bar{b}) = \langle(\Gamma;\ A;\ B;\ c)\rangle$.
- $K \equiv (y{:}N)Y(y)$. $C_\Gamma(K;\bar{x};\bar{a};\bar{b}) = C_{\Gamma,y:N}(Y(y);\bar{x};\bar{a};\bar{b}) = \langle(\Gamma, y{:}N;\ A(y);\ B(y);\ c)\rangle$.

— In the following two examples, $\bar{x} \equiv Y_1, Y_2$, $\bar{a} \equiv A_1, A_2$ and $\bar{b} \equiv B_1, B_2$:

- $K \equiv (Y_1)Y_2$.

$$\begin{aligned} C_\Gamma(K;\bar{x};\bar{a};\bar{b}) &= C_\Gamma(Y_1;\bar{x};\bar{b};\bar{a})\,\hat{}\,C_\Gamma(Y_2;\bar{x};\bar{a};\bar{b}) \\ &= \langle(\Gamma;\ B_1;\ A_1;\ c_1),\ (\Gamma;\ A_2;\ B_2;\ c_2)\rangle\,. \end{aligned}$$

- $K \equiv (y{:}Y_1)Y_2(y)$.

$$\begin{aligned} C_\Gamma(K;\bar{x};\bar{a};\bar{b}) &= C_\Gamma(Y_1;\bar{x};\bar{b};\bar{a})\,\hat{}\,C_{\Gamma,x':B_1}(Y_2(y);\bar{x}::y;\bar{a}::c_{Y_1}(x');\bar{b}::x') \\ &= \langle(\Gamma;\ B_1;\ A_1;\ c_1),\ (\Gamma, x'{:}B_1;\ A_2(c_1(x'));\ B_2(x');\ c_2)\rangle\,. \end{aligned}$$

[†] Strictly speaking, the notion c_K is not enough, but we will abuse the notation with the understanding that different schematic letters are associated with different occurrences of a type, and different c_K are associated with different occurrences of a kind.

Definition 3.3 (Components of schemata). Let Θ be an inductive schema in $\Gamma, \bar{x} :: \bar{P}$, $\Gamma \vdash \bar{a} :: \bar{P}$ and $\Gamma \vdash \bar{b} :: \bar{P}$. We define $C_\Gamma(\Theta; \bar{x}; \bar{a}; \bar{b})$, the *components of* Θ (with respect to $\bar{x}$, $\bar{a}$ and $\bar{b}$), by induction on the structure of Θ.

— $\Theta \equiv X$. $C_\Gamma(\Theta; \bar{x}; \bar{a}; \bar{b}) =_{\mathrm{df}} \langle\rangle$.

— $\Theta \equiv (y{:}K\,[\bar{x}])\Theta_0$.

$$C_\Gamma(\Theta; \bar{x}; \bar{a}; \bar{b}) =_{\mathrm{df}} \begin{cases} C_\Gamma(\Theta_0; \bar{x}; \bar{a}; \bar{b}) & \text{if } \bar{x} \notin K \text{ and } y \notin \Theta_0 \\ C_{\Gamma, y:K}(\Theta_0; \bar{x}; \bar{a}; \bar{b}) & \text{if } \bar{x} \notin K \text{ and } y \in \Theta_0 \\ C_\Gamma(K; \bar{x}; \bar{a}; \bar{b}) \hat{} C_\Gamma(\Theta_0; \bar{x}; \bar{a}; \bar{b}) & \text{if } \bar{x} \in K \text{ and } y \notin \Theta_0 \\ C_\Gamma(K; \bar{x}; \bar{a}; \bar{b}) & \\ \quad \hat{} C_{\Gamma, x':K\,[\bar{a}]}(\Theta_0; \bar{x}::y; \bar{a}::x'; \bar{b}::c_K(x')) & \text{if } \bar{x} \in K \text{ and } y \in \Theta_0\,. \end{cases}$$

— $\Theta \equiv (\Phi)\Theta_0$. $C_\Gamma(\Theta; \bar{x}; \bar{a}; \bar{b}) =_{\mathrm{df}} C_\Gamma(\Phi; \bar{x}; \bar{b}; \bar{a}) \hat{} C_\Gamma(\Theta_0; \bar{x}; \bar{a}; \bar{b})$.

Example 3.4. The following examples illustrate this definition:

— $\Theta \equiv (N)X$.

$$C_\Gamma(\Theta; \bar{x}; \bar{a}; \bar{b}) = C_\Gamma(X; \bar{x}; \bar{a}; \bar{b}) = \langle\rangle\,.$$

— $\Theta \equiv (y{:}N)(Vect(y))X$, where $Vect : (N)Type$ is the inductive family of vectors (lists of fixed lengths) of natural numbers.

$$C_\Gamma(\Theta; \bar{x}; \bar{a}; \bar{b}) = C_{\Gamma, y:N}((Vect(y))X; \bar{x}; \bar{a}; \bar{b}) = C_{\Gamma, y:N}(X; \bar{x}; \bar{a}; \bar{b}) = \langle\rangle\,.$$

— $\Theta \equiv (Y)(X)X$, where $Y : Type$ is a parameter. This is one of the schemata that generate the types of lists parameterised by Y (see Section 2.2).

$$C_\Gamma(\Theta; Y; A; B) = C_\Gamma(Y; Y; A; B) \hat{} C_\Gamma((X)X; Y; A; B) = \langle(\Gamma;\ A;\ B;\ c)\rangle\,.$$

— $\Theta \equiv (y{:}A)(B(y))X$, where $A : Type$ and $B : (A)Type$ are parameters. This is the schema that generates Σ-types.

$$\begin{aligned} C_\Gamma(\Theta; \langle A, B\rangle; \langle A_1, B_1\rangle; \langle A_2, B_2\rangle) & \\ &= C_\Gamma(A; \langle A, B\rangle; \langle A_1, B_1\rangle; \langle A_2, B_2\rangle) \\ &\qquad \hat{} C_{\Gamma, x':A_1}((B(y))X; \langle A, B, y\rangle; \langle A_1, B_1, x'\rangle; \langle A_2, B_2, c_A(x')\rangle) \\ &= \langle(\Gamma;\ A_1;\ A_2;\ c_1),\ (\Gamma, x'{:}A_1;\ B_1(x');\ B_2(c_1(x'));\ c_2)\rangle\,. \end{aligned}$$

— $\Theta \equiv ((Y)X)X$, where $Y : Type$ is a parameter.

$$C_\Gamma(\Theta; Y; A; B) = C_\Gamma((Y)X; Y; B; A) \hat{} C_\Gamma(X; Y; A; B) = \langle(\Gamma;\ B;\ A;\ c)\rangle\,.$$

The components of a parameterised inductive type are given by the sequence that is the concatenation of those of the generating schemata.

Definition 3.5 (Components of parameterised inductive types). Let $\mathscr{T}$ be a parameterised inductive type, parameterised by $\bar{Y} :: \bar{P}$ and generated by $\bar{\Theta} \equiv \Theta_1, ..., \Theta_m$, and $\Gamma \vdash \bar{A} :: \bar{P}$ and $\Gamma \vdash \bar{B} :: \bar{P}$. Then, the *components of* $\mathscr{T}$ (wrt $\bar{A}$ and $\bar{B}$) are given by the sequence defined by

$$C_\Gamma(\mathscr{T}; \bar{A}; \bar{B}) =_{\mathrm{df}} C_\Gamma(\Theta_1; \bar{Y}; \bar{A}; \bar{B}) \hat{} \cdots \hat{} C_\Gamma(\Theta_m; \bar{Y}; \bar{A}; \bar{B})\,.$$

Example 3.6. The following show the sequences of components of the parameterised inductive types $List(A)$ (Section 2.2) together with those in Example 2.7:

— N has no parameters. Therefore, $C_\Gamma(N;\langle\rangle;\langle\rangle) = \langle\rangle$.
— $C_\Gamma(List;A;B) = \langle(\Gamma;\ A;\ B;\ c)\rangle$.
— $C_\Gamma(\Pi;A_1,B_1;A_2,B_2) = \langle(\Gamma;\ A_2;\ A_1;\ c_1),\ \ (\Gamma,y{:}A_2;\ B_1(c_1(y));\ B_2(y);\ c_2)\rangle$.
— $C_\Gamma(\Sigma;A_1,B_1;A_2,B_2) = \langle(\Gamma;A_1;A_2;c_1),\ (\Gamma,y{:}A_1;\ B_1(y);\ B_2(c_1(y));\ c_2)\rangle$.
— $C_\Gamma(Union;A_1,A_2;B_1,B_2) = \langle(\Gamma;\ A_1;\ B_1;\ c_1),\ \ (\Gamma;\ A_2;\ B_2;\ c_2)\rangle$.

Remark. Parameterised inductive types may have redundant parameters that, for example, do not occur in the generating schemata. For instance, $List'$ might be parameterised by $Y : Type$ and (the redundant parameter) $foo : K$, and generated by the same schemata X and $(Y)(X)X$. Provided K is non-empty (say with object k), $List'(A,k)$ behaves in the same way as $List(A)$, and their components are the same.

3.2. *Mapping functor*

Having defined the components (of the premises of the subtyping rules), we are now ready to show how the mapping functor (the coercions in the conclusions of the subtyping rules) can be constructed from the components.

We shall give a formal definition first, and then some explanatory examples. In the following definitions we assume that $\mathscr{T}$ is a parameterised inductive type:

— with parameters $\bar{Y} : \bar{P}$;
— generated by $\bar{\Theta} \equiv \Theta_1, ..., \Theta_m$; and
— with constructors $\bar{\iota} \equiv \iota_1, ..., \iota_m$.

Definition 3.7 (Methods). Let $\mathscr{T}$ be a parameterised inductive type parameterised by $\bar{Y} :: \bar{P}$, $\Gamma \vdash \bar{A} :: \bar{P}$ and $\Gamma \vdash \bar{B} :: \bar{P}$. The *method* $M_\Gamma[\Theta]$ that corresponds to a generating schema of $\mathscr{T}$ is defined below with the help of an intermediate notion $m_\Gamma[\Phi]$ for strictly positive operators Φ.

— Let $\Phi[\bar{x}, X]$ be a strictly positive operator in context $\Gamma, \bar{x} :: \bar{K}$, and let $\Gamma \vdash \bar{a} :: \bar{K}$ and $\Gamma \vdash \bar{b} :: \bar{K}$. For any $T : Type$ and $f : \Phi[\bar{a}, T]$, we define $m_\Gamma[\Phi](f)$ of kind $\Phi[\bar{b}, T]$ as follows:

  – $\Phi \equiv X$.
    Then, $m_\Gamma[\Phi](f) =_{\mathrm{df}} f$.

  – $\Phi \equiv (x{:}K\,[\bar{x}])\Phi_0$.
    Then $m_\Gamma[\Phi](f) =_{\mathrm{df}} [x{:}K\,[\bar{b}]]m_{\Gamma,x:K\,[\bar{b}]}[\Phi_0](f(c_K(x)))$, where c_K is defined with respect to $C_\Gamma(K;\bar{x};\bar{b};\bar{a})$.

— Let $\Theta \equiv (\bar{x}::\bar{M})X$ be a generating inductive schema of $\mathscr{T}$, $\iota : (\bar{Y} :: \bar{P})\Theta[\bar{Y}, \mathscr{T}(\bar{Y})]$ be the associated constructor and $\langle \Phi_{i_1}, \ldots \Phi_{i_k} \rangle$ be the subsequence of $\bar{M}$ that consists of all the strictly positive operators of $\bar{M}$. For any $g : \Theta[\bar{B}, \mathscr{T}(\bar{B})]$, we define $M_\Gamma[\Theta](g)$ of kind $[\bar{A}/\bar{Y}](\Theta^\circ[\mathscr{T}(\bar{Y}), [z{:}\mathscr{T}(\bar{Y})]\mathscr{T}(\bar{B}), \iota(\bar{Y})]$ as follows:

$$\begin{aligned} M_\Gamma[\Theta](g) \;=_{\mathrm{df}}\; & [\bar{x} :: \bar{M}[\bar{A}, \mathscr{T}(\bar{A})]] \\ & [x'_{i_1} : \Phi_{i_1}[\bar{A}, \mathscr{T}(\bar{B})]] \ldots [x'_{i_k} : \Phi_{i_k}[\bar{A}, \mathscr{T}(\bar{B})]] \\ & g(k_1, \ldots, k_n) \end{aligned}$$

where n is the length of $\bar{M}$ and for $i = 1, \ldots, n$,

$$k_i \equiv \begin{cases} c_{M_i}(x_i) & \text{if } M_i \text{ is a small kind} \\ m_\Gamma[\Phi_i](x'_i) & \text{if } M_i \text{ is a strictly positive operator} \end{cases}$$

where the context Γ_i is defined by

$$\begin{aligned} \Gamma_1 &\equiv \Gamma \\ \Gamma_{i+1} &\equiv \begin{cases} \Gamma_i, x_i{:}M_i[\bar{A}, \mathscr{T}(\bar{A})] & \text{if } x_i \text{ occurs free in one of } M_{i+1}, \ldots, M_n \\ \Gamma_i & \text{otherwise.} \end{cases} \end{aligned}$$

If M_i is a small kind, c_{M_i} is defined with respect to the components

$$C_\Gamma(M_i; \bar{Y}, x_{j_1}, \ldots, x_{j_r}; \bar{A}, x_{j_1}, \ldots, x_{j_r}; \bar{B}, k_{j_1}, \ldots, k_{j_r})$$

where $\Gamma_i \equiv \Gamma, x_{j_1}{:}M_{j_1}[\bar{A}, \mathscr{T}(\bar{A})], \ldots, x_{j_r}{:}M_{j_r}[\bar{A}, \mathscr{T}(\bar{A})]$.

Definition 3.8 (Mapping functor). Suppose that $\Gamma \vdash \bar{A} :: \bar{P}$, $\Gamma \vdash \bar{B} :: \bar{P}$, and

$$C_\Gamma(\mathscr{T}; \bar{A}; \bar{B}) = \langle (\Gamma_1; T_1; T'_1; c_1),\ \ldots,\ (\Gamma_n; T_n; T'_n; c_n) \rangle,$$

where $\Gamma_i \equiv \Gamma, \Delta_i$ $(i = 1, \ldots, n)$. Let $\bar{d} = \langle d_1, \ldots, d_n \rangle$ be the sequence of $d_i \equiv [\Delta_i]c_i$ $(i = 1, \ldots, n)$. The mapping functor of kind $(\mathscr{T}(\bar{A}))\mathscr{T}(\bar{B})$ is defined by

$$\mathrm{map}_\mathscr{T}[\bar{A}, \bar{B}, \bar{d}] =_{\mathrm{df}} \mathscr{E}_\mathscr{T}(\bar{A},\ [z{:}\mathscr{T}(\bar{A})]\mathscr{T}(\bar{B}),\ M_\Gamma[\Theta_1](\iota_1(\bar{B})),\ \ldots,\ M_\Gamma[\Theta_m](\iota_m(\bar{B})))$$

where $M_\Gamma[\Theta_j](\iota_j(\bar{B}))$ $(j = 1, \ldots, m)$ are the methods defined in Definition 3.7.

Example 3.9. For the examples of types considered in Examples 2.7 and 3.6, the mapping functors are:

— For the constant type N, there is no parameter. The mapping functor is

$$\mathrm{map}_N[] \;=_{\mathrm{df}}\; \mathscr{E}_N([x{:}N]N, 0, [x, y{:}N]succ(y)),$$

which is extensionally equal to the identity function on N.

— For the types of lists,

$$\begin{aligned} \mathrm{map}_{List}[A, B, c] \;=_{\mathrm{df}}\; & \mathscr{E}_{List}(A,\ [l{:}List(A)]List(B), \\ & nil(B),\ [x{:}A][l{:}List(A)][l'{:}List(B)]cons(B, c(x), l')). \end{aligned}$$

— For the Π-types,

$$\begin{aligned}\mathrm{map}_{\Pi}[A_1, B_1, A_2, B_2, c_1, [x_2{:}A_2]c_2] \\ =_{\mathrm{df}} \ \mathscr{E}_{\Pi}(A_1,\ B_1,\ [f{:}\Pi(A_1, B_1)]\Pi(A_2, B_2), \\ [f{:}(x{:}A_1)B_1(x)]\lambda(A_2, B_2, [x_2{:}A_2]c_2(f(c_1(x_2))))).\end{aligned}$$

— For the Σ-types,

$$\begin{aligned}\mathrm{map}_{\Sigma}[A_1, B_1, A_2, B_2, c_1, [x_1{:}A_1]c_2] \\ =_{\mathrm{df}} \ \mathscr{E}_{\Sigma}(A_1,\ B_1,\ [f{:}\Sigma(A_1, B_1)]\Sigma(A_2, B_2), \\ [x_1{:}A_1][y{:}B_1(x_1)]pair(A_2, B_2, c_1(x_1), c_2(y))).\end{aligned}$$

— For the union types,

$$\begin{aligned}\mathrm{map}_{Union}[A_1, A_2, B_1, B_2, c_1, c_2] \\ =_{\mathrm{df}} \ \mathscr{E}_{Union}(A_1,\ B_1,\ [x{:}A_1 + A_2]B_1 + B_2, \\ [x{:}A_1]inl(B_1, B_2, c_1(x)),\ [y{:}A_2]inr(B_1, B_2, c_2(y))).\end{aligned}$$

3.3. *Structural subtyping rules*

We are now ready to specify the structural subtyping rules (∗). Let

$$C_{\Gamma}(\mathscr{T}; \bar{A}; \bar{B}) = \langle(\Gamma_1; T_1; T_1'; c_1), \ldots, (\Gamma_n; T_n; T_n'; c_n)\rangle,$$

where $\Gamma_i \equiv \Gamma, \Delta_i$ $(i = 1, ..., n)$, and let $\bar{d} = \langle d_1, ..., d_n\rangle$ be the sequence of $d_i \equiv [\Delta_i]c_i$. One of the subtyping rules for $\mathscr{T}$ can be written down immediately:

$$\frac{\Gamma \vdash \bar{A} :: \bar{P} \qquad \Gamma \vdash \bar{B} :: \bar{P} \qquad \Gamma_1 \vdash T_1 <_{c_1} T_1' \quad \cdots \quad \Gamma_n \vdash T_n <_{c_n} T_n'}{\Gamma \vdash \mathscr{T}(\bar{A}) <_{\mathrm{map}_{\mathscr{T}}[\bar{A},\bar{B},\bar{d}]} \mathscr{T}(\bar{B})}$$

The others are formed from this by performing the following on some, but *not* all, of the premises:

— Change the premise $\Gamma_i \vdash T_i <_{c_i} T_i'$ to $\Gamma_i \vdash T_i = T_i' : Type$; and
— Replace c_i with $id_{T_i} \equiv [x{:}T_i]x$ wherever it occurs in the rest of the rule.

When a parameterised inductive type has k components, there are $2^k - 1$ structural subtyping rules for $\mathscr{T}$.

Example 3.10. For Π-types (see Examples 2.7, 3.6 and 3.9), the structural subtyping rules are (omitting the well-typedness judgements in the premises):

$$\frac{\Gamma \vdash A_2 <_{c_1} A_1 \quad \Gamma, x_2{:}A_2 \vdash B_1(c_1(x_2)) <_{c_2} B_2(x_2)}{\Gamma \vdash \Pi(A_1, B_1) <_{\mathrm{map}^1_{\Pi}} \Pi(A_2, B_2)}$$

$$\frac{\Gamma \vdash A_2 = A_1 : Type \quad \Gamma, x_2{:}A_2 \vdash B_1(id_{A_2}(x_2)) <_{c_2} B_2(x_2)}{\Gamma \vdash \Pi(A_1, B_1) <_{\mathrm{map}^2_{\Pi}} \Pi(A_2, B_2)}$$

$$\frac{\Gamma \vdash A_2 <_{c_1} A_1 \quad \Gamma, x_2{:}A_2 \vdash B_1(c_1(x_2)) = B_2(x_2) : Type}{\Gamma \vdash \Pi(A_1, B_1) <_{\mathrm{map}^3_\Pi} \Pi(A_2, B_2)}.$$

The mapping operations in the above rules are:

$$\begin{aligned}
\mathrm{map}^1_\Pi &\equiv \mathrm{map}_\Pi[A_1, B_1, A_2, B_2, c_1, [x_2{:}A_2]c_2] \\
&\equiv \mathscr{E}_\Pi(A_1,\ B_1,\ [f{:}\Pi(A_1, B_1)]\Pi(A_2, B_2), \\
&\qquad [f{:}(x{:}A_1)B_1(x)]\ \lambda(A_2, B_2, [x_2{:}A_2]c_2(f(c_1(x_2))))) \\
\mathrm{map}^2_\Pi &\equiv \mathrm{map}_\Pi[A_1, B_1, A_2, B_2, id_{A_2}, [x_2{:}A_2]c_2] \\
&\equiv \mathscr{E}_\Pi(A_1,\ B_1,\ [f{:}\Pi(A_1, B_1)]\Pi(A_2, B_2), \\
&\qquad [f{:}(x{:}A_1)B_1(x)]\ \lambda(A_2, B_2, [x_2{:}A_2]c_2(f(id_{A_2}(x_2))))) \\
\mathrm{map}^3_\Pi &\equiv \mathrm{map}_\Pi[A_1, B_1, A_2, B_2, c_1, [x_2{:}A_2]id_{B_1(c_1(x_2))}] \\
&\equiv \mathscr{E}_\Pi(A_1,\ B_1,\ [f{:}\Pi(A_1, B_1)]\Pi(A_2, B_2), \\
&\qquad [f{:}(x{:}A_1)B_1(x)]\ \lambda(A_2, B_2, [x_2{:}A_2]id_{B_1(c_1(x_2))}(f(c_1(x_2)))))
\end{aligned}$$

where $\mathrm{map}^1_\Pi \equiv \mathrm{map}_\Pi[A_1, B_1, A_2, B_2, c_1, [x_2{:}A_2]c_2]$ is as given in Example 3.9 and map^2_Π and map^3_Π are obtained from map^1_Π by *replacing* (not substituting) c_1 with id_{A_2} and c_2 with $id_{B_1(c_1(x_2))}$, respectively.

Remark (Non-applicable rules). As mentioned above, there are $2^k - 1$ rules if $\mathscr{T}$ has k components. Some of these rules may be non-applicable. For example, if $\mathscr{T}$ is generated by a schemata including $((Y)Y)X$, where $Y : Type$ is the only parameter, then one of the rules is (omitting the well-typedness judgements)

$$\frac{\Gamma \vdash A <_c B \qquad \Gamma \vdash B <_d A}{\Gamma \vdash \mathscr{T}(A) <_{\mathrm{map}_\mathscr{T}} \mathscr{T}(B)}.$$

Since the premises are contradictory (and never satisfied), the above rule can never be applied.

As another example, consider the inductive type $BooList$ parameterised by $Y : Type$ and generated by the schemata X, $(Y)(X)X$ and $(Y)(X)X$ (two repeated schemata). This would give the following rules

$$\frac{A <_c B \qquad A <_d B}{BooList(A) < BooList(B)} \qquad \frac{A <_c B \qquad A = B}{BooList(A) < BooList(B)} \qquad \frac{A = B \qquad A <_d B}{BooList(A) < BooList(B)}$$

where the mapping coercions in the conclusions are omitted. The first of the above rules is applicable only when $c = d$ and the other two are non-applicable.

3.4. *χ-rules: functorial laws of equality*

The functorial laws express the fact that the mapping functor is distributive with respect to the composition of 'morphisms'. In the simplest case, a morphism is just a functional operation and the distributivity concerned deals with the composition of two functional operations (*c.f.*, the example for *List* in the introduction). In the general case, a morphism is a sequence of objects that correspond to the schematic letters of the components of the generating schemata.

Definition 3.11 (Morphisms). Let $\mathscr{T}$ be an inductive type parameterised by $\bar{Y} :: \bar{P}$, $\Gamma \vdash \bar{A} :: \bar{P}$ and $\Gamma \vdash \bar{B} :: \bar{P}$. Suppose that the sequence of components

$$\langle (\Gamma_1, T_1, T_1', c_1),\ ...,\ (\Gamma_n, T_n, T_n', c_n) \rangle\,,$$

where $\Gamma_i \equiv \Gamma, \bar{x}_i{:}\bar{K}_i$, is one of the following:

1 $C_\Gamma(K\,;\bar{x};\bar{A};\bar{B})$, where K is a small kind;
2 $C_\Gamma(\Theta;\bar{x};\bar{A};\bar{B})$, where Θ is an inductive schema; or
3 $C_\Gamma(\mathscr{T}\,;\bar{A};\bar{B})$.

Then a sequence $\bar{k} \equiv k_1, ..., k_n$ is called a

1 K-morphism (from $\bar{A}$ to $\bar{B}$ with respect to $\bar{x}$), notation $\Gamma \vdash \bar{k} : \bar{A} \to^{\bar{x}}_{K} \bar{B}$,
2 Θ-morphism (from $\bar{A}$ to $\bar{B}$ with respect to $\bar{x}$), notation $\Gamma \vdash \bar{k} : \bar{A} \to^{\bar{x}}_{\Theta} \bar{B}$, or
3 $\mathscr{T}$-morphism (from $\bar{A}$ to $\bar{B}$), notation $\Gamma \vdash \bar{k} : \bar{A} \to_{\mathscr{T}} \bar{B}$,

respectively, if the judgements obtained by replacing c_i by $k_i(\bar{x}_i)$ from the judgements $\Gamma, \bar{x}_i{:}\bar{K}_i \vdash c_i : (T_i)T_i'$ $(i = 1, ..., n)$ are all derivable.

The following example explains the above definition, in particular, the process of replacement.

Example 3.12. Consider the parameterised inductive type (of triples) Σ_3 parameterised by $A : Type$, $B : (A)Type$ and $C : (x{:}A)(B(x))Type$ and generated by the schema $(x : A)(y : B(x))(z : C(x, y))X$. Its components are

$$\begin{aligned} C_\Gamma(\Sigma_3\,;A_1,B_1,C_1\,;A_2,B_2,C_2) \;=\; \langle\ & (\Gamma;\ A_1;\ A_2;\ c),\\ & (\Gamma, x{:}A_1;\ B_1(x);\ B_2(c(x));\ d),\\ & (\Gamma, x{:}A_1, y{:}B_1(x);\ C_1(x,y);\ C_2(c(x), d(y));\ e)\ \rangle\,. \end{aligned}$$

In this case, $\langle f, g, h\rangle$ is a Σ_3-morphism from $\langle A_1, B_1, C_1\rangle$ to $\langle A_2, B_2, C_2\rangle$ if the following judgements are derivable:

$$\begin{aligned} \Gamma &\vdash f : (A_1)A_2,\\ \Gamma, x{:}A_1 &\vdash g(x) : (B_1(x))B_2(f(x))\\ \Gamma, x{:}A_1, y{:}B_1(x) &\vdash h(x,y) : (C_1(x,y))C_2(f(x), g(x,y))\,. \end{aligned}$$

For example, the last judgement above is obtained from

$$\Gamma, x{:}A_1, y{:}B_1(x) \vdash e : (C_1(x,y))C_2(c(x), d(y))$$

by replacing c with f, d with $g(x)$ and e with $h(x, y)$, respectively.

Notation. We shall adopt the following notation for sequences, where $\bar{a} = \langle a_1, ..., a_n\rangle$ and $\bar{b} = \langle b_1, ..., b_n\rangle$:

— $[x{:}K]\bar{a} = \langle [x{:}K]a_1, ..., [x{:}K]a_n\rangle$.
— $\bar{a}(\bar{b}) = \langle a_1(b_1), ..., a_n(b_n)\rangle$.
In the special case where $b_1 = ... = b_n = b$, we write $\bar{a}(b)$ for $\bar{a}(\bar{b})$.

Definition 3.13 (Composition of morphisms). Let $\mathscr{T}$ be an inductive type parameterised by $\bar{Y} :: \bar{P}$ and $\bar{A}, \bar{B}, \bar{C} :: \bar{P}$.

— *Composition of K-morphisms*:
Let K be a small kind, $\Gamma \vdash \bar{k} : \bar{A} \to^{\bar{x}}_{K} \bar{B}$ and $\Gamma \vdash \bar{l} : \bar{B} \to^{\bar{x}}_{K} \bar{C}$. We define the K-morphism $\bar{l} \bullet \bar{k}$ such that $\Gamma \vdash \bar{l} \bullet \bar{k} : \bar{A} \to^{\bar{x}}_{K} \bar{C}$ using induction on the structure of K:

– $K \equiv El(D[\bar{x}])$.

$$\bar{l} \bullet \bar{k} =_{\mathrm{df}} \begin{cases} \langle\rangle & \text{if } \bar{x} \notin D \text{ (and hence } \bar{k} = \bar{l} = \langle\rangle) \\ d' \circ d & \text{if } \bar{x} \in D \text{ (and hence } \bar{k} = \langle d \rangle \text{ and } \bar{l} = \langle d' \rangle) \end{cases}$$

where $\Gamma \vdash d : (D[\bar{A}])D[\bar{B}]$ and $\Gamma \vdash d' : (D[\bar{B}])D[\bar{C}]$.

– $K \equiv (y{:}K_1[\bar{x}])K_2$.

$$\bar{l} \bullet \bar{k} =_{\mathrm{df}} \begin{cases} \bar{l} \bullet \bar{k} & \text{if } \bar{x} \notin K_1 \text{ and } y \notin K_2 \quad (1) \\ [y{:}K_1]\bar{l}(y) \bullet \bar{k}(y) & \text{if } \bar{x} \notin K_1 \text{ and } y \in K_2 \quad (2) \\ (\bar{k}_1 \bullet \bar{l}_1)\hat{}\,(\bar{l}_2 \bullet \bar{k}_2) & \text{if } \bar{x} \in K_1 \text{ and } y \notin K_2 \quad (3) \\ (\bar{k}_1 \bullet \bar{l}_1)\hat{}\,[z{:}K_1[\bar{C}]]\bar{l}_2(z) \bullet \bar{k}_2([\bar{l}_1/\bar{c}]c_{K_1}(z)) & \text{if } \bar{x} \in K_1 \text{ and } y \in K_2 \quad (4) \end{cases}$$

where, in the four different cases above, we have:

(1) $\Gamma \vdash \bar{k} : \bar{A} \to^{\bar{x}}_{K_2} \bar{B}$ and $\Gamma \vdash \bar{l} : \bar{B} \to^{\bar{x}}_{K_2} \bar{C}$.

(2) $\Gamma, y{:}K_1 \vdash \bar{k}(y) : \bar{A} \to^{\bar{x}}_{K_2} \bar{B}$ and $\Gamma, y{:}K_1 \vdash \bar{l}(y) : \bar{B} \to^{\bar{x}}_{K_2} \bar{C}$.

(3) $\bar{k} = \bar{k}_1\hat{}\,\bar{k}_2$ and $\bar{l} = \bar{l}_1\hat{}\,\bar{l}_2$ such that

$$\begin{aligned} \Gamma &\vdash \bar{k}_1 : \bar{B} \to^{\bar{x}}_{K_1} \bar{A} \\ \Gamma &\vdash \bar{l}_1 : \bar{C} \to^{\bar{x}}_{K_1} \bar{B} \\ \Gamma &\vdash \bar{k}_2 : \bar{A} \to^{\bar{x}}_{K_2} \bar{B} \\ \Gamma &\vdash \bar{l}_2 : \bar{B} \to^{\bar{x}}_{K_2} \bar{C}\,. \end{aligned}$$

(4) $\bar{k} = \bar{k}_1\hat{}\,\bar{k}_2$ and $\bar{l} = \bar{l}_1\hat{}\,\bar{l}_2$ such that

$$\begin{aligned} \Gamma &\vdash \bar{k}_1 : \bar{B} \to^{\bar{x}}_{K_1} \bar{A} \\ \Gamma &\vdash \bar{l}_1 : \bar{C} \to^{\bar{x}}_{K_1} \bar{B} \\ \Gamma, y{:}K_1[\bar{B}] &\vdash \bar{k}_2(y) : \bar{A}{::}[\bar{k}_1/\bar{c}]c_{K_1}(y) \to^{\bar{x}::y}_{K_2} \bar{B}{::}y \\ \Gamma, z{:}K_1[\bar{C}] &\vdash \bar{l}_2(z) : \bar{B}{::}[\bar{l}_1/\bar{c}]c_{K_1}(z) \to^{\bar{x}::z}_{K_2} \bar{C}{::}z \end{aligned}$$

where $\bar{c}$ are the schematic letters introduced in the components of K_1.

— *Composition of Θ-morphisms*:
Let Θ be an inductive schema. Suppose $\Gamma \vdash \bar{k} : \bar{A} \to^{\bar{x}}_{\Theta} \bar{B}$ and $\Gamma \vdash \bar{l} : \bar{B} \to^{\bar{x}}_{\Theta} \bar{C}$. We define the Θ-morphism $\bar{l} \bullet \bar{k}$ such that $\Gamma \vdash \bar{l} \bullet \bar{k} : \bar{A} \to^{\bar{x}}_{\Theta} \bar{C}$ using induction on the structure of Θ:

– $\Theta \equiv X$.
Then, $\bar{k} = \bar{l} = \langle\rangle$ and we define $\bar{l} \bullet \bar{k} =_{\mathrm{df}} \langle\rangle$.

– $\Theta \equiv (y{:}K\,[\bar{x}])\Theta_0$.

$$\bar{l} \bullet \bar{k} = \begin{cases} \bar{l} \bullet \bar{k} & \text{if } \bar{x} \notin K \text{ and } y \notin \Theta_0 \quad (1) \\ [y{:}K]\bar{l}(y) \bullet \bar{k}(y) & \text{if } \bar{x} \notin K \text{ and } y \in \Theta_0 \quad (2) \\ (\bar{l}_1 \bullet \bar{k}_1)\hat{}\,(\bar{l}_2 \bullet \bar{k}_2) & \text{if } \bar{x} \in K \text{ and } y \notin \Theta_0 \quad (3) \\ (\bar{l}_1 \bullet \bar{k}_1)\hat{}\,[x{:}K\,[\bar{A}]]\bar{l}_2([\bar{k}_1/\bar{c}]c_K(x)) \bullet \bar{k}_2(x) & \text{if } \bar{x} \in K \text{ and } y \in \Theta_0 \quad (4) \end{cases}$$

where, in the four different cases above, we have:

(1) $\Gamma \vdash \bar{k} : \bar{A} \to^{\bar{x}}_{\Theta_0} \bar{B}$ and $\Gamma \vdash \bar{l} : \bar{B} \to^{\bar{x}}_{\Theta_0} \bar{C}$.

(2) $\Gamma, y{:}K \vdash \bar{k}(y) : \bar{A} \to^{\bar{x}}_{\Theta_0} \bar{B}$ and $\Gamma, y{:}K \vdash \bar{l}(y) : \bar{B} \to^{\bar{x}}_{\Theta_0} \bar{C}$.

(3) $\bar{k} = \bar{k}_1\hat{}\,\bar{k}_2$ and $\bar{l} = \bar{l}_1\hat{}\,\bar{l}_2$ such that

$$\Gamma \vdash \bar{k}_1 : \bar{A} \to^{\bar{x}}_{K} \bar{B}$$
$$\Gamma \vdash \bar{l}_1 : \bar{B} \to^{\bar{x}}_{K} \bar{C}$$
$$\Gamma \vdash \bar{k}_2 : \bar{A} \to^{\bar{x}}_{\Theta_0} \bar{B}$$
$$\Gamma \vdash \bar{l}_2 : \bar{B} \to^{\bar{x}}_{\Theta_0} \bar{C}$$

(4) $\bar{k} = \bar{k}_1\hat{}\,\bar{k}_2$ and $\bar{l} = \bar{l}_1\hat{}\,\bar{l}_2$ such that $\bar{l} = \bar{l}_1\hat{}\,\bar{l}_2$ such that

$$\Gamma \vdash \bar{k}_1 : \bar{A} \to^{\bar{x}}_{K} \bar{B}$$
$$\Gamma \vdash \bar{l}_1 : \bar{B} \to^{\bar{x}}_{K} \bar{C}$$
$$\Gamma, x{:}K\,[\bar{A}] \vdash \bar{k}_2(x) : \bar{A}{::}x \to^{\bar{x}::x}_{\Theta_0} \bar{B}{::}[\bar{k}_1/\bar{c}]c_K(x)$$
$$\Gamma, y{:}K\,[\bar{B}] \vdash \bar{l}_2(y) : \bar{B}{::}y \to^{\bar{x}::y}_{\Theta_0} \bar{C}{::}[\bar{l}_1/\bar{c}]c_K(y)$$

where $\bar{c}$ are the schematic letters introduced in the components of K.

– $\Theta \equiv (\Phi)\Theta_0$.
Then $\bar{k} = \bar{k_1}\hat{}\,\bar{k_2}$ and $\bar{l} = \bar{l_1}\hat{}\,\bar{l_2}$ such that $\bar{l} = \bar{l}_1\hat{}\,\bar{l}_2$ such that

$$\Gamma \vdash \bar{k_1} : \bar{B} \to^{\bar{x}}_{\Phi} \bar{A}$$
$$\Gamma \vdash \bar{l_1} : \bar{C} \to^{\bar{x}}_{\Phi} \bar{B}$$
$$\Gamma \vdash \bar{k_2} : \bar{A} \to^{\bar{x}}_{\Theta_0} \bar{B}$$
$$\Gamma \vdash \bar{l_2} : \bar{B} \to^{\bar{x}}_{\Theta_0} \bar{C}\,.$$

We define

$$l \bullet k =_{\mathrm{df}} (\bar{k_1} \bullet \bar{l_1})\hat{}\,(\bar{l_2} \bullet \bar{k_2})\,.$$

— *Composition of $\mathscr{T}$-morphisms*:
Suppose $\Gamma \vdash \bar{k} : \bar{A} \to_{\mathscr{T}} \bar{B}$ and $\Gamma \vdash \bar{l} : \bar{B} \to_{\mathscr{T}} \bar{C}$. We have that $\bar{k} = \bar{k_1}\hat{}\,\cdots\hat{}\,\bar{k_m}$ and $\bar{l} = \bar{l_1}\hat{}\,\cdots\hat{}\,\bar{l_m}$ such that $\Gamma \vdash \bar{k_i} : \bar{A} \to^{\bar{Y}}_{\Theta_i} \bar{B}$ and $\Gamma \vdash \bar{l_i} : \bar{B} \to^{\bar{Y}}_{\Theta_i} \bar{C}$. We define the $\mathscr{T}$-morphism $\bar{l} \bullet \bar{k}$ such that $\Gamma \vdash \bar{l} \bullet \bar{k} : \bar{A} \to_{\mathscr{T}} \bar{C}$ by

$$\bar{l} \bullet \bar{k} =_{\mathrm{df}} (\bar{l_1} \bullet \bar{k_1})\hat{}\,\cdots\hat{}\,(\bar{l_n} \bullet \bar{k_n})\,.$$

Example 3.14. The following are some examples of morphisms:

— For the parameterised inductive types *List* generated by schemata X and $(Y)(X)X$, the *List*-morphisms are just functional operations $f : (A)B$ and $g : (B)C$, and their composition is just $g \circ f$.

— Let $\bar{A} \equiv A_1, A_2$, $\bar{B} \equiv B_1, B_2$ and $\bar{C} = C_1, C_2$.

- For the Π-types generated by schema $((x{:}Y_1)Y_2(x))X$ the morphisms $\Gamma \vdash \bar{k} : \bar{A} \to_\Pi \bar{B}$ and $\Gamma \vdash \bar{l} : \bar{B} \to_\Pi \bar{C}$ are pairs $\langle k_1, k_2\rangle$ and $\langle l_1, l_2\rangle$, respectively. Their composition yields $\bar{l} \bullet \bar{k} = \langle k_1 \circ l_1,\ [z{:}C_1]l_2(z) \circ k_2(l_1(z))\rangle$.
- For the Σ-types generated by the schema $(x{:}Y_1)(Y_2(x))X$, the morphisms $\Gamma \vdash \bar{k} : \bar{A} \to_\Sigma \bar{B}$ and $\Gamma \vdash \bar{l} : \bar{B} \to_\Sigma \bar{C}$ are pairs $\langle k_1, k_2\rangle$ and $\langle l_1, l_2\rangle$, respectively. Their composition yields $\bar{l} \bullet \bar{k} = \langle l_1 \circ k_1,\ [x{:}A_1]l_2(k_1(x)) \circ k_2(x)\rangle$.

χ-rules

With the composition operation, we can now describe the following χ-rule corresponding to the composition of morphisms:

$$(\chi) \qquad \frac{\Gamma \vdash \bar{k} : \bar{A} \to_{\mathscr{T}} \bar{B} \quad \Gamma \vdash \bar{l} : \bar{B} \to_{\mathscr{T}} \bar{C} \quad \Gamma \vdash a : \mathscr{T}(\bar{A})}{\Gamma \vdash \mathrm{map}_{\mathscr{T}}[\bar{A}, \bar{C}, \bar{l} \bullet \bar{k}](a) = \mathrm{map}_{\mathscr{T}}[\bar{B}, \bar{C}, \bar{l}]\left(\mathrm{map}_{\mathscr{T}}[\bar{A}, \bar{B}, \bar{k}](a)\right) : \mathscr{T}(\bar{C})}\,.$$

The simplest example is to consider the parameterised inductive types *List*. The above χ-rule gives for $l : List(A)$

$$\mathrm{map}_{List}[A, C, c' \circ c](l) = \mathrm{map}_{List}[B, C, c']\left(\mathrm{map}_{List}[A, B, c](l)\right),$$

which yields the equality $\mathrm{map}_{List}[A, C, c' \circ c] = \mathrm{map}_{List}[B, C, c'] \circ \mathrm{map}_{List}[A, B, c]$ (see the introduction for an explanation). For the other cases, such as Π-types and Σ-types, one can easily write down the corresponding results from Example 3.14.

We can also define the identity morphisms

$$Id_{\mathscr{T}}[\bar{A}] : \mathscr{T}(\bar{A}) \to_{\mathscr{T}} \mathscr{T}(\bar{A}),$$

and, similarly, we can write down the χ-rule corresponding to the identity morphism declaring that $\mathrm{map}_{\mathscr{T}}[\bar{A}, \bar{A}, Id_{\mathscr{T}}[\bar{A}]](a) = a : \mathscr{T}(\bar{A})$ for $a : \mathscr{T}(\bar{A})$. There is a categorical structure here: for $\mathscr{T}$ parameterised by $\bar{Y} :: \bar{P}$, there is a category whose objects are the sequences $\bar{A} :: \bar{P}$ and whose arrows are the $\mathscr{T}$-morphisms, with the identity given by the identity morphisms and the composition given by Definition 3.13. In such a categorical structure, the mapping that maps the object $\bar{A}$ to $\mathscr{T}(\bar{A})$ and the arrow $\bar{k}$ to $\mathrm{map}_{\mathscr{T}}[\bar{A}, \bar{B}, \bar{k}]$ becomes a functor in the presence of the χ-rules (for identity and composition), which corresponds to the functorial laws of computational equality for such a mapping.

In this paper, only the rule (χ) corresponding to composition is included in our system (and used in proving that transitivity is admissible for the structural subtyping rules in Section 5), and not the rule corresponding to the identity morphisms. The reason is that we consider the proper subtyping relation $A <_c B$ rather than the relation $A \leqslant_c B$. If the latter were used to formulate our system (and hence identity operations would become coercions; see Luo and Soloviev (1999)), the χ-rule corresponding to the identity morphisms would be needed.

Remark. Some researchers may be reluctant to add extensional equality rules to an intensional type theory or might even consider it as problematic (for example, from philosophical considerations). We are not taking a view against this. Besides their use in

considering subtyping, another 'justification' for introducing the χ-rules is that they make the mapping functors into true functors in the sense of category theory.

4. Injectivity of inductive type constructors

In this section, we shall prove that the type constructors of parameterised inductive types are injective with respect to computational equality. More precisely, for any parameterised inductive type $\mathscr{T}$ with parameters $\bar{Y} :: \bar{P}$, the rule

$$\texttt{(injectivity)} \qquad \frac{\Gamma \vdash \mathscr{T}(\bar{A}) = \mathscr{T}(\bar{B}) : Type}{\Gamma \vdash \bar{A} = \bar{B} :: \bar{P}}$$

is admissible under certain restrictions on computation rules. This injectivity result will be used in the proofs of the properties of the subtyping rules in the next section.

4.1. *Injectivity: an informal discussion*

The above injectivity result looks easy to prove, but it is not. The injectivity rule is admissible in a type theory such as the intensional type theory where the Church–Rosser property holds (the Church–Rosser property implies the injectivity result). However, it is not known whether this is the case for a type theory with the extensional χ-rules: meta-theoretic properties, such as Church–Rosser, have not been proved. As the injectivity result plays an essential role in our proofs in the next section, a proof of injectivity is required[†].

Intuitively, injectivity holds, even with the χ-rules, because the only way to compute within $\mathscr{T}(\bar{A})$ is to compute within one of the terms of $\bar{A}$. However, the proof is complicated by the fact that $\beta\eta$-equalities hold in the logical framework LF. An attempt to prove it directly in LF has failed: it would use a result like Church–Rosser for $\beta\eta$-reduction, which does not hold in general for type theories specified in LF[‡].

We could try to prove injectivity for an individual type theory (say, UTT extended with the χ-rules) by attempting to prove Church–Rosser for its reduction relation. This property is, however, difficult to prove for a reduction relation that includes χ-reduction; and, besides, we prefer to establish general results for type theories specified in LF. We shall prove that injectivity holds for every type theory that can be specified in LF, provided it involves no computation rules between types.

The tool we use for this purpose is the lambda-free logical framework TF (Aczel 2001; Adams 2004) – see Section 2.1.2 for a brief introduction. This is a logical framework that involves no computation at the logical framework level. Every $\beta\eta$-equivalence class of

[†] Thanks are due to Conor McBride who pointed out to us that a direct proof of injectivity is possible. Although the proof as presented here is indirect (through TF) and more complicated, because of the λ-terms and the associated β-equality in LF, it was his idea to consider a proof of injectivity by introducing in an inductive proof a notion more general than derivability.

[‡] To see that this is the case, suppose that A and B are computationally equal but not $\beta\eta$-equal (for example, $A \equiv Vect(2+2)$ and $B \equiv Vect(4)$). Then, the term $[x{:}A]([x{:}B]x)(x)$ β-reduces to $[x{:}A]x$ and η-reduces to $[x{:}B]x$, which have no common $\beta\eta$-reduct.

objects in LF corresponds to a unique term in TF. In TF therefore, it is true that the only way to compute with $\mathscr{T}(\bar{A})$ is to compute within one of the arguments $\bar{A}$. Injectivity is therefore much easier to prove in TF. From the fact that LF is a conservative extension of TF, we can conclude that injectivity holds in LF too.

Later in this section we assume that in declaring the type theory under consideration no computation rules between types are declared. That is, for every declared computation rule

$$k = k' : T \quad \text{for} \quad k_i : K_i \quad (i = 1, \ldots, n).$$

T is of the form $El(A)$. (Recall that we have already stipulated that one cannot declare equalities between objects of dependent product kinds (Luo 1999).) This restriction excludes type universes as their formulation would introduce computational equalities between types. However, such type theories do include Martin-Löf's type theory without type universes or UTT without type universes. In particular, the computation rules for the parameterised inductive types in Section 2.2 and the χ-rules in Section 3.4 are all allowed.

4.2. *Conservativity of LF over TF*

LF is a conservative extension over TF. In the following, we describe the two translations

$$\begin{aligned} \mathrm{NF} &: \mathrm{LF} \to \mathrm{TF} \\ \mathrm{lift} &: \mathrm{TF} \to \mathrm{LF}, \end{aligned}$$

and prove their properties, from which the conservativity result follows.

4.2.1. *Translation* NF *from LF to TF* This translation consists of reducing each object of LF to its β-normal and η-long form (erasing kind labels as we go). This process will be guided by the arities of TF (see Section 2.1.2), which tell us, in particular, how far to η-expand each variable and constant.

We begin by assigning an arity $\mathrm{Ar}(K)$ to each kind K of LF as follows:

$$\begin{aligned} \mathrm{Ar}(Type) \equiv \mathrm{Ar}(\mathrm{El}(A)) &\equiv \mathbf{0} \\ \mathrm{Ar}\left((x{:}K)K'\right) &\equiv (\alpha, \beta_1, ..., \beta_n) \end{aligned}$$

where $\mathrm{Ar}(K) = \alpha$ and $\mathrm{Ar}\left(K'\right) = (\beta_1, ..., \beta_n)$. Furthermore, we assume that TF and LF use the same sets of variables and constants; in particular, every variable and constant has an associated arity, and that

— whenever a variable declaration $x{:}K$ appears (in a context, λ-object or Π-kind), the variable x has arity $\mathrm{Ar}(K)$;

— whenever a constant declaration $c{:}K$ is made, the constant c has arity $\mathrm{Ar}(K)$.

Note that the equality rules for λ-objects and Π-kinds do not cause any problems for this convention, as it is easy to see that, whenever $\Gamma \vdash K = K'$ is derivable, $\mathrm{Ar}(K) \equiv \mathrm{Ar}\left(K'\right)$.

Definition 4.1 (Translation NF**).** NF is defined as follows:

— For every object M of LF, the corresponding object $\mathrm{NF}(M)$ of TF is defined as

$$\mathrm{NF}(M) =_{\mathrm{df}} \begin{cases} z_\eta & \text{if } M \equiv z, \text{ with } z \text{ a variable or constant} \\ [x]\mathrm{NF}(k) & \text{if } M \equiv [x{:}K]k \\ \mathrm{NF}(k_1)\{\mathrm{NF}(k_2)\} & \text{if } M \equiv k_1(k_2)\,. \end{cases}$$

— For every kind K of LF, the corresponding kind $\mathrm{NF}(K)$ of TF is defined as

$$\mathrm{NF}(K) =_{\mathrm{df}} \begin{cases} Type & \text{if } K \equiv Type \\ El(\mathrm{NF}(A)) & \text{if } K \equiv El(A) \\ (x : \mathrm{NF}(K_1))\mathrm{NF}(K_2) & \text{if } K \equiv (x{:}K_1)K_2\,. \end{cases}$$

The definition of NF is extended to contexts and judgements as usual.

Suppose we have specified a type theory T in LF by declaring constants and computation rules. Then there is a set of declarations we can make in TF that yield 'the same' type theory as T:

— For every constant declaration $c : K$ made in LF, declare $c : \mathrm{NF}(K)$ in TF.
— For every computation rule declared in LF by

$$k_1 = k_2 : T \quad \text{for} \quad \bar{x} : \bar{J}\,,$$

declare in TF the computation rule

$$\mathrm{NF}(k_1) = \mathrm{NF}(k_2) : \mathrm{NF}(T) \quad \text{for} \quad \bar{x} : \mathrm{NF}(\bar{J})\,.$$

We shall refer to this TF type theory as the *normalised form* of the LF type theory T.

The translation NF is not defined on every expression of the raw syntax; for example, $\mathrm{NF}(k_1(k_2))$ is only defined if $\mathrm{NF}(k_1)$ and $\mathrm{NF}(k_2)$ have appropriate arities. However, it is defined on every typable object, and is a sound translation from LF to TF, as shown by the theorem below, which is proved using the following lemma on how substitution behaves under the translation.

Lemma 4.2. If $\mathrm{NF}(k)$ and $\mathrm{NF}(k')$ are defined, and $\mathrm{NF}(k)$ has the same arity as x, then $\mathrm{NF}([k/x]k')$ is defined and $\mathrm{NF}([k/x]k') \equiv \{\mathrm{NF}(k)/x\}\mathrm{NF}(k')$.

Proof. The proof is by structural induction, using Proposition 2.3(1) when $k \equiv x$ or $k \equiv k_1k_2$. □

Theorem 4.3 (Soundness of NF**).** Suppose we have declared a type theory T in LF, and its normalised form in TF is T_0. If the judgement J is derivable in T, then $\mathrm{NF}(J)$ is well-defined and derivable in T_0.

Proof. The proof is by induction on the derivation of J and using the properties in Proposition 2.3. As an illustration, we will only consider the case where the last rule of the derivation is

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

The induction hypothesis gives the following judgements in T_0:

$$\begin{aligned} &\mathrm{NF}(\Gamma) \vdash \mathrm{NF}(f) : (x{:}\mathrm{NF}(K))\mathrm{NF}(K') && (\#)\\ &\mathrm{NF}(\Gamma) \vdash \mathrm{NF}(k) : \mathrm{NF}(K)\,. \end{aligned}$$

Note that $\mathrm{NF}(f)$ must therefore have the same arity as $(x{:}\mathrm{NF}(K))\mathrm{NF}(K')$, and so have the form $[x]N_0$. Therefore, $\mathrm{NF}(f(k))$ is well defined and is the object $\{\mathrm{NF}(k)/x\}N_0$. The judgement $(\#)$ above is

$$\mathrm{NF}(\Gamma), x{:}\mathrm{NF}(K) \vdash N_0 : \mathrm{NF}(K')\,,$$

so Proposition 2.3(4) gives

$$\mathrm{NF}(\Gamma) \vdash \{\mathrm{NF}(k)/x\}N_0 : \{\mathrm{NF}(k)/x\}\mathrm{NF}(K')\,,$$

which, by Lemma 4.2, is what was required. □

4.2.2. *Translation* lift *from TF to LF* This translation lifts entities in TF to LF and consists of little more than inferring the kind labels in the λ-abstractions in LF. To do this for an object M, we need to know the context in which we are working (to provide the expected kinds). Furthermore, we need to know the kinds of the possible arguments of an object, and this information will be given by a context of the same arity as that of the object.

Definition 4.4 (Translation lift**).** The translation lift from TF to LF is defined by simultaneous induction:

— For any contexts Γ and Δ and any object M, where Δ and M have the same arity (with the intention that in TF, $\Gamma \vdash M : (\Delta)T$ for some T), the corresponding LF object $\mathrm{lift}^{\Delta}_{\Gamma}(M)$ is defined as:

– If $M \equiv zP_1 ... P_n$, where $z : (\bar{y}{:}\bar{\Gamma})T$, with $\Gamma_i \equiv (\Delta_i)T_i$ $(i = 1, ..., n)$, is either a variable in Γ or a declared constant, then

$$\mathrm{lift}^{\langle\rangle}_{\Gamma}(M) =_{\mathrm{df}} z(\mathrm{lift}^{\Delta_1}_{\Gamma}(P_1), ..., \mathrm{lift}^{\Delta_n}_{\Gamma}(P_n))\,.$$

– If $M \equiv [x]M_0$, then $\mathrm{lift}^{x:K,\Delta}_{\Gamma}(M) =_{\mathrm{df}} [x{:}\mathrm{lift}_{\Gamma}(K)]\mathrm{lift}^{\Delta}_{\Gamma,x:K}(M_0)$.

— For any kind K and context Γ in TF, the corresponding LF kind $\mathrm{lift}_{\Gamma}(K)$ is defined by

$$\mathrm{lift}_{\Gamma}(M) =_{\mathrm{df}} \begin{cases} Type & \text{if } M \equiv Type \\ El(\mathrm{lift}^{\langle\rangle}_{\Gamma}(A)) & \text{if } M \equiv El(A) \\ (x{:}\mathrm{lift}_{\Gamma}(K))\mathrm{lift}_{\Gamma,x:K}(K') & \text{if } M \equiv (x{:}K)K'\,. \end{cases}$$

The lifting operation is extended to contexts by

$$\begin{aligned} \mathrm{lift}(\langle\rangle) &\equiv \langle\rangle \\ \mathrm{lift}(\Gamma, x{:}K) &\equiv \mathrm{lift}(\Gamma), x{:}\mathrm{lift}_{\Gamma}(K) \end{aligned}$$

and to judgements by

$$\begin{aligned} \mathrm{lift}(\Gamma \text{ valid}) &\equiv \mathrm{lift}(\Gamma) \text{ valid} \\ \mathrm{lift}(\Gamma \vdash M : K) &\equiv \mathrm{lift}(\Gamma) \vdash \mathrm{lift}^{\langle\rangle}_{\Gamma}(M) : \mathrm{lift}_{\Gamma}(K) \end{aligned}$$

$$\mathrm{lift}(\Gamma \vdash M = N : K) \equiv \mathrm{lift}(\Gamma) \vdash \mathrm{lift}_\Gamma^{\langle\rangle}(M) = \mathrm{lift}_\Gamma^{\langle\rangle}(N) : \mathrm{lift}_\Gamma(K).$$

The soundness of the lifting operation is shown by Theorem 4.6 below, which is proved using the following lemma on how substitution behaves under the translation.

Lemma 4.5. Let M be an object, K be a kind, x be a variable and Δ be a context, all of the same arity α. Then:

1 $[\mathrm{lift}_\Gamma^\Delta(M)/x]\mathrm{lift}_{\Gamma,x:(\Delta)T}^\Theta(N) \to_\beta^* \mathrm{lift}_\Gamma^{\{M/x\}\Theta}(\{M/x\}N)$.
2 $[\mathrm{lift}_\Gamma^\Delta(M)/x]\mathrm{lift}_{\Gamma,x:(\Delta)T}(K) \to_\beta^* \mathrm{lift}_\Gamma(\{M/x\}K)$.

Proof. The proof is by simultaneous induction on α and the structures of N and K. □

Theorem 4.6 (Soundness of lift**).** Suppose we have declared a type theory T in LF, and its normalised form in TF is T_0. If the judgement J is derivable in T_0, then $\mathrm{lift}(J)$ is derivable in T.

Proof. The proof is by induction on derivations, making repeated use of Lemma 4.5 and Theorem 2.1. □

4.2.3. *Conservativity of LF over TF* The following theorem establishes the conservativity result. More specifically, it shows that, up to judgemental equality, the translation from TF to LF (lift) is a left inverse of that from LF to TF (NF).

Theorem 4.7 (conservativity of LF over TF). Assume that type theory T has been declared in LF.

1 If $\Gamma \vdash k : K$ in T, then $\mathrm{NF}(K)$ is defined and has the form $\mathrm{NF}(K) \equiv (\Delta)T$, and, furthermore, $\mathrm{lift}_{\mathrm{NF}(\Gamma)}^\Delta(\mathrm{NF}(k))$ is defined and $\Gamma \vdash k = \mathrm{lift}_{\mathrm{NF}(\Gamma)}^\Delta(\mathrm{NF}(k)) : K$ in T.
2 If $\Gamma \vdash K\ kind$ in T, then $\mathrm{lift}_{\mathrm{NF}(\Gamma)}(\mathrm{NF}(K))$ is defined and $\Gamma \vdash K = \mathrm{lift}_{\mathrm{NF}(\Gamma)}(\mathrm{NF}(K))$ in T.

Proof. Both parts are proved by induction on the derivation of the premise. We will only deal here with the case where the following rule is the last one in the derivation:

$$\frac{\Gamma \vdash f : (x{:}K)K' \qquad \Gamma \vdash k : K}{\Gamma \vdash f(k) : [k/x]K'}.$$

Let $\mathrm{NF}(K) \equiv (\Delta)T$ and $\mathrm{NF}(K') \equiv (\Delta')T'$. Then, by Lemma 4.2, $\mathrm{NF}([k/x]K') \equiv (\{\mathrm{NF}(k)/x\}\Delta')\{\mathrm{NF}(k)/x\}T'$. By the induction hypothesis, we have

$$\begin{aligned}\Gamma &\vdash f = \mathrm{lift}_{\mathrm{NF}(\Gamma)}^{x:\mathrm{NF}(K),\Delta'}(\mathrm{NF}(f)) : (x{:}K)K' \\ \Gamma &\vdash k = \mathrm{lift}_{\mathrm{NF}(\Gamma)}^{\Delta}(\mathrm{NF}(k)) : K,\end{aligned}$$

so

$$\Gamma \vdash f(k) = \mathrm{lift}_{\mathrm{NF}(\Gamma)}^{x:\mathrm{NF}(K),\Delta'}(\mathrm{NF}(f))(\mathrm{lift}_{\mathrm{NF}(\Gamma)}^{\Delta}(\mathrm{NF}(k))) : [k/x]K'.$$

The result will then follow by Subject Reduction (Theorem 2.1) if we can show

$$\mathrm{lift}_{\mathrm{NF}(\Gamma)}^{x:\mathrm{NF}(K),\Delta'}(\mathrm{NF}(f))\,\mathrm{lift}_{\mathrm{NF}(\Gamma)}^{\Delta}(\mathrm{NF}(k)) \to_\beta^* \mathrm{lift}_{\mathrm{NF}(\Gamma)}^{\{\mathrm{NF}(k)/x\}\Delta'}(\mathrm{NF}(f)\{\mathrm{NF}(k)\})\,z.$$

To prove this, we shall prove more generally that

$$\mathrm{lift}_\Gamma^{x:(\Delta)K,\Delta'}(M)\,\mathrm{lift}_\Gamma^{\Delta}(N) \to_\beta^* \mathrm{lift}_\Gamma^{\{N/x\}\Delta'}\left(M\{N\}\right).$$

Let $M \equiv [x]M_0$. Then

$$\begin{aligned}
\mathrm{lift}_\Gamma^{x:(\Delta)K,\Delta'}([x]M_0)\,\mathrm{lift}_\Gamma^{\Delta}(N) &\equiv ([x{:}\mathrm{lift}_\Gamma((\Delta)K)]\mathrm{lift}_{\Gamma,x:(\Delta)K}^{\Delta'}(M_0))\mathrm{lift}_\Gamma^{\Delta}(N)\\
&\to_\beta [\mathrm{lift}_\Gamma^{\Delta}(N)/x]\mathrm{lift}_{\Gamma,x:(\Delta)K}^{\Delta'}(M_0)\\
&\to_\beta^* \mathrm{lift}_\Gamma^{\{N/x\}\Delta'}\left(\{N/x\}M_0\right) \qquad \text{(Lemma 4.5)}\\
&\equiv \mathrm{lift}_\Gamma^{\{N/x\}\Delta'}\left(M\{N\}\right).
\end{aligned}$$

This completes the proof. □

Remark. It is also true that NF is a left inverse of lift up to syntactic identity (that is, $\mathrm{NF}(\mathrm{lift}_\Gamma^{\Delta}(M)) \equiv M$). But we shall not need this fact here.

4.3. *Proof of injectivity*

We shall prove that the injectivity property holds in TF, then use the conservativity result above to deduce that the corresponding property holds in LF also.

Theorem 4.8 (Injectivity in TF). Assume we have declared a type theory in TF that involves no computation rules between types and includes the constant declaration

$$\mathscr{T} : (\bar{Y} :: \bar{P})Type.$$

If $\Gamma \vdash \mathscr{T}(\bar{A}) = N : Type$ or $\Gamma \vdash N = \mathscr{T}(\bar{A}) : Type$, then $N \equiv \mathscr{T}(\bar{B})$ for some terms $\bar{B}$ such that $\Gamma \vdash \bar{A} = \bar{B} :: \bar{P}$.

Proof. The proof is by induction on derivations of the premise in TF. We have the following cases:

— Rule (CA) of 'constant application' (in Figure 2):

$$\frac{\Gamma \vdash \bar{A} = \bar{B} :: \bar{P}}{\Gamma \vdash \mathscr{T}(\bar{A}) = \mathscr{T}(\bar{B}) : Type}.$$

The result is immediate from the induction hypothesis.

— The reflexivity rule (R):

$$\frac{\Gamma \vdash \mathscr{T}(\bar{A}) : Type}{\Gamma \vdash \mathscr{T}(\bar{A}) = \mathscr{T}(\bar{A}) : Type}.$$

Inverting the premise, we have $\Gamma \vdash \bar{A} :: \bar{P}$ for some $\bar{P}$, and hence $\Gamma \vdash \bar{A} = \bar{A} :: \bar{P}$ by (R).

— The symmetry rule (S).

The result is immediate.

— The transitivity rule (T):

$$\frac{\Gamma \vdash \mathscr{T}(\bar{A}) = M : Type \qquad \Gamma \vdash M = N : Type}{\Gamma \vdash \mathscr{T}(\bar{A}) = N : Type}\,.$$

By the induction hypothesis, $M \equiv \mathscr{T}(\bar{B})$ and $N \equiv \mathscr{T}(\bar{C})$ such that $\Gamma \vdash \bar{A} = \bar{B} :: \bar{P}$ and $\Gamma \vdash \bar{B} = \bar{C} :: \bar{P}$. Therefore, $\Gamma \vdash \bar{A} = \bar{C} :: \bar{P}$ by (T). □

Remark. The restriction that there is no computation rule declared between types is important for the above proof to go through. If we did not impose this restriction, we would have to consider a case where there are declared equalities between types, and the proof would not go through as stated. For instance, if one declared an equation

$$List(N) = List(Unit) : Type$$

where *Unit* is the unit type with one object, it is obvious that we could not prove that $N = Unit : Type$. Having said this, however, we believe that the injectivity theorem is still true in more restricted cases, such as the introduction of universes.

Finally, the following corollary shows that the inductive type constructors are injective.

Corollary 4.9 (Injectivity in LF). Assume we have declared a type theory T in LF that includes no computation rules between types and includes the declarations for a parameterised inductive type $\mathscr{T}$ parameterised by $\bar{Y} : \bar{P}$, as in Section 2.2. If $\Gamma \vdash \mathscr{T}(\bar{A}) = \mathscr{T}(\bar{B}) : Type$, then $\Gamma \vdash \bar{A} = \bar{B} :: \bar{P}$.

Proof. We declare in TF the normalised form of T; in particular, we make the declaration

$$\mathscr{T} : (\bar{Y} :: \mathrm{NF}(\bar{P}))Type\,.$$

We use $\vdash^{\mathrm{TF}}$ for the judgements in TF thus extended, and $\vdash^{\mathrm{LF}}$ for the judgements in LF with T declared. Then we have

$$\begin{aligned}
\Gamma \vdash^{LF} \mathscr{T}(\bar{A}) = \mathscr{T}(\bar{B}) : Type &\Longrightarrow \mathrm{NF}(\Gamma) \vdash^{\mathrm{TF}} \mathscr{T}(\mathrm{NF}(\bar{A})) = \mathscr{T}(\mathrm{NF}(\bar{B})) : Type \\
&\qquad\qquad \text{(Theorem 4.3)} \\
&\Longrightarrow \mathrm{NF}(\Gamma) \vdash^{\mathrm{TF}} \mathrm{NF}(\bar{A}) = \mathrm{NF}(\bar{B}) :: \mathrm{NF}(\bar{P}) \\
&\qquad\qquad \text{(Theorem 4.8)} \\
&\Longrightarrow \mathrm{lift}(\mathrm{NF}(\Gamma)) \vdash^{LF} \mathrm{lift}^{\mathrm{NF}(\bar{P})}_{\mathrm{NF}(\Gamma)}\left(\mathrm{NF}(\bar{A})\right) = \\
&\qquad\qquad \mathrm{lift}^{\mathrm{NF}(\bar{P})}_{\mathrm{NF}(\Gamma)}\left(\mathrm{NF}(\bar{B})\right) :: \mathrm{lift}_{\mathrm{NF}(\Gamma)}(\mathrm{NF}(\bar{P})) \\
&\qquad\qquad \text{(Theorem 4.6)}\,.
\end{aligned}$$

By Theorem 4.7, we have

$$\begin{aligned}
\vdash^{LF} \ & \Gamma = \mathrm{lift}(\mathrm{NF}(\Gamma)) \\
\Gamma \vdash^{LF} \ & \bar{P} = \mathrm{lift}_{\mathrm{NF}(\Gamma)}(\mathrm{NF}(\bar{P})) \\
\Gamma \vdash^{LF} \ & \bar{A} = \mathrm{lift}^{\mathrm{NF}(\bar{P})}_{\mathrm{NF}(\Gamma)}\left(\mathrm{NF}(\bar{A})\right) :: \bar{P} \\
\Gamma \vdash^{LF} \ & \bar{B} = \mathrm{lift}^{\mathrm{NF}(\bar{P})}_{\mathrm{NF}(\Gamma)}\left(\mathrm{NF}(\bar{B})\right) :: \bar{P}\,.
\end{aligned}$$

Therefore, we have $\Gamma \vdash^{LF} \bar{A} = \bar{B} :: \bar{P}$. □

Remark. It may appear that this use of TF is just a technical trick for proving the injectivity of inductive type constructors. However, we believe that lambda-free logical frameworks such as TF are more than that: they are a powerful tool for proving results about logical frameworks. Proving the conservativity of LF over TF is a one-time-only cost. We expect many results to be easier to prove in TF than in LF; the results can then be lifted to LF in the manner of the proof of Corollary 4.9.

5. Meta-theoretic properties of structural subtyping

In this section we show that for a type theory T (such as Martin-Löf's type theory or UTT) without universes, the structural subtyping rules for parameterised inductive types, as defined in Section 3.3, are coherent and satisfy the property that transitivity is admissible in the presence of the functorial χ-rules for computational equality.

We shall make the following assumptions and use the corresponding notational conventions:

— $\mathscr{C}$ denotes a WDC (set of well-defined judgements of coercions), representing 'the existing derivable subtyping judgements'.
— PIT denotes an arbitrary set of parameterised inductive types such that:
  – If $\Gamma \vdash A <_c B \in \mathscr{C}$, then neither A nor B is computationally equal to a $\mathscr{T}$-type for $\mathscr{T} \in$ PIT.
— $\mathscr{R}_{\mathrm{PIT}}$ denotes the set of subtyping rules consisting of the $(\mathscr{C})$-rule (see the end of Section 2.3) and the structural subtyping rules for the parameterised inductive types in PIT (as given in Section 3.3).
— $\mathscr{C}_{\mathrm{PIT}}$ denotes the set of subtyping judgements derivable in $T_\chi[\mathscr{R}_{\mathrm{PIT}}]_0$, where T_χ is the type theory T extended with the χ-rules, while T is a type theory specified in LF without equality declarations between types (see Section 4.1 for discussions on this last restriction).

What we shall show in the following subsection is that $\mathscr{C}_{\mathrm{PIT}}$ is also a WDC, that is, after being extended by the subtyping rules for the parameterised inductive types in PIT, the resulting type theory still has the good properties, and, in particular, it is coherent and satisfies the property of transitivity elimination.

Considering PIT to be an arbitrary set has important practical consequences with regard to the 'stepwise enlargement' of the set of coercive judgements. See Section 5.2 below for further discussion.

5.1. *Coherence, admissibility of transitivity and other properties*

In this subsection we present the lemmas and theorems that show that $\mathscr{C}_{\mathrm{PIT}}$ is a WDC. First, it is not difficult to prove the following lemma.

Lemma 5.1.

1 $\Gamma \vdash A <_c B \in \mathscr{C}_{\mathrm{PIT}}$ implies that $\Gamma \vdash A : Type$, $\Gamma \vdash B : Type$ and $\Gamma \vdash c : (A)B$.
2 $\Gamma \vdash A <_c A \notin \mathscr{C}_{\mathrm{PIT}}$ for any Γ, A and c.
3 If $\Gamma \vdash A <_c B \in \mathscr{C}_{\mathrm{PIT}}$, Γ' *valid*, and $\Gamma \subseteq \Gamma'$, then $\Gamma' \vdash A <_c B \in \mathscr{C}_{\mathrm{PIT}}$.

Proof. We use induction on derivations. We only note that in proving the second clause we need to prove the slightly more general statement

$$\text{if } \Gamma \vdash A <_c B \in \mathscr{C}_{\mathrm{PIT}}, \text{ then } \Gamma \not\vdash A = B : Type,$$

and use the injectivity result (Corollary 4.9). For example, for the non-applicable rule

$$\frac{\Gamma \vdash A <_c B \qquad \Gamma \vdash B <_d A}{\Gamma \vdash \mathscr{T}(A) <_{\mathrm{map}_{\mathscr{T}}} \mathscr{T}(B)}$$

as considered in the remark at the end of Section 3.3, we use the injectivity result to show that $\mathscr{T}(A) = \mathscr{T}(B)$ would imply that $A = B$, which is impossible by the induction hypothesis. □

The following theorem shows that the coherence property still holds after the addition of the structural subtyping rules for the parameterised inductive types in PIT.

Theorem 5.2 (Coherence). The following rule is admissible in $T_\chi[\mathscr{R}_{\mathrm{PIT}}]_0$:

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

Proof. We will prove that the following slightly more general rule is admissible in $T_\chi[\mathscr{R}_{\mathrm{PIT}}]_0$:

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

Let J and J' be $\Gamma \vdash A <_c B$ and $\Gamma \vdash A' <_{c'} B'$, respectively. Then there are two possibilities:

— J and J' are both in $\mathscr{C}$ (the original WDC).
— For some $\mathscr{T} \in$ PIT and some $\bar{A}$, $\bar{B}$, $\bar{A}'$ and $\bar{B}'$:

$$\Gamma \vdash A = \mathscr{T}(\bar{A}) : Type \qquad \Gamma \vdash B = \mathscr{T}(\bar{B}) : Type$$

$$\Gamma \vdash A' = \mathscr{T}(\bar{A}') : Type \qquad \Gamma \vdash B' = \mathscr{T}(\bar{B}') : Type.$$

The former case is trivial (by the assumption that $\mathscr{C}$ is a WDC), and we prove the latter by induction on derivations. In the latter case, the derivations of J and J' must be of the following forms, respectively:

Derivation of J:

$$\dfrac{\dfrac{J_1, \quad ..., \quad J_n}{\Gamma \vdash \mathscr{T}(\bar{A}) <_d \mathscr{T}(\bar{B})}}{\dfrac{(\textit{applications of congruence})}{\Gamma \vdash A <_c B}}$$

Derivation of J':

$$\dfrac{\dfrac{J'_1, \quad ..., \quad J'_{n'}}{\Gamma \vdash \mathscr{T}(\bar{A}') <_{d'} \mathscr{T}(\bar{B}')}}{\dfrac{(\textit{applications of congruence})}{\Gamma \vdash A' <_{c'} B'}}$$

where we have $\Gamma \vdash c = d : (A)B$ and $\Gamma \vdash c' = d' : (A')B'$.

As $\Gamma \vdash A = A' : Type$ and $\Gamma \vdash B = B' : Type$, we have $\Gamma \vdash \mathscr{T}(\bar{A}) = \mathscr{T}(\bar{A}') : Type$ and $\Gamma \vdash \mathscr{T}(\bar{B}) = \mathscr{T}(\bar{B}') : Type$. By the injectivity result (Corollary 4.9), $\Gamma \vdash \bar{A} = \bar{A}' : \bar{P}$ and $\Gamma \vdash \bar{B} = \bar{B}' : \bar{P}$, where $\bar{P}$ are the types of the parameters of $\mathscr{T}$.

Now, consider J_i and J_i' for the case in which they are both subtyping judgements,

$$J_i \equiv \Gamma_i \vdash S_i <_{c_i} T_i \quad \text{and} \quad J_i' \equiv \Gamma_i' \vdash S_i' <_{c_i'} T_i',$$

and assume that we already have $\Gamma_j \vdash c_j = c_j' : (S_j)T_j$ $(j = 1, ..., i-1)$. From the formation of the premises of the structural subtyping rule for $\mathscr{T}$, we have $\Gamma_i = \Gamma_i'$, $\Gamma_i \vdash S_i = S_i' : Type$ and $\Gamma_i \vdash T_i = T_i' : Type$. Therefore, by the induction hypothesis, $\Gamma_i \vdash c_i = c_i' : (S_i)T_i$.

From the above, we have, by the definition of d (from c_i) and d' (from c_i'), that $\Gamma \vdash d = d' : (\mathscr{T}(\bar{A}))\mathscr{T}(\bar{B})$. Therefore, $\Gamma \vdash c = c' : (A)B$. □

Remark. One can see from this proof that the theorem is also true for $T[\mathscr{R}_{\mathrm{PIT}}]$, the type system without the χ-rules.

The following lemma shows that the extension of the system maintains the property of substitution.

Lemma 5.3 (Substitution). If $\Gamma, x{:}K, \Gamma'[x] \vdash A[x] <_{c[x]} B[x] \in \mathscr{C}_{\mathrm{PIT}}$ and $\Gamma \vdash k : K$, then $\Gamma, \Gamma'[k] \vdash A[k] <_{c[k]} B[k] \in \mathscr{C}_{\mathrm{PIT}}$.

Proof. The result is proved by induction on derivations, using the fact that, for $\Delta \equiv \Gamma, x{:}K, \Gamma'[x]$ and $\Delta_k \equiv \Gamma, \Gamma'[k]$, if $\Gamma \vdash k : K$ and

$$C_\Delta(\mathscr{T}; \bar{A}[x]; \bar{B}[x]) = \langle (\Delta, \Delta_i[x];\ T_i[x];\ T_i'[x];\ c_i)\rangle_{i=1,\ldots,n},$$

then

$$C_{\Delta_k}(\mathscr{T}; \bar{A}[k]; \bar{B}[k]) = \langle (\Delta_k, \Delta_i[k];\ T_i[k];\ T_i'[k];\ c_i)\rangle_{i=1,\ldots,n}.$$ □

The following theorem shows that transitivity is admissible in the presence of the χ-rules. The proof is different from that of an earlier result for Π-types and Σ-types (Luo and Luo 2001) in that it proves the admissibility of a more general rule directly by induction on derivations. This has overcome a difficulty in the earlier research effort and avoided the use of a special complexity measure (Chen 1998).

Theorem 5.4 (Admissibility of transitivity). The following rule is admissible in $T_\chi[\mathscr{R}_{\mathrm{PIT}}]_0$:

$$\frac{\Gamma \vdash A <_c B \quad \Gamma \vdash B <_{c'} C}{\Gamma \vdash A <_{c' \circ c} C}.$$

Proof. We prove a more general case, showing that the following rule is admissible in $T_\chi[\mathscr{R}_{\mathrm{PIT}}]_0$:

$$(ST) \qquad \frac{\begin{array}{c}\Gamma, \bar{x}{:}\bar{K} \vdash A[\bar{x}] <_{c[\bar{x}]} B[\bar{x}] \quad \Gamma, \bar{y}{:}\bar{K}' \vdash B'[\bar{y}] <_{c'[\bar{y}]} C[\bar{y}] \\ \Gamma, \Delta \vdash \bar{M} : \bar{K} \quad \Gamma, \Delta \vdash \bar{N} : \bar{K}' \quad \Gamma, \Delta \vdash B[\bar{M}] = B'[\bar{N}] : Type\end{array}}{\Gamma, \Delta \vdash A[\bar{M}] <_{c'[\bar{N}] \circ c[\bar{M}]} C[\bar{N}]}.$$

We obtain the admissibility of the rule in the theorem by taking $B \equiv B'$ and Δ, $\bar{x}$ and $\bar{y}$ as empty (and hence so are $\bar{K}$, $\bar{K}'$, $\bar{M}$ and $\bar{N}$.)

Let $J \equiv \Gamma, \bar{x}{:}\bar{K} \vdash A <_c B$ and $J' \equiv \Gamma, \bar{y}{:}\bar{K}' \vdash B' <_{c'} C$. The proof proceeds by induction on (the sum of the lengths of) the derivations of J and J'. In the base case, J and J' are both in the original WDC $\mathscr{C}$, so the conclusion of the rule is derivable by the definition of WDC and the congruence rule.

In the step case, either J and J' are both in $\mathscr{C}$, in which case it is trivial (as above), or J and J' must be derived by congruence from judgements of the form $\Gamma \vdash \mathscr{T}(...) < \mathscr{T}(...)$ for some parameterised inductive type $\mathscr{T} \in \mathrm{PIT}$. That is, in the latter case, the derivations of J and J' are of the following forms:

Derivation of J:

$$\dfrac{\dfrac{J_1, \quad ..., \quad J_n}{\Gamma,\bar{x}{:}\bar{K} \vdash \mathscr{T}(\bar{A}[\bar{x}]) <_{\mathrm{map}_{\mathscr{T}}[\bar{A},\bar{B},\bar{d}[\bar{x}]]} \mathscr{T}(\bar{B}[\bar{x}])}}{\dfrac{(\textit{applications of congruence})}{\Gamma,\bar{x}{:}\bar{K} \vdash A[\bar{x}] <_{c[\bar{x}]} B[\bar{x}]}}$$

Derivation of J':

$$\dfrac{\dfrac{J'_1, \quad ..., \quad J'_n}{\Gamma,\bar{y}{:}\bar{K}' \vdash \mathscr{T}(\bar{B}'[\bar{y}]) <_{\mathrm{map}_{\mathscr{T}}[\bar{B}',\bar{C},\bar{d}'[\bar{y}]]} \mathscr{T}(\bar{C}[\bar{y}])}}{\dfrac{(\textit{applications of congruence})}{\Gamma,\bar{y}{:}\bar{K}' \vdash B'[\bar{y}] <_{c'[\bar{y}]} C[\bar{y}]}}$$

and we have in $(\Gamma, \bar{x}{:}\bar{K})$:

$$A = \mathscr{T}(\bar{A}),\ B = \mathscr{T}(\bar{B}) \ \text{ and } \ c = \mathrm{map}_{\mathscr{T}}[\bar{A}, \bar{B}, \bar{d}]\,;$$

and in $(\Gamma, \bar{y}{:}\bar{K}')$:

$$B' = \mathscr{T}(\bar{B}'),\ C = \mathscr{T}(\bar{C}) \ \text{ and } \ c' = \mathrm{map}_{\mathscr{T}}[\bar{B}', \bar{C}, \bar{d}']\,.$$

Since $\Gamma, \Delta \vdash B[\bar{M}] = B'[\bar{N}] : \mathit{Type}$, we have $\Gamma, \Delta \vdash \mathscr{T}(\bar{B}[\bar{M}]) = \mathscr{T}(\bar{B}'[\bar{N}]) : \mathit{Type}$. Therefore, by injectivity (Corollary 4.9), $\Gamma, \Delta \vdash \bar{B}[\bar{M}] = \bar{B}'[\bar{N}]$. Because of this and by the induction hypothesis, we can apply (ST) to the judgements $J_1, \ldots, J_n$ and $J'_1, \ldots, J'_n$. Therefore, we can derive, with a subtyping rule for $\mathscr{T}$, the judgement

$$\Gamma, \Delta \vdash \mathscr{T}(\bar{A}[\bar{M}]) <_{\mathrm{map}_{\mathscr{T}}[\bar{A}[\bar{M}],\bar{C}[\bar{N}],\bar{d}'[\bar{N}] \bullet \bar{d}[\bar{M}]]} \mathscr{T}(\bar{C}[\bar{N}])\,.$$

By congruence and the rule (χ), we have

$$\Gamma, \Delta \vdash \mathscr{T}(\bar{A}[\bar{M}]) <_{\mathrm{map}_{\mathscr{T}}[\bar{B}'[\bar{N}],\bar{C}[\bar{N}],\bar{d}'[\bar{N}]] \,\circ\, \mathrm{map}_{\mathscr{T}}[\bar{A}[\bar{M}],\bar{B}[\bar{M}],\bar{d}[\bar{M}]]} \mathscr{T}(\bar{C}[\bar{N}])\,.$$

Then, by congruence, $\Gamma, \Delta \vdash A[\bar{M}] <_{c'[\bar{N}] \,\circ\, c[\bar{M}]} C[\bar{N}]$. □

Example 5.5. This example uses Π-types to illustrate the above proof. Assume that the derivations of the subtyping judgements J and J' are obtained from

$$\dfrac{(*_1)\ \Gamma, \bar{x}{:}\bar{K} \vdash A_2[\bar{x}] <_{c_1[\bar{x}]} A_1[\bar{x}] \quad (*_2)\ \Gamma, \bar{x}{:}\bar{K}, x_2{:}A_2 \vdash B_1[\bar{x}](c_1(x_2)) <_{c_2[\bar{x},x_2]} B_2[\bar{x}](x_2)}{\Gamma, \bar{x}{:}\bar{K} \vdash \Pi(A_1, B_1) <_{\mathrm{map}_{\Pi}} \Pi(A_2, B_2)}$$

$$\dfrac{(*_3)\ \Gamma, \bar{y}{:}\bar{K}' \vdash A_3[\bar{y}] <_{c'_1[\bar{y}]} A'_2[\bar{y}] \quad (*_4)\ \Gamma, \bar{y}{:}\bar{K}', x_3{:}A_3 \vdash B'_2[\bar{y}](c'_1(x_3)) <_{c'_2[\bar{y},x_3]} B_3[\bar{y}](x_3)}{\Gamma, \bar{y}{:}\bar{K}' \vdash \Pi(A'_2, B'_2) <_{\mathrm{map}'_{\Pi}} \Pi(A_3, B_3)}$$

and for some $\bar{M}_0 :: \bar{K}$ and $\bar{N}_0 :: \bar{K}'$, we have $[\bar{M}_0/\bar{x}]\Pi(A_2, B_2) = [\bar{N}_0/\bar{y}]\Pi(A_2', B_2')$. Therefore, by injectivity, $A_2[\bar{M}_0] = A_2'[\bar{N}_0]$ and $B_2[\bar{M}_0] = B_2'[\bar{N}_0]$.

Applying the induction hypothesis to $(*_1)$ and $(*_3)$, we obtain, in Γ,

$$A_3[\bar{N}_0] <_{c_1'[\bar{N}_0]} A_2'[\bar{N}_0] = A_2[\bar{M}_0] <_{c_1[\bar{M}_0]} A_1[\bar{M}_0]\,,$$

which yields

$$\Gamma \vdash A_3[\bar{N}_0] <_{c_1[\bar{M}_0] \circ c_1'[\bar{N}_0]} A_1[\bar{M}_0]\,. \tag{1}$$

Applying the induction hypothesis to $(*_2)$ and $(*_4)$, by taking $\bar{M} \equiv \bar{M}_0 :: c_1'(x_3)$ and $\bar{N} \equiv \bar{N}_0 :: x_3$, we obtain, in $\Gamma, x_3 : A_3$,

$$\begin{array}{rll}
 & [\bar{M}_0/\bar{x}, c_1'[\bar{N}_0](x_3)/x_2]B_1(c_1(x_2)) & \equiv\ [\bar{M}_0/\bar{x}]B_1(c_1(c_1'[\bar{N}_0](x_3))) \\
<_{c_2[\bar{M}_0, c_1'[\bar{N}_0](x_3)]} & [\bar{M}_0/\bar{x}, c_1'[\bar{N}_0](x_3)/x_2]B_2(x_2) & \equiv\ [\bar{M}_0/\bar{x}]B_2(c_1'[\bar{N}_0](x_3)) \\
= & [\bar{N}_0/\bar{y}, x_3/x_3]B_2'(c_1'(x_3)) & \equiv\ [\bar{N}_0/\bar{y}]B_2'(c_1'(x_3)) \\
<_{c_2'[\bar{N}_0, x_3]} & [\bar{N}_0/\bar{y}, x_3/x_3]B_3(x_3) & \equiv\ [\bar{N}_0/\bar{y}]B_3(x_3)\,,
\end{array}$$

which yields

$$\Gamma, x_3 : A_3 \vdash B_1[\bar{M}_0](c_1[\bar{M}_0](c_1'[\bar{N}_0](x_3))) <_{c_2'[\bar{N}_0, x_3] \circ c_2[\bar{M}_0, c_1'[\bar{N}_0](x_3)]} B_3[\bar{N}_0](x_3)\,. \tag{2}$$

Then a Π-subtyping rule can be applied to give, from (1) and (2),

$$\Gamma \vdash \Pi(A_1[\bar{M}_0], B_1[\bar{M}_0]) <_{\mathrm{map}_\Pi^0} \Pi(A_3[\bar{N}_0], B_3[\bar{N}_0])$$

where

$$\mathrm{map}_\Pi^0 \equiv \mathrm{map}_\Pi \left[A_1[\bar{M}_0], B_1[\bar{M}_0], A_3[\bar{N}_0], B_3[\bar{N}_0], \bar{d}'[\bar{N}_0] \bullet \bar{d}[\bar{M}_0]\right]$$

with

$$\begin{array}{rcl}
\bar{d}[\bar{M}_0] & \equiv & \langle c_1[\bar{M}_0],\ [x_3 : A_3]c_2\left[\bar{M}_0, c_1'[\bar{N}_0](x_3)\right]\rangle \\
\bar{d}'[\bar{N}_0] & \equiv & \langle c_1'[\bar{N}_0],\ [x_3 : A_3]c_2'\left[\bar{N}_0, x_3\right]\rangle \\
\bar{d}'[\bar{N}_0] \bullet \bar{d}[\bar{M}_0] & \equiv & \langle c_1[\bar{M}_0] \circ c_1'[\bar{N}_0],\ [x_3 : A_3]\ c_2'\left[\bar{N}_0, x_3\right] \circ c_2\left[\bar{M}_0, c_1'[\bar{N}_0](x_3)\right]\rangle\,.
\end{array}$$

Then the appropriate χ-rule can be applied to give

$$\Gamma \vdash \Pi(A_1[\bar{M}_0], B_1[\bar{M}_0]) <_{\mathrm{map}_\Pi[\bar{d}[\bar{M}_0]] \circ \mathrm{map}_\Pi[\bar{d}'[\bar{N}_0]]} \Pi(A_3[\bar{N}_0], B_3[\bar{N}_0])\,,$$

where we have omitted the sequences of type arguments of the map-operations.

The above theorems and lemmas have shown that $\mathscr{C}_{\mathrm{PIT}}$, the set of derivable subtyping judgements in $T_\chi[\mathscr{R}_{\mathrm{PIT}}]_0$, is a set of well-defined judgements of coercions.

Corollary 5.6. $\mathscr{C}_{\mathrm{PIT}}$ is a WDC.

Proof. The statement follows from Lemma 5.1, Theorem 5.2, Lemma 5.3 and Theorem 5.4. □

5.2. *Stepwise development*

In this section we present some remarks concerning our decision to consider PIT to be an arbitrary set of parameterised inductive types. First, because PIT is an arbitrary set, it does not have to contain all of the inductive types; for example, it does not have to contain the types such as *Nat* and *Even*, the types of natural numbers and even numbers. Hence, in the original WDC $\mathscr{C}$, one may have subtyping judgements such as $Even < Nat$ (the type of even numbers is a subtype of that of natural numbers; see Luo (1999) for more detailed discussions of such subtyping relations in coercive subtyping).

Second, as we mentioned earlier, we want to consider 'stepwise development' in practice, that is, assuming that the existing rules generate well-behaved coercion judgements, an extension will also result in a well-behaved system. More precisely, in our context, we want to consider stepwise extensions by the structural subtyping rules of parameterised inductive types. However, we have not considered 'rule extension' directly because of a difficulty illustrated by the following example. Suppose we extend the system first by *List* and then by Σ (that is, the subtyping judgements generated by *List* is in the WDC $\mathscr{C}$ when we add Σ). Then, if $\Sigma(A, B) < \Sigma(A', B')$, because of the newly added subtyping rules for Σ, we have that $List(\Sigma(A, B)) < List(\Sigma(A', B'))$ would *not* be derivable as we would have expected.

A solution to the above problem is to consider extensions of $\mathscr{C}$ by the subtyping rules of the parameterised inductive types in an *arbitrary* set PIT. For the above example of *List* and Σ, if both *List* and Σ are in PIT, it does not matter which of them is added to the system first. Therefore, the results obtained in this section allow one to add step by step the structural subtyping rules for those parameterised inductive types that are of interest in the application.

Remark. The above point of stepwise development has been a guiding principle in the development of both this work and related earlier work. It is a bit unfortunate that this point was not made more clearly in Luo and Luo (2005) and Luo (2005).

6. Conclusions and future work

We have studied structural subtyping for general inductive types in the framework of coercive subtyping and shown that the extension with structural subtyping rules of parameterised inductive types preserves good meta-theoretic properties such as coherence and transitivity elimination in the presence of the functorial χ-rules for computational equality.

When a type theory has some extensional equality rules, such as the functorial χ-rules, it is in general unknown whether the usual good properties enjoyed by an intensional type theory will still be satisfied: *viz*. Church–Rosser, subject reduction and strong normalisation. Future work includes the study of the meta-theoretic properties of a type theory with the χ-reduction rules. We conjecture that when an intensional type theory is extended with the χ-rules, the resulting calculus should have these properties. A promising approach is to work with a λ-free logical framework such as TF, which may

provide useful tools in such studies. However, it remains to be seen how research in this direction will go.

The proof of the injectivity result as presented in this paper has the restriction that there is no declaration of computational equalities between types in the type theory under consideration. This has excluded type universes. However, we believe that the proof can be extended to include universes, although further research is called for. Also, a direct proof in LF, rather than going through TF, should be possible, although we have so far failed to obtain such a proof.

Soloviev *et al.* have investigated the notion of χ-reduction in a non-dependently typed theory with inductive types (Barral *et al.* 2005). This is why we have called the functorial equality rules 'χ-rules'. As for related work on coercive subtyping, we should mention that, before the study of coercive subtyping, the notion of coercion appeared in the literature for simple type systems (see, for example, Mitchell (1991; 1983)). Some early works on subtyping and dependent types was carried out by Aspinall and Compagnoni in studying subtyping for dependent types in the Edinburgh Logical Framework (Aspinall and Compagnoni 1996; 2001). The early development of the framework of coercive subtyping is closely related to Aczel's idea of type-checking methods in classes with overloading (Aczel 1994) and the work by Breazu-Tannen *et al.* on giving coercion semantics to lambda calculi with subtyping (Breazu-Tannen *et al.* 1991). Barthe and his colleagues have studied constructor subtyping and its possible applications in proof systems (Barthe and Frade 1999; Barthe and van Raamsdonk 2000). Among other research work on coercive subtyping, Y. Luo's Ph.D. work provided an extensive study of structural subtyping for inductive types, especially the notion of weak transitivity (Luo 2005; Luo and Luo 2005).

Subtyping for inductive types in proof assistants is expected to be useful in practice. It would be interesting to see the structural subtyping rules for parameterised inductive types implemented and used in proof assistants that support coercive subtyping.

Acknowledgements

The authors are very grateful to the referees and Paul Callaghan for their very useful comments on earlier versions of the paper, which have helped greatly in improving it.

References

Aczel, P. (1994) Simple overloading for type theories. (Draft.)

Aczel, P. (2001) Yet another logical framework. (Notes – private communication.)

Adams, R. (2004) *A Modular Hierarchy of Logical Frameworks*, Ph.D. thesis, University of Manchester.

Aspinall, D. and Compagnoni, A. (1996) Subtyping dependent types. In: *Proc. 11th Ann. Symp. on Logic in Computer Science*, New Jersey, U.S.A.

Aspinall, D. and Compagnoni, A. (2001) Subtyping dependent types. *Theoretical Computer Science* **266**.

Bailey, A. (1999) *The Machine-checked Literate Formalisation of Algebra in Type Theory*, Ph.D. thesis, University of Manchester.

Barral, F., Soloviev, S. and Chemouil, D. (2005) Inductive type schemas as functors. Talk given at Second International Workshop on Isomorphisms of Types, Toulouse, France.

Barthe, G. and Frade, M. (1999) Constructor subtyping. In: Proceedings of ESOP'99. *Springer-Verlag Lecture Notes in Computer Science* **1576**.

Barthe, G. and van Raamsdonk, F. (2000) Constructor subtyping in the calculus of inductive constructions. In: Proceedings of FOSSACS'00. *Springer-Verlag Lecture Notes in Computer Science* **1784**.

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

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

Chen, G. (1998) *Subtyping, Type Conversion and Transitivity Elimination*, Ph.D. thesis, University of Paris VII.

Coq (2004) *The Coq Proof Assistant Reference Manual (Version 8.0)*, The Coq Development Team, INRIA.

Dybjer, P. (1991) Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In: Huet, G. and Plotkin, G. (eds.) *Logical Frameworks*, Cambridge University Press.

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

Harper, R., Honsell, F. and Plotkin, G. (1993) A framework for defining logics. *Journal of the Association for Computing Machinery* **40** (1) 143–184.

Luo, Y. (2005) *Coherence and Transitivity in Coercive Subtyping*, Ph.D. thesis, University of Durham.

Luo, Y. and Luo, Z. (2001) Coherence and transitivity in coercive subtyping. Proc. of the 8th Inter. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'01), Havana, Cuba. *Springer-Verlag Lecture Notes in Artificial Intelligence* **2250**.

Luo, Y., Luo, Z. and Soloviev, S. (2002) Weak transitivity in coercive subtyping. In: Geuvers, H. and Wiedijk, F. (eds.) Types for Proofs and Programs. *Springer-Verlag Lecture Notes in Computer Science* **2646** 220–239.

Luo, Z. (1994) *Computation and Reasoning: A Type Theory for Computer Science*, Oxford University Press.

Luo, Z. (1997) Coercive subtyping in type theory. In: Proc. of CSL'96, the 1996 Annual Conference of the European Association for Computer Science Logic, Utrecht. *Springer-Verlag Lecture Notes in Computer Science* **1258**.

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

Luo, Z. and Callaghan, P. (1998) Coercive subtyping and lexical semantics (extended abstract). In: *Logical Aspects of Computational Linguistics (LACL'98)*.

Luo, Z. and Luo, Y. (2005) Transitivity in coercive subtyping. *Information and Computation* **197** 122–144.

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

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

McBride, C. and McKinna, J. (2004) The view from the left. *Journal of Functional Programming* **14** (1).

Mitchell, J. C. (1983) Coercion and type inference. In: *Proc. of Tenth Annual Symposium on Principles of Programming Languages (POPL)*.

Mitchell, J. C. (1991) Type inference with simple subtypes. *Journal of Functional Programming* **1** (2) 245–286.

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

Paulin-Mohring, C. (1993) Inductive definitions in the system Coq: rules and properties. Proceedings of the Inter. Conf. on Typed Lambda Calculi and Applications (TLCA'93). *Springer-Verlag Lecture Notes in Computer Science* **664**.

Saïbi, A. (1997) Typing algorithm in type theory with inheritance. In: *Proc of POPL'97*.

Soloviev, S. and Luo, Z. (2002) Coercion completion and conservativity in coercive subtyping. *Annals of Pure and Applied Logic* **113** (1-3) 297–322.