Some Algorithmic and Proof-Theoretical Aspects of Coercive Subtyping

Alex Jones, Zhaohui Luo and Sergei Soloviev
Computer Science Department, Durham University
South Road, Durham DH1 3LE, U.K.
E-mail: A.P.Jones, Zhaohui.Luo, Sergei.Soloviev @durham.ac.uk

Abstract. Coercive subtyping offers a conceptually simple but powerful framework to understand subtyping and subset relationships in type theory. In this paper we study some of its proof-theoretic and computational properties.

1 Introduction

Coercive subtyping, as first introduced in [Luo96], offers a conceptually simple but powerful framework to understand subtyping and subset relationships in type theories with sophisticated type structures such as dependent types, inductive types, and type universes. A basic idea behind coercive subtyping is that subtyping provides a powerful mechanism for notational abbreviation in type theory. If A is a subtype of B given by a specified coercion function, an object of type A can be regarded as an object of type B, that is, its image via the coercion function, and hence objects of a subtype can be used as abbreviations for objects of a supertype.

With coercive subtyping, this abbreviational mechanism is formally treated at the level of the logical framework – the meta-level language used to specify type theories. Given two kinds (meta-level types) K and K', where K is a (proper) subkind of K' and a functional operation f whose source kind is K', one can apply f to any object a of kind K as well as those of kind K'. The meaning of such an application is that $f(a)$ is definitionally equal to $f(c(a))$, where c is the coercion between K and K'.

This simple extension of the logical framework (and hence of the specified type theories such as Martin-Löf's type theory [NPS90] and the type theory UTT [Luo94,Gog94]), provides a surprisingly powerful mechanism that facilitates useful ways of reasoning about subsets of objects, helps proof reuse and modularisation, and gives a proper treatment of the subtyping relation between type universes (see [Luo96] for details). These provide the basis for more efficient subset and substructure reasoning and proof reuse (cf. [Jon95,Bai93]). For instance, if we define the type of groups as a subset of the type of monoids, then any proof to do with monoids can then be automatically applied to groups. Such a mechanism has been implemented by Bailey in LEGO [Bai96] and Saibi in Coq [Sai97] to support proof reuse and notational abbreviation. Similar mechanisms can be used for development of program specification and verification proofs (c.f., [Luo93]).

In this paper, we study some of the basic proof-theoretic and computational properties of coercive subtyping. The formal system we consider, as presented in Section 2, is essentially the same as that in [Luo96] except for one major change: we use subkinding judgements with explicitly associated coercion terms, of the form $K <_c K'$. We also restrict the basic coercions to be closed terms between closed types (as considered in [Luo96]). The presentation of a more general framework of coercive subtyping can be found in [Luo97], which uses subkinding judgements with explicit coercion terms and also allows more general basic subtyping rules such as those for parameterised inductive types.

In section 3 we prove several important meta-theoretic results, including substitution elimination, uniqueness of coercions, presupposition theorems, transitivity elimination, and some results about the relationship between the original type theory and its extension with coercive subtyping. There are at least two important aspects of this meta-theoretic development that deserve attention. One is that we study the meta-theory of the resulting type system with judgemental equality *directly*, without considering a meta-level notion of conversion.[1] Actually, as discussed in the paper, to prove various elimination results requires a careful treatment of the rules in LF by first considering elimination results of their weaker versions.

Another meta-theoretic development reported here is to show that, for the system considered in this paper, the extension with coercive subtyping is conservative over the original type theory. This can be seen as a justification of the claim that coercive subtyping is essentially an abbreviational mechanism [Luo96]. Furthermore, we expect that further development along this line will allow us to transfer the meta-theoretic results for the original type theory to its extension with coercive subtyping.

In section 4 we present a type-checking algorithm for coercive subtyping and prove its soundness using some of the meta-theoretic results of section 3. This is a part of our ongoing effort to study the computational behaviour of type theory with coercive subtyping and different computational strategies for efficient implementation.

Discussions on future research topics can be found in the Conclusion.

2 Coercive subtyping: the formal system

The basic system on which the formal systems considered here are based is the logical framework LF, a typed version of Martin-Löf's logical framework [NPS90]. The formulation of LF and a detailed discussion on how to use LF in

[1] The work by Healfdene Goguen on typed operational semantics (TOS) [Gog94] may be regarded as an indirect development of the meta-theory of LF and UTT, where for example, substitution elimination results are obtained by proving the soundness of TOS wrt the system with substitution rules and the completeness of TOS wrt the system without substitution rules.

specifying type theories (with formal definitions and examples) can be found in Chapter 9 of [Luo94]. Here it is enough to say, that a judgement in a type theory T, specified in LF is derivable if it is derivable in the extension of LF by the constants and computation rules specifying the type theory.

The extension of any type theory T (specified in LF) with coercive subtyping is defined with respect to a set $\mathcal{C}$ of triples (a subtype, a supertype and a coercion term, representing an embedding of a subtype into its supertype). One of our main goals is to investigate the general meta-theoretic properties of this subtyping extension (we shall denote it by T[$\mathcal{C}$]) and its relationship with the original type theory T, since many of them can be studied independently from the underlying type theory[2]. Let us notice that the systems we are going to consider are ordered by inclusion as follows: $LF \subseteq T \subseteq T[\mathcal{C}]$.

2.1 Judgement forms

Besides context validity, we have the following basic judgements forms in the language:

- K:**kind** asserts that K is a *kind*.
- $k{:}K$ asserts that K is the *principal kind* of k.
- $K = K'$ asserts that K and K' are equal kinds.
- $k = k'{:}K$ asserts that k and k' are equal objects with principal kind K.
- $A <_c B$:**Type** asserts that A is a *proper subtype of* B *with coercion* c.
- $K <_c K'$ asserts that K is a *proper subkind of* K' *with coercion* c.

The above are the judgement forms included in our formal presentation. With these judgement forms we can define other judgement forms, for example:

- $K < K'$ (K is a proper subkind of K') stands for '$K <_c K'$ for some c'.
- $K \leq K'$ (K is a subkind of K') stands for '$K = K'$ or $K <_c K'$ for some c'.
- $k \; :: \; K$ (k is of kind K) stands for '$k{:}K$ or $k{:}K_0$ for some K_0 such that $K_0 <_c K$ for some c'.
- $k = k' \; :: \; K$ (k and k' are equal objects of kind K) stands for '$k = k'{:}K$ or $k = k'{:}K_0$ for some K_0 such that $K_0 <_c K$ for some c'.

2.2 The logical framework LF

The logical framework LF is presented as in [Luo94]. We shall denote by $\equiv$ syntactic equality (up to α-conversion, i.e., renaming of bound variables.) Here, we only list the inference rules of LF.

Contexts and assumptions

$$(1.1)\ \frac{}{<>\vdash \textbf{valid}} \qquad (1.2)\ \frac{\Gamma \vdash K{:}\textbf{kind} \quad x \notin FV(\Gamma)}{\Gamma, x{:}K \vdash \textbf{valid}} \qquad (1.3)\frac{\Gamma, x{:}K, \Gamma' \vdash \textbf{valid}}{\Gamma, x{:}K, \Gamma' \vdash x{:}K}$$

[2] Types on meta-level are called *kinds*, and one may say that in this paper we are mostly studying properties of *subkinding*.

General equality rules

$$(2.1)\ \frac{\Gamma \vdash K{:}\mathbf{kind}}{\Gamma \vdash K = K} \qquad (2.2)\ \frac{\Gamma \vdash K = K'}{\Gamma \vdash K' = K} \qquad (2.3)\ \frac{\Gamma \vdash K = K' \quad \Gamma \vdash K' = K''}{\Gamma \vdash K = K''}$$

$$(2.4)\ \frac{\Gamma \vdash k{:}K}{\Gamma \vdash k = k{:}K} \qquad (2.5)\ \frac{\Gamma \vdash k = k'{:}K}{\Gamma \vdash k' = k{:}K} \qquad (2.6)\frac{\Gamma \vdash k = k'{:}K \quad \Gamma \vdash k' = k''{:}K}{\Gamma \vdash k = k''{:}K}$$

Equality typing rules

$$(3.1)\frac{\Gamma \vdash k{:}K \quad \Gamma \vdash K = K'}{\Gamma \vdash k{:}K'} \qquad (3.2)\frac{\Gamma \vdash k = k'{:}K \quad \Gamma \vdash K = K'}{\Gamma \vdash k = k'{:}K'}$$

The kind Type

$$(4.1)\frac{\Gamma \vdash \mathbf{valid}}{\Gamma \vdash \mathbf{Type}{:}\mathbf{kind}} \qquad (4.2)\frac{\Gamma \vdash A{:}\mathbf{Type}}{\Gamma \vdash El(A){:}\mathbf{kind}} \qquad (4.3)\frac{\Gamma \vdash A = B{:}\mathbf{Type}}{\Gamma \vdash El(A) = El(B)}$$

Rules for dependent product kinds and $\beta\eta$-equalities

$$(5.1)\frac{\Gamma, x{:}K \vdash K'{:}\mathbf{kind}}{\Gamma \vdash (x{:}K)K'{:}\mathbf{kind}} \qquad (5.2)\frac{\Gamma \vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash K_1' = K_2'}{\Gamma \vdash (x{:}K_1)K_1' = (x{:}K_2)K_2'}$$

$$(5.3)\frac{\Gamma, x{:}K \vdash k{:}K'}{\Gamma \vdash [x{:}K]k{:}(x{:}K)K'} \qquad (5.4)\frac{\Gamma \vdash K_1 = K_2 \quad \Gamma, x{:}K_1 \vdash k_1 = k_2{:}K}{\Gamma \vdash [x{:}K_1]k_1 = [x{:}K_2]k_2{:}(x{:}K_1)K}$$

$$(5.5)\frac{\Gamma \vdash f{:}(x{:}K)K' \quad \Gamma \vdash k{:}K}{\Gamma \vdash f(k){:}[k/x]K'} \qquad (5.6)\frac{\Gamma \vdash f = f'{:}(x{:}K)K' \quad \Gamma \vdash k_1 = k_2{:}K}{\Gamma \vdash f(k_1) = f'(k_2){:}[k_1/x]K'}$$

$$(5.7)\frac{\Gamma, x{:}K \vdash k'{:}K' \quad \Gamma \vdash k{:}K}{\Gamma \vdash ([x{:}K]k')k = [k/x]k'{:}[k/x]K'} \qquad (5.8)\frac{\Gamma \vdash f{:}(x{:}K)K' \quad x \notin FV(\Gamma)}{\Gamma \vdash [x{:}K]f(x) = f{:}(x{:}K)K'}$$

We shall call EL-kinds the kinds of the form $El(A)$, and product-kinds the kinds of the form $(x{:}K)K'$. A standard notational convention is that if in the kind $(x{:}K)K'$ the variable x does not occur freely in K', we may write $(K)K'$.

The substitution rules below are separated into two groups, since their formal behaviours are quite different (see section 3).

Simple substitutions

$$(6.1)\frac{\Gamma, x{:}K, \Gamma' \vdash \mathbf{valid} \quad \Gamma \vdash k{:}K}{\Gamma, [k/x]\Gamma' \vdash \mathbf{valid}} \qquad (6.2)\frac{\Gamma, x{:}K, \Gamma' \vdash K'{:}\mathbf{kind} \quad \Gamma \vdash k{:}K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K'{:}\mathbf{kind}}$$

$$(6.3)\frac{\Gamma, x{:}K, \Gamma' \vdash k'{:}K' \quad \Gamma \vdash k{:}K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k'{:}[k/x]K'} \qquad (6.4)\ \frac{\Gamma, x{:}K, \Gamma' \vdash K' = K'' \quad \Gamma \vdash k{:}K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' = [k/x]K''}$$

$$(6.5)\frac{\Gamma, x{:}K, \Gamma' \vdash k' = k''{:}K' \quad \Gamma \vdash k{:}K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' = [k/x]k''{:}K'}$$

Equality substitutions

$$(6.6)\ \frac{\Gamma, x{:}K, \Gamma' \vdash K'{:}\mathbf{kind} \quad \Gamma \vdash k_1 = k_2{:}K}{\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]K' = [k_2/x]K'}$$

$$(6.7)\ \frac{\Gamma, x{:}K, \Gamma' \vdash k'{:}K' \quad \Gamma \vdash k_1 = k_2{:}K}{\Gamma, [k/x]\Gamma' \vdash [k_1/x]k' = [k_1/x]k'{:}[k/x]K'}$$

2.3 Coercive subtyping and subkinding

Let T be any type theory specified in LF (without subtyping and subkinding), as described in [Luo94]. We consider two new judgement forms:

- $\Gamma \vdash A <_c B{:}\mathbf{Type}$ asserts that A is a proper subtype of B with coercion c.
- $\Gamma \vdash K <_c K'$ asserts that K is a proper subkind of K' with coercion c.

The label c will be called a coercion term. It will be *proved* that, for any derivable judgement $\Gamma \vdash K <_c K'$, $\Gamma \vdash c{:}(K)K'$ is derivable.

Subtyping We first consider an intermediate system $T[\mathcal{C}]_0$, which spells out how subtyping relations are set up. $T[\mathcal{C}]_0$ is obtained from T by adding the subtyping judgement form $\Gamma \vdash A <_c B{:}\mathbf{Type}$ and the following rules:

- Basic subtyping rules.
- Transitivity rules for subtyping.
- Substitution rule for subtyping.

In this paper, the basic subtyping rules correspond to a set $\mathcal{C}$ of basic coercions:

$$(ST.1)\frac{\Gamma \vdash A{:}\mathbf{Type} \quad \Gamma \vdash B{:}\mathbf{Type} \quad \Gamma \vdash c{:}(El(A))El(B) \quad (A, c, B) \in \mathcal{C}}{\Gamma \vdash A {<_c} B{:}\mathbf{Type}},$$

where A, c, and B do not contain free variables. For a more general framework allowing more general basic subtyping rules, see [Luo97].

Transitivity rules

$$(ST.2)\frac{\Gamma \vdash A <_c B{:}\mathbf{Type} \quad \Gamma \vdash A = A'{:}\mathbf{Type}}{\Gamma \vdash A' <_c B{:}\mathbf{Type}}$$

$$(ST.3)\frac{\Gamma \vdash A <_c B{:}\mathbf{Type} \quad \Gamma \vdash B = B'{:}\mathbf{Type}}{\Gamma \vdash A <_c B'{:}\mathbf{Type}}$$

$$(ST.4)\frac{\Gamma \vdash A <_{c_1} B{:}\mathbf{Type} \quad \Gamma \vdash B <_{c_2} C{:}\mathbf{Type}}{\Gamma \vdash A <_{[x{:}El(A)]c_2(c_1(x))} C{:}\mathbf{Type}}$$

Substitution rule for subtyping

$$(ST.5)\frac{\Gamma, x{:}K, \Gamma' \vdash A <_c B{:}\mathbf{Type} \quad \Gamma \vdash k{:}K}{\Gamma, [k/x']\Gamma' \vdash [k/x]A <_{[k/x]c} [k/x]B{:}\mathbf{Type}}$$

Coherence conditions The following lemma is obvious, because the extension above has no means to obtain a judgement which is not of subtyping form from a subtyping judgement.

Lemma 1. *$T[\mathcal{C}]_0$ is a conservative extension of T. That is, if J is not of the form $A <_c B{:}\mathbf{Type}$, then $\Gamma \vdash J$ is derivable in T if and only if $\Gamma \vdash J$ is derivable in $T[\mathcal{C}]_0$.*

Definition 2. (coherence conditions) We say that $\mathcal{C}$ is coherent if $T[\mathcal{C}]_0$ has the following properties:

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

In this paper, we assume that $\mathcal{C}$ is coherent.

Subkinding A further intermediate system $T[\mathcal{C}]_{0K}$ is obtained by adding the new subkinding judgement form $\Gamma \vdash K <_c K'$, and the following inference rules $(SK.1-8)$.

The basic subkinding rule

$$(SK.1)\frac{\Gamma \ \vdash A <_c B{:}\mathbf{Type}}{\Gamma \vdash El(A) <_c El(B)},$$

Subkinding for dependent products kinds

$$(SK.2)\frac{\Gamma \vdash K_1' <_{c_1} K_1 \quad \Gamma, x'{:}K_1' \vdash [c_1(x')/x]K_2 = K_2' \quad \Gamma, x{:}K_1 \vdash K_2 \ {:}\mathbf{kind}}{\Gamma \vdash (x{:}K_1)K_2 <_c (x'{:}K_1')K_2'},$$

where $c = [f{:}(x{:}K_1)K_2][x'{:}K_1']f(c_1(x'))$;

$$(SK.3)\frac{\Gamma \vdash K_1' = K_1 \quad \Gamma, x{:}K_1' \vdash K_2 <_{c_2} K_2' \quad \Gamma, x{:}K_1 \vdash K_2 \ {:}\mathbf{kind}}{\Gamma \vdash (x{:}K_1)K_2 <_c (x{:}K_1')K_2'},$$

where $c = [f{:}(x{:}K_1)K_2][x{:}K_1']c_2(f(x))$;

$$(SK.4)\frac{\Gamma \vdash K_1' <_{c_1} K_1 \quad \Gamma, x'{:}K_1' \vdash [c_1(x')/x]K_2 <_{c_2} K_2' \quad \Gamma, x{:}K_1 \vdash K_2 \ {:}\mathbf{kind}}{\Gamma \vdash (x{:}K_1)K_2 <_c (x'{:}K_1')K_2'},$$

where $c = [f{:}(x{:}K_1)K_2][x'{:}K_1']c_2(f(c_1(x')))$.

Transitivity rules for subkinding

$$(SK.5)\frac{\Gamma \vdash K_1 <_c K_2 \quad \Gamma \vdash K_2 = K_2'}{\Gamma \vdash K_1' <_{c'} K_2'} \quad (SK.6)\frac{\Gamma \vdash K_1 <_c K_2 \quad \Gamma \vdash K_1 = K_1'}{\Gamma \vdash K_1' <_{c'} K_2'}$$

$$(SK.7)\frac{\Gamma \vdash K <_{c_1} K' \quad \Gamma \vdash K' <_{c_2} K''}{\Gamma \vdash K <_{[x:K]c_2(c_1(x))} K''}$$

Substitution rule for subkinding

$$(SK.8)\frac{\Gamma, x{:}K, \Gamma' \vdash K_1 <_c K_2 \quad \Gamma \vdash k{:}K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K_1 <_{[k/x]c} [k/x]K_2}$$

Lemma 3. *$T[\mathcal{C}]_{0K}$ is a conservative extension of T and $T[\mathcal{C}]_0$.*

Because $T[\mathcal{C}]_{0K}$ is conservative extension of $T[\mathcal{C}]_0$, coherence conditions hold also in $T[\mathcal{C}]_{0K}$.

Coercive rules The extension of T with coercive subtyping, the system $T[\mathcal{C}]$, is obtained from $T[\mathcal{C}]_{0K}$ by adding the following rules, which establish the essential connection between the original system T and its subtyping/subkinding extension.

New rules for application

$$(CA.1)\frac{\Gamma \vdash f{:}(x{:}K)K' \quad \Gamma \vdash k{:}K_0 \quad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f(k){:}[c(k)/x]K'}$$

$$(CA.2)\frac{\Gamma \vdash f = f'{:}(x{:}K)K' \quad \Gamma \vdash k_1 = k_2{:}K_0 \quad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f(k_1) = f'(k_2){:}[c(k_1)/x]K'}$$

Coercive definition rule

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

Note that, unlike $T[\mathcal{C}]_0$, it is not at all obvious that $T[\mathcal{C}]$ is a conservative extension of T, because now there are means to derive judgements of other forms using subtyping/subkinding judgements. See subsection 3.5 for further discussions.

3 Basic Meta-Theoretical Properties

A relatively large calculus, like our T[$\mathcal{C}$], contains many groups of rules, interacting in a complex and not always obvious way. To make it a really efficient instrument for the development of the meta-theory, it is necessary to investigate first its basic properties. This includes several results which may be considered as forms of (partial) cut-elimination. Take, for example, elimination of the substitution rules and the transitivity rules for subkinding. Such results will be very helpful in the part, corresponding to the typechecking algorithm.

Another important question is that of "presupposed" judgements and their derivations (one of the characteristic features of dependent type theories). For example, the derivability of a judgement

$$x_1{:}Q_1, ..., x_n{:}Q_n \vdash K_1 = K_2$$

presupposes conceptually the derivability of judgements

$$x_1{:}Q_1, ..., x_{i-1}{:}Q_{i-1} \vdash Q_i{:}\mathbf{kind} \quad x_1{:}Q_1, ..., x_i{:}Q_i \vdash \mathbf{valid}(i \leq n)$$

and $x_1{:}Q_1, ..., x_n{:}Q_n \vdash K_j{:}\mathbf{kind}(j = 1, 2)$. If instead of $K_1 = K_2$ we take $k_1 = k_2{:}K$, presupposed judgements will include $x_1{:}Q_1, ..., x_n{:}Q_n \vdash K{:}\mathbf{kind}$ and $x_1{:}Q_1, ..., x_n{:}Q_n \vdash k_j{:}K(j = 1, 2)$. Subkinding $K_1 <_c K_2$ presupposes in addition the derivability of $x_1{:}Q_1, ..., x_n{:}Q_n \vdash c{:}(K_1)K_2$ etc. But how may a complex judgement be "split", how do the derivations of the "presupposed" judgements correspond to the main derivation?

Neither the elimination problem nor the "splitting" of complex judgements admit straightforward solutions.

As an illustration of the possible difficulties, let us consider several examples. They will also give us a better understanding of the order in which the lemmas leading to the main results are organised.

Let us try, for example, to eliminate equality substitutions using some sort of induction on derivations. Consider the inference

$$(6.7)\frac{(3.2)\dfrac{\Gamma, x{:}K, \Gamma' \vdash k{:}K'' \quad \Gamma, x{:}K, \Gamma' \vdash K'' = K'}{\Gamma, x{:}K, \Gamma' \vdash k{:}K'} \qquad \Gamma \vdash k_1 = k_2{:}K}{\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]k = [k_2/x]k{:}[k_1/x]K'}$$

It seems natural to apply substitution and the inductive hypothesis to the left premise of 3.2. The result is $\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]k = [k_2/x]k{:}[k_1/x]K'$.

To replace the kind $[k_1/x]K''$ by $[k_1/x]K'$ one may derive the judgement $\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]K'' = [k_1/x]K'$ by simple substitution from $\Gamma' \vdash K'' = K'$. But to obtain the premise $\Gamma \vdash k_1{:}K$ for this substitution we need to "split" $\Gamma \vdash k_1 = k_2{:}K$.

Meanwhile, "splitting" itself poses problems. To prove it, again by some induction on derivations, new substitutions could be necessary, as in case of a derivation, ending by

$$(5.6)\frac{\Gamma \vdash f = f'{:}(x{:}K)K' \quad \Gamma \vdash k_1 = k_2{:}K}{\Gamma \vdash f(k_1) = f'(k_2){:}[k_1/x]K'}$$

(By "splitting" premises and using application without equality 5.5 one obtains for the right part only $\Gamma \vdash f'(k_2){:}[k_2/x]K$ whereas the kind should be $[k_1/x]K$.) In other cases one may even need a rule which is not included in the formulation of the system.

Consider, for example, a derivation ending with

$$\frac{\Gamma, x{:}K_1 \vdash K_1' = K_2' \quad \Gamma \vdash K_1 = K_2}{\Gamma \vdash (x{:}K_1)K_1' = (x{:}K_2)K_2'}(5.2)$$

By the inductive hypothesis applied to the left premise, one obtains $\Gamma, x{:}K_1 \vdash K_1'$ and $\Gamma, x{:}K_1 \vdash K_2'$. From the first judgement one has, by 5.1, $\Gamma \vdash (x{:}K_1)K_1'$, as required. But 5.1 applied to the second will give $\Gamma \vdash (x{:}K_1)K_2'$. The kind K_1 should be changed to K_2 first. To save the situation, one apparently needs the rule

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

We solve these problems by considering first some weaker, less problematic rules, proving, with their help, a variant of the "split-theorem" and only then passing to the rules that present the main difficulties.

3.1 Some basic lemmas

The next three lemmas hold for $\mathrm{T}[\mathcal{C}]$ and any of its subsystems and are proved by straightforward induction on the length of derivation.

Lemma 4. *(a) If a judgement of the form $\Gamma \vdash K{:}\mathbf{kind}$ is derivable, then K is either an El-kind, product-kind or the kind* **Type**. *(b) If $\Gamma \vdash K = K'$ is derivable, then both K, K' are either El-kinds or product-kinds or else the kind* **Type**. *(c)If $\Gamma \vdash K <_c K'$ is derivable, then both K, K' are either El-kinds or product-kinds.*

Lemma 5. *If $\Gamma \vdash J$, then $FV(J) \subseteq FV(\Gamma)$*

Lemma 6. *If $\Gamma, x{:}A \vdash$* **valid**, *then $x \notin FV(\Gamma)$.*

3.2 Elimination results

Let $\mathrm{T}[\mathcal{C}]^-(T[\mathcal{C}]_0^-, T[\mathcal{C}]_{0K}^-)$ be the system obtained from $\mathrm{T}[\mathcal{C}](T[\mathcal{C}]_0, T[\mathcal{C}]_{0K}$ respectively) by removing

- the substitution rules in LF(rules 6.1-6.7);
- the substitution rules for subtyping and subkinding (rules ST.5, SK.8).

We shall show, that every derivation in $\mathrm{T}[\mathcal{C}]$may be transformed into a $\mathrm{T}[\mathcal{C}]^-$-derivation of the same judgement.

Temporarily we need a system with even more restricted rules, where some subtyping and subkinding rules have more premises than usual. We take

$$(ST.2')\frac{\Gamma \vdash A <_c B{:}\mathbf{Type} \;\; \Gamma \vdash A = A'{:}\mathbf{Type} \;\; \Gamma \vdash A'{:}\mathbf{Type}}{\Gamma \vdash A' <_c B{:}\mathbf{Type}}(= -right)$$

instead of ST.2. We add in a similar way the premise $\Gamma \vdash B'$:**Type** to ST.3, the premise $\Gamma, x':K_1' \vdash K_2'$:**kind** to SK.2-4 (product-introductions for subkinding), the premises $\Gamma \vdash K_2'$:**kind**, $\Gamma \vdash K_1'$:**kind** and $\Gamma \vdash c':(K)K'$ to $SK.5(= -right)$ and $SK.6(= -left)$.

Let us call this system $T[\mathcal{C}]^=$. The following lemma holds for $T[\mathcal{C}]^-$as well as $T[\mathcal{C}]^=$ and is proved by straightforward induction on the size (number of rules) of the derivation.

Lemma 7. *a)Every derivation of* $\Gamma_1, \Gamma_2 \vdash J$ *contains a sub-derivation of* $\Gamma_1 \vdash$ **valid** *in the same system. b)Every derivation of* $\Gamma_1, x:K, \Gamma_2 \vdash J$ *contains a sub-derivation of* $\Gamma_1 \vdash K$:**kind** *in the same system. c) Every derivation of* $\Gamma \vdash (x:K_1)K_2$ *contains a sub-derivation of* $\Gamma, x:K_1 \vdash K_2$.

Lemma 8. *There are algorithms that transform every derivation of a judgement of the form* $\Gamma \vdash A <_c B$:**Type** *(of the form* $\Gamma \vdash K <_c K'$*) in* $T[\mathcal{C}]^=$ *into derivations of* $\Gamma \vdash A$:**Type**, $\Gamma \vdash B$:**Type**, $\Gamma \vdash c:(El(A))El(B)$ *(respectively derivations of* $\Gamma \vdash K$:**kind**, $\Gamma \vdash K'$:**kind**, $\Gamma \vdash c:(K)K'$*) in the same calculus.*

Proof. Straightforward induction (the presence of the extra premise is essential for it). □

Lemma 9. *(weakening) There is an algorithm that transforms every derivation in* $T[\mathcal{C}]^=$ *extended by the following weakening rule:*

$$\frac{\Gamma_1, \Gamma_2 \vdash J \quad \Gamma_1, \Gamma_3 \vdash \mathbf{valid}}{\Gamma_1, \Gamma_3, \Gamma_2 \vdash J}(wkn)$$

(where $FV(\Gamma_2) \cap FV(\Gamma_3) = \emptyset$*) into a* $T[\mathcal{C}]^=$*-derivation of the same judgement.*

Proof. First we consider the case of a derivation containing exactly one wkn at the end, and prove the lemma by induction on the size of the derivation of its left premise. Then the proof is completed by trivial induction on the number of wkn rules (topmost are eliminated first). □

Lemma 10. *(Elimination of simple substitutions.) There is an algorithm that transforms every derivation in* $T[\mathcal{C}]^= \cup \{6.1 - 6.5\}$ *into a derivation in* $T[\mathcal{C}]^=$.

Proof is essentially similar to the proof of lemma 9, though contains more cases. □

Lemma 11. *(Elimination of retypings.) There is an algorithm that transforms every derivation in* $T[\mathcal{C}]^=$ *extended by the following rule*

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

(J stands for any form of judgement) into a derivation in $T[\mathcal{C}]^=$.

Proof as above. □

In the following lemma we consider "weak" equality substitutions:

$$(6.6')\ \frac{\Gamma, x{:}K, \Gamma' \vdash K'{:}\mathbf{kind} \quad \Gamma \vdash k_1 = k_2{:}K \quad \Gamma \vdash k_1{:}K \quad \Gamma \vdash k_2{:}K}{\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]K' = [k_2/x]K'}$$

$$(6.7')\ \frac{\Gamma, x{:}K, \Gamma' \vdash k'{:}K' \quad \Gamma \vdash k_1 = k_2{:}K \quad \Gamma \vdash k_1{:}K \quad \Gamma \vdash k_2{:}K}{\Gamma, [k/x]\Gamma' \vdash [k_1/x]k' = [k_1/x]k'{:}[k/x]K'}$$

Lemma 12. *There is an algorithm that transforms every derivation in $T[\mathcal{C}]^= \cup \{6.6', 6.7'\}$ into a derivation in $T[\mathcal{C}]^=$*

Proof as above; in each case when "splitting" would be necessary for ordinary 6.6., 6.7 the extra premises are used. □

3.3 The "Split-theorem" and its consequences

Theorem 13. *(Split-theorem) In $T[\mathcal{C}]^=$ there are algorithms that transform every derivation of*

1. *$\Gamma \vdash K_1 = K_2$ into a derivation of $\Gamma \vdash K_1{:}\mathbf{kind}$ and of $\Gamma \vdash K_2{:}\mathbf{kind}$;*
2. *$\Gamma \vdash k_1 = k_2{:}K$ into a derivation of $\Gamma \vdash k_1{:}K$ and $\Gamma \vdash k_2{:}K$;*
3. *$\Gamma \vdash \Sigma{:}K$ into a derivation of $\Gamma \vdash K{:}\mathbf{kind}$ (Σ denotes here term or term equality);*

Proof By simultaneous induction on the size of the main derivation, using elimination lemmas 9, 10, 11, 12 and lemma 8. To illustrate why only weak formulations of the rules are enough, let's consider statement 1). Assume that the derivation ends with

$$\frac{\Gamma, x{:}K_1 \vdash K_1' = K_2' \quad \Gamma \vdash K_1 = K_2}{\Gamma \vdash (x{:}K_1)K_1' = (x{:}K_2)K_2'}(5.2)$$

By applying the inductive hypothesis to the left premise we obtain $\Gamma, x{:}K_1 \vdash K_1'$ and $\Gamma, x{:}K_1 \vdash K_2'$. From the first judgement we have by 5.1 $\Gamma \vdash (x{:}K_1)K_1'$, as required. We apply the inductive hypothesis to the second premise of 5.2 and obtain a derivation of $\Gamma \vdash K_2$ (this is why only "retyping" with three premises is needed.) Now

$$(5.1)\frac{\dfrac{\Gamma, x{:}K_1 \vdash K_2' \quad \Gamma \vdash K_1 = K_2 \quad \Gamma \vdash K_2}{\Gamma, x{:}K_2 \vdash K_2'}}{\Gamma \vdash (x{:}K_2)K_2'}$$

The weak "retyping" may be eliminated by lemma 11. In other cases weakenings, substitutions and equality substitutions may appear, but the inductive hypothesis always allows us to derive extra premises and use appropriate lemmas. □.

Remark 14. The following rule

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

is admissible. We shall call it "context-retyping".

The following lemma clarifies the relationship between T$[\mathcal{C}]^-$and $T[\mathcal{C}]^=$.

Lemma 15. *There is an algorithm that transforms every derivation in T[$\mathcal{C}$]$^-$ into a derivation of the same judgement in* $T[\mathcal{C}]^=$*, and an inverse algorithm from* $T[\mathcal{C}]^=$*-derivations to T[$\mathcal{C}$]$^-$-derivation.*

Proof. Theorem 13 is used to obtain the extra premises with their derivations in the algorithm that transforms T$[\mathcal{C}]^-$-derivations into $T[\mathcal{C}]^=$-derivations. The inverse algorithm cuts off the extra premises. □

Theorem 16. *There are algorithms that transform every derivation in T[$\mathcal{C}$]into a derivation of the same judgement in* $T[\mathcal{C}]^=$ *and T[$\mathcal{C}$]$^-$.*

Proof is similar to the proof of the previous lemma, using in addition to the "split-theorem" (which provides extra premises) the elimination lemmas. It also uses previous lemma. □

Note, that the reduction above does not change coercion terms at all. Thus, lemma 4 holds for the whole T$[\mathcal{C}]$.

The following lemmas are easy consequences of results above. They hold in T$[\mathcal{C}]$and its subsystems $T[\mathcal{C}]^-$ and $T[\mathcal{C}]^=$.

Lemma 17. *(a)If* $\Gamma \vdash (x_1{:}K_1)Q$*, then* $\Gamma \vdash K_1{:}\mathbf{kind}$ *and* $\Gamma, x_1{:}K_1 \vdash Q{:}\mathbf{kind}$.
(b)If $\Gamma \vdash (x_1{:}K_1)Q = (x_1{:}K_1')Q'$ *then* $\Gamma \vdash K_1 = K_1'$ $\Gamma, x_1{:}K_1 \vdash Q = Q'$.

Lemma 18. *If* $\Gamma \vdash [x{:}A]k{:}(x{:}A')K$*, then* $\Gamma, x{:}A \vdash k{:}K$ *and* $\Gamma \vdash A = A'{:}\mathbf{kind}$.

Lemma 19. *The coercion terms are unique up to equality in T[$\mathcal{C}$]. If* $\Gamma \vdash K <_{c_1} K'$, $\Gamma \vdash K <_{c_2} K'$ *derivable in T[$\mathcal{C}$], then* $\Gamma \vdash c_1 = c_2{:}(K)K'$.

Proof. We may assume that the derivations belong to T$[\mathcal{C}]^-$. By lemma 8, $\Gamma \vdash K$, $\Gamma \vdash K'$. The two judgements are used to derive $\Gamma, x{:}K \vdash \mathbf{valid}$, $\Gamma, x{:}K \vdash x{:}K$ and $\Gamma \vdash [y{:}K']y{:}(K')K'$ (we assume x, y fresh). Now we add $x{:}K$ to Γ in $\Gamma \vdash K <_{c_i} K'$, $\Gamma \vdash [y{:}K']y{:}(K')K'$ by *wkn* and apply the coercive definition rule CD to obtain $\Gamma, x{:}K \vdash [y{:}K']y(c_i(x)) = ([y{:}K']y)x{:}K'$. Using the β-rule we derive $\Gamma, x{:}K \vdash c_i(x) = ([y{:}K']y)x{:}K'$, and, by the symmetry and transitivity of $=$, $\Gamma, x{:}K \vdash c_1(x) = c_2(x){:}K'$. From this, $\Gamma \vdash [x{:}K]c_1(x) = [x{:}K]c_2(x){:}(K)K'$. With help of the η-rule we obtain $\Gamma \vdash c_1 = c_2{:}(K)K'$. □

Theorem 20. *If* $\Gamma \vdash k{:}K$ *and* $\Gamma \vdash k{:}K'$ *are derivable in T[$\mathcal{C}$], then* $\Gamma \vdash K = K'{:}\mathbf{kind}$ *in T[$\mathcal{C}$].*

Proof. We may assume that both derivations belong to T[$\mathcal{C}$]$^-$. The proof is by induction on the sum of sizes of derivations using the previous lemma. □

Remark 21. The proofs of all lemmas and theorems above (except lemma 19) go through without essential changes if we replace T[$\mathcal{C}$]and T[$\mathcal{C}$]$^-$by $T[\mathcal{C}]_0$ and $T[\mathcal{C}]_0^-$, $T[\mathcal{C}]_{0K}$ and $T[\mathcal{C}]_{0K}^-$ respectively. (We have only to drop some cases.)

Lemma 19 will still be true (with T-equality instead of T[$\mathcal{C}$]-equality), but its proof has to be changed (see below).

The proof of lemma 19 shows the power of the coercive rules more so than anything about the nature of T[$\mathcal{C}$]-equality and its relationship to the equality in T. This provides additional motivation to consider the conservativity issue.

3.4 Elimination of transitivity of subkinding

To prove the following lemmas, we introduce a *rank* of kinds in such a way, that $rank(El(k)) = 0$, $rank((x{:}K_1)K_2) = max(rank(K_1), rank(K_2)) + 1$, and $rank([k/x]K) = rank(k)$ (the last property holds, because substitution into kinds will always fall in scope of some El). The lemmas are proved by induction on rank.

Lemma 22. *There is an algorithm that transforms every derivation d of a subkinding judgement $\Gamma \vdash K <_c K'$ in T[$\mathcal{C}$]($T[\mathcal{C}]_{0K}$), where d has exactly one application of one of =-left ($SK.5$), =-right ($SK.6$) into a derivation d' of the same judgement in T[$\mathcal{C}$]($T[\mathcal{C}]_{0K}$) which does not contain transitivity rules at all.*

Lemma 23. *There is an algorithm that transforms every derivation d of a subkinding judgement $\Gamma \vdash K <_c K'$ in T[$\mathcal{C}$]($T[\mathcal{C}]_{0K}$), where d has exactly one application of $SK.7$ at the end, into a derivation d' of the judgement $\Gamma \vdash K <_{c'} K'$, and $\Gamma \vdash c = c'{:}(K)K'$ in T[$\mathcal{C}$](in T).*

Theorem 24. *(Elimination of transitivity for subkinding in $T[\mathcal{C}]_{0K}$.) a)There is an algorithm that transforms every derivation of the judgement $\Gamma \vdash K <_c K'$ in $T[\mathcal{C}]_{0K}$ into a derivation of the judgement $\Gamma \vdash K <_{c'} K'$ in the same calculus, not containing the rules $SK.5-7$, and such that $\Gamma \vdash c' = c{:}(K)K'$ in T.*

b)If $\Gamma \vdash K <_c K'$ and $\Gamma \vdash K <_{c'} K'$ in $T[\mathcal{C}]_{0K}$, then $\Gamma \vdash c' = c{:}(u{:}K)K'$ in T.

*Proof.*a) It is enough to consider the derivations in $T[\mathcal{C}]_{0K}^-$. We use induction on the number of transitivity rules, using previous lemmas.

b) If $\Gamma \vdash K <_c K'$ and $\Gamma \vdash K <_{c'} K'$ are derived without transitivity, the statement follows (by induction on derivations) from coherence conditions. But every coercion term , by a), is T-equal to the term corresponding to some derivation without transitivity. □

The situation in T[$\mathcal{C}$]is complicated by the presence of coercive application and definition rules. Note, that uniqueness of coercion terms (up to $T[\mathcal{C}]$-equality) is already proved (lemma 19).

Theorem 25. *(Elimination of transitivity for subkinding in $T[\mathcal{C}]$.) There is an algorithm that transforms every $T[\mathcal{C}]$-derivation of the judgement $\Gamma \vdash J$, where J does not have subkinding form, into a derivation of the same judgement, and also every derivation of $\Gamma \vdash K <_c K'$ into a derivation of the judgement $\Gamma \vdash K <_{c'} K'$, not containing the rules $SK.5-7$, and such that $\Gamma \vdash c' = c{:}(K)K'$ in $T[\mathcal{C}]$.*

Proof. We may consider derivations in $\mathrm{T}[\mathcal{C}]^-$. The proof proceeds by induction on the number of judgements in the derivation, whose sub-derivation contains at least one application of transitivity rules $SK.5-7$. We consider the last rule in the derivation and apply inductive hypothesis to its premises. If this last rule was itself transitivity, we use afterwards lemma 22 or 23. Of the rest, only the coercive rules CA.1, CA.2, CD present some difficulties. The inductive hypothesis guarantees only that the premise $\Gamma \vdash K <_c K'$ will be replaced by $\Gamma \vdash K <_{c'} K'$ with $\Gamma \vdash c' = c{:}(K)K'$. To have in the conclusion a judgement which is identical to the original one, we derive appropriate equality between kinds and use 3.1. In case of CD we have to derive also equality between terms and use 2.6. □

3.5 Results about coercion completion

The safety of coercions as an abbreviational mechanism relies strongly on the possibility to insert in some uniform way all the coercions which were omitted (when CA.1, CA.2 and CD were applied). Presumably, a derivation in the underlying type theory T should be obtained.

Consider an inference of, say, coercive application rule $CA.1$.

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

Assume that there exist some T-derivations of the premises $\Gamma \vdash f{:}(x{:}K)K'$ and $\Gamma \vdash k{:}K_0$, and a $T[\mathcal{C}]_{0K}$-derivation of $\Gamma \vdash K_0 <_c K$. From the $T[\mathcal{C}]_{0K}$-derivation of $\Gamma \vdash K_0 <_c K$ we obtain by lemma 4 a T-derivation of $\Gamma \vdash c{:}(K_0)K$. Now we obtain a T-derivation of the judgement $\Gamma \vdash f(c(k)){:}[c(k)/x]K'$ (using the LF-rules) as follows:

$$\cfrac{\cfrac{}{\Gamma \vdash f{:}(x{:}K)K'} \quad \cfrac{\Gamma \vdash c{:}(K_0)K \quad \Gamma \vdash k{:}K_0}{\Gamma \vdash c(k){:}K}(5.1)}{\Gamma \vdash f(c(k)){:}[c(k)/x]K'}(5.1).$$

The coercive equality application rule and coercive definition rule may be modified in the same way.

This construction suggests an idea of how to define a transformation Θ on the whole derivation. We should begin at the top, and move to the bottom replacing subkinding judgements in the premises of the coercive rules CA.1, CA.2 and CD by the derivations of their coercion terms, and modifying the rules accordingly. The intended result is a T-derivation of the judgement forms which are present in

T, and a derivation in its conservative extensions $T[\mathcal{C}]_0$ or $T[\mathcal{C}]_{0K}$ for subtyping and subkinding judgements respectively.

This idea will work correctly only if we can guarantee that the premises of all rules will be matching (at least, up to equality in T). Note, that even the identical kinds or terms may be modified in different ways in different derivations, since different coercion terms may be inserted.

For example, if we consider the ordinary application rule

$$\frac{\Gamma \vdash f{:}(x{:}K_1)K_2 \quad \Gamma \vdash k{:}K_1}{\Gamma \vdash f(k){:}[k/x]K_2},$$

and assume that some T[$\mathcal{C}$]-derivations of its premises, say, d_1, d_2 became the derivations $\Theta(d_1), \Theta(d_2)$ in T of $\Gamma' \vdash f'{:}(x{:}K_1')K_2'$ and $\Gamma'' \vdash k''{:}K_1''$ respectively, then the corresponding kinds in Γ' and Γ'' should be equal in T, and the same for K_1' and K_1''. If they are T-equal, we insert the appropriate instances of the equality rules 3.1, 3.2, of the admissible rule 3.3, and then use the same rule as in the main derivation.

We say that the transformation Θ is defined for a derivation d if the construction outlined above can be performed on the whole of d. In order to use technical results, obtained above, we define it on the class of derivations, including substitutions, the rules wkn, $context - retyping$ and the rules with extra premises (they may be eliminated later from the resulting derivation). If a derivation does not contain coercive rules then Θ is the identity and the derivation is not changed. In principle, how a judgement is modified depends on its derivation and if we have two derivations of the same judgement, then applying Θ we may obtain two T-derivations of two different judgements.

We have a proof-sketch of the following results. (The technical details still have yet to be checked.)

Lemma 26. *Let d be a derivation. Assume that one of the lemmas 9, 10, 11, 12 or 8 holds. Let d' be obtained from d by the algorithm described in that lemma. If Θ is defined for d then it is defined for d' and the final judgements of $\Theta(d)$ and $\Theta(d')$ are equal in T component-wise (i.e., they have the same form, their contexts have equal length, the kinds of corresponding variables are T-equal and so on).*

Lemma 27. *Let d be a derivation of some judgement of the form $\Gamma \vdash J$. Let $\Gamma \vdash J'$ be one of its presupposed judgements as described in theorem 13, and d' be its derivation, obtained by the corresponding algorithm. If Θ is defined for d then it is defined for d'. If $\Theta_d(\Gamma \vdash J)$ is the final judgement of $\Theta(d)$, then the final judgement of $\Theta(d')$ is T-equal component-wise to the corresponding presupposed judgement of $\Theta_d(\Gamma \vdash J)$.*

Lemma 28. *Let d be a derivation. Assume that theorem 16 holds. Let d' be obtained from d by the algorithm described in that theorem. If Θ is defined for d then it is defined for d' and the final judgements of $\Theta(d)$ and $\Theta(d')$ are equal in T component-wise.*

Theorem 29. *The transformation Θ is defined for all derivations in T[$\mathcal{C}$]and if d, d' are derivations of the same judgement $\Gamma \vdash J$, then the judgements $\Theta_d(\Gamma \vdash J)$ and $\Theta'_d(\Gamma \vdash J)$ are T-equal.*

Corollary 30. *If $\Gamma \vdash K = K'$ in T[$\mathcal{C}$], and $\Gamma \vdash K$:**kind**, $\Gamma \vdash K'$:**kind** in T, then $\Gamma \vdash K = K'$:**kind** is derivable in T.*

Proof. Applying Θ to the derivation d of $\Gamma \vdash K = K'{:}\mathbf{kind}$ in T[$\mathcal{C}$]and to the corresponding derivations d', d'' of the presupposed judgements $\Gamma \vdash K{:}\mathbf{kind}$ and $\Gamma \vdash K'{:}\mathbf{kind}$, we obtain the T-derivations of $\Theta_d(\Gamma \vdash K = K'{:}\mathbf{kind})$, $\Theta'_d(\Gamma \vdash K{:}\mathbf{kind})$ and $\Theta''_d(\Gamma \vdash K')$. By lemma 27, the corresponding parts of these judgements are T-equal. Let d_0, d'_0 be some derivations of $\Gamma \vdash K{:}\mathbf{kind}, \Gamma \vdash K'{:}\mathbf{kind}$.

$$\Theta_{d_0}(\Gamma \vdash K{:}\mathbf{kind}) \equiv \Gamma \vdash K{:}\mathbf{kind}, \Theta_{d'_0}(\Gamma \vdash K'{:}\mathbf{kind}) \equiv \Gamma \vdash K'{:}\mathbf{kind}$$

Applying theorem 29 to d', d_0, d'', d'_0 respectively, and putting together all T-equalities, we see, that $\Gamma \vdash K = K'$ in T. □

4 A type-checking algorithm and its soundness proof

In this section, we present a type-checking algorithm, which is based on Coquand's algorithm described in [Coq91] but uses the notion of *typed reduction*, and prove its soundness using the meta-results in section 3.

4.1 The notion of typed reduction

The intended notion of computation for the extended type theory T[$\mathcal{C}$] is the notion of *typed reduction*, which is is the reflexive and transitive closure generated from the following rules and the computation rules for the original type theory T.

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

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

$$\frac{\Gamma \vdash f \rhd f'{:}(x{:}K)K' \quad \Gamma \vdash k_1 \rhd k_2{:}K_0 \quad \Gamma \vdash K_0 <_c K}{\Gamma \vdash f(k_1) \rhd f'(k_2){:}[c(k_1)/x]K'}$$

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

Note that typed reduction is restricted by principal kinding requirements, in contrast to the usual untyped reduction. For instance, if $\Gamma \vdash K_1 <_c K_2$, then with untyped reduction, we have

$$[x{:}K_1]([y{:}K_2]y)(x) \rhd_\beta [x{:}K_1]x$$
$$[x{:}K_1]([y{:}K_2]y)(x) \rhd_\eta [y{:}K_2]y$$

Hence, there are two different normal forms. With a typed reduction strategy, neither of the above reductions can be made. For the first, x would have to have kind K_2 and for the second the target of $[y{:}K_2]y$ would have to be K_1. In fact, with the typed reduction strategy the above reduction steps would not be admissible, but what we can do is the following:

$$\begin{aligned} [x{:}K_1]([y{:}K_2]y)(x) &\rhd_c [x{:}K_1]([y{:}K_2]y)(c(x)) \\ &\rhd_\beta [x{:}K_1]c(x) \\ &\rhd_\eta c \end{aligned}$$

4.2 Type-checking

In type-checkers such as LEGO ([LP92]) and Coq ([H$^+$96]) type-checking is based upon the methods described in [Hue89]. The main thrust here is placed on a conversion algorithm as it is the most tricky part of type-checking. Conversion in these systems is tested by reducing both terms to a head normal form and then by recursively checking the structure of both head normal forms. It is important to note here that this method gives us a reduction strategy (outward reductions are performed first) and that both reduction and conversion are untyped. This then gives us a rather efficient method of type-checking.

Our algorithm is based on Coquand's algorithm described in [Coq91] which deals with $\beta\eta$-conversion. The method of checking whether two terms are convertible here is to reduce both to an unkinded β-weak head-normal form (whnf). The structures of the whnfs are then compared and if the terms are of the form $\lambda x.M$ and N, then M and $N.x$ are tested, thus checking for η-conversion.

In the presence of coercive subtyping, an unkinded algorithm leads away from uniqueness of normal forms as demonstrated in the example above and thus our algorithm is partially kinded. Kinding information is always checked before an attempt at either β, η or c-conversion and also when we want to return a weak head-normal form (whnf) of an arbitrary term. Kinds are not checked when we are dealing with the conversion of two terms and recursively checking their syntactic structure. Any less kinding makes the proof of soundness very difficult and any more kinding makes the proof of completeness more difficult. The proof of completeness is still an open problem

The structure of the algorithm The algorithm is divided mainly into the nine separate functions listed below, given that C is the graph of coercions and Γ is the current context:

- $\texttt{valid}^C(\Gamma)$ is the predicate that checks to see if Γ is a valid context,
- $\texttt{whnf}^C_\Gamma(k)$ computes the β, c-weak head-normal form of k,
- $\texttt{conv}^C_\Gamma(k_1, k_2)$ checks whether k_1 and k_2 are equal objects of some kind K or of **kind** itself.
- $\texttt{kcheck}^C_\Gamma(k, K)$ checks whether k is an object of kind K in Γ.
- $\texttt{kinfer}^C_\Gamma(k)$ computes a kind of k,

– $\texttt{iskind}^C_\Gamma(K)$ returns true iff $\texttt{kinfer}^C_\Gamma(K) \equiv \mathbf{kind}$.
– $\texttt{subkind}^C_\Gamma(K, K')$ checks to see if K is a proper subkind of K' with some coercion c.
– $\texttt{coercion}^C_\Gamma(K, K')$ computes the coercion above coercion c if the subkinding predicate holds. Its behaviour is undefined otherwise.

In the rest of this section we give more detailed description of the main functions used in the algorithm. It is to be understood that for each part of the algorithm the cases are to be tried sequentially until a suitable case is found. If no such case is found, then the algorithm returns either false or failure.

Substitution Substitution is taken as a syntactic operation on terms.

Adding new free variables

1. $$\frac{}{\texttt{valid}^C(\langle\rangle)}$$
2. $$\frac{\texttt{valid}^C(\Gamma)\ \texttt{iskind}^C_\Gamma(A)\ x \notin FV(\Gamma)}{\texttt{valid}^C(\Gamma, x{:}A)}$$

Note that in everything to follow it is assumed that all contexts are valid.

Weak Head-Normal Form

1. $$\frac{\begin{array}{ll}\texttt{whnf}^C_\Gamma(f) \equiv [x{:}A]f' & \texttt{kinfer}^C_\Gamma(f) \equiv (x{:}A)B \\ \texttt{kinfer}^C_\Gamma(k) \equiv K & \texttt{conv}^C_\Gamma(A, K) \\ \texttt{whnf}^C_{\Gamma,x:A}([k/x]f') \equiv k' & \end{array}}{\texttt{whnf}^C_\Gamma(f(k)) \equiv k'}$$
2. $$\frac{\begin{array}{ll}\texttt{kinfer}^C_\Gamma(f) \equiv (x{:}A)B & \texttt{kinfer}^C_\Gamma(k) \equiv K \\ \texttt{subkind}^C_\Gamma(K, A) & \texttt{coercion}^C_\Gamma(K, A) \equiv c \\ \texttt{whnf}^C_\Gamma(f(c(k))) \equiv k' & \end{array}}{\texttt{whnf}^C_\Gamma(f(k)) \equiv k'}$$
3. $$\frac{\texttt{kinfer}^C_\Gamma(k) \equiv K}{\texttt{whnf}^C_\Gamma(k) \equiv k}$$

Note that the above case is only tried if the first two fail. This means that it covers every other possible case such as product kinds and the constant type. The premise of the above case means that only well-typed terms have a whnf. Thus, in particular, **kind** does not have a whnf.

Conversion Note that the conversion relation is defined so that it is obviously symmetric.

1. $$\frac{\texttt{whnf}^C_\Gamma(m) \equiv \mathbf{Type}\ \texttt{whnf}^C_\Gamma(n) \equiv \mathbf{Type}}{\texttt{conv}^C_\Gamma(m, n)}$$

2. $$\frac{\texttt{whnf}^C_\Gamma(m) \equiv x \ \texttt{whnf}^C_\Gamma(n) \equiv x}{\texttt{conv}^C_\Gamma(m,n)}$$

3. $$\frac{\begin{array}{l}\texttt{whnf}^C_\Gamma(m) \equiv El(A) \ \texttt{whnf}^C_\Gamma(n) \equiv El(B) \\ \texttt{conv}^C_\Gamma(A,B)\end{array}}{\texttt{conv}^C_\Gamma(m,n)}$$

4. $$\frac{\begin{array}{ll}\texttt{whnf}^C_\Gamma(m) \equiv (x{:}A)B \ \texttt{whnf}^C_\Gamma(n) \equiv (x{:}A')B' & \\ \texttt{conv}^C_{\Gamma,x:A}(B,B') & \texttt{conv}^C_{\Gamma,x:A'}(B,B') \\ \texttt{conv}^C_\Gamma(A,A') & \end{array}}{\texttt{conv}^C_\Gamma(m,n)}$$

5. $$\frac{\begin{array}{ll}\texttt{whnf}^C_\Gamma(m) \equiv [x{:}A]k \ \texttt{whnf}^C_\Gamma(n) \equiv [x{:}A']k' & \\ \texttt{conv}^C_{\Gamma,x:A}(k,k') & \texttt{conv}^C_{\Gamma,x:A'}(k,k') \\ \texttt{conv}^C_\Gamma(A,A') & \end{array}}{\texttt{conv}^C_\Gamma(m,n)}$$

6. $$\frac{\begin{array}{ll}\texttt{whnf}^C_\Gamma(m) \equiv f(k) & \texttt{whnf}^C_\Gamma(n) \equiv f'(k') \\ \texttt{kinfer}^C_\Gamma(f) \equiv (x{:}A)B & \texttt{kcheck}^C_\Gamma(f',(x{:}A)B) \\ \texttt{kcheck}^C_\Gamma(k,A) & \texttt{kcheck}^C_\Gamma(k',A) \\ \texttt{conv}^C_\Gamma(f,f') & \texttt{conv}^C_\Gamma(k,k')\end{array}}{\texttt{conv}^C_\Gamma(m,n)}$$

7. $$\frac{\begin{array}{ll}\texttt{whnf}^C_\Gamma(m) \equiv [x{:}A]k & \texttt{whnf}^C_\Gamma(n) \equiv n' \\ \texttt{kinfer}^C_\Gamma(m) \equiv (x{:}A)K & \texttt{kcheck}^C_\Gamma(n,(x{:}A)K) \\ \texttt{conv}^C_{\Gamma,x:A}(k,n'(x)) & \end{array}}{\texttt{conv}^C_\Gamma(m,n)}$$

8. $$\frac{\begin{array}{ll}\texttt{whnf}^C_\Gamma(m) \equiv m' & \texttt{whnf}^C_\Gamma(n) \equiv [x{:}A]k \\ \texttt{kinfer}^C_\Gamma(n) \equiv (x{:}A)K & \texttt{kcheck}^C_\Gamma(m,(x{:}A)K) \\ \texttt{conv}^C_{\Gamma,x:A}(k,m'(x)) & \end{array}}{\texttt{conv}^C_\Gamma(m,n)}$$

Kind checking

1. $$\frac{\begin{array}{l}\texttt{kinfer}^C_\Gamma(k) \equiv K' \ \texttt{iskind}^C_\Gamma(K) \\ \texttt{conv}^C_\Gamma(K,K')\end{array}}{\texttt{kcheck}^C_\Gamma(k,K)} \text{ if } K \not\equiv \mathbf{kind},$$

2. $$\frac{\texttt{iskind}^C_\Gamma(K)}{\texttt{kcheck}^C_\Gamma(K,\mathbf{kind})}$$

Kind Inference

1. $$\frac{\texttt{valid}^C(\Gamma)}{\texttt{kinfer}^C_\Gamma(x) \equiv \Gamma(x)}$$

2. $$\frac{\texttt{kinfer}^C_{\Gamma,x:A}(k) \equiv B}{\texttt{kinfer}^C_\Gamma([x{:}A]k) \equiv (x{:}A)B}$$

3. $$\frac{\texttt{iskind}^C_{\Gamma,x:A}(B)}{\texttt{kinfer}^C_\Gamma((x{:}A)B) \equiv \mathbf{kind}}$$

4. $$\frac{\begin{array}{l}\texttt{kinfer}^C_\Gamma(f) \equiv (x{:}A)B \;\; \texttt{kinfer}^C_\Gamma(k) \equiv K \\ \texttt{conv}^C_\Gamma(K, A)\end{array}}{\texttt{kinfer}^C_\Gamma(f(k)) \equiv [k/x]B}$$

5. $$\frac{\begin{array}{ll}\texttt{kinfer}^C_\Gamma(f) \equiv (x{:}A)B & \texttt{kinfer}^C_\Gamma(k) \equiv K \\ \texttt{subkind}^C_\Gamma(K, A) & \texttt{coercion}^C_\Gamma(K, A) \equiv c\end{array}}{\texttt{kinfer}^C_\Gamma(f(k)) \equiv [c(k)/x]B}$$

6. $\texttt{kinfer}^C_\Gamma(\mathbf{Type}) \equiv \mathbf{kind}$

7. $$\frac{\texttt{kinfer}^C_\Gamma(A) \equiv \mathbf{Type}}{\texttt{kinfer}^C_\Gamma(El(A)) \equiv \mathbf{kind}}$$

Checking for Kinds

1. $$\frac{\texttt{kinfer}^C_\Gamma(K) \equiv \mathbf{kind}}{\texttt{iskind}^C_\Gamma(K)}$$

The subkinding relation

1. $$\frac{\begin{array}{ll}\texttt{kcheck}^C_\Gamma(A, \mathbf{Type}) & \texttt{kcheck}^C_\Gamma(B, \mathbf{Type}) \\ \texttt{kcheck}^C_\Gamma(c, (x{:}El(A))El(B)) & (A, c, B) \in C^*\end{array}}{\begin{array}{c}\texttt{subkind}^C_\Gamma(El(A), El(B)) \\ \texttt{coercion}^C_\Gamma(El(A), El(B)) \equiv c\end{array}}$$

2. $$\frac{\begin{array}{ll}\texttt{iskind}^C_{\Gamma,x:K_1}(K_2) & \texttt{subkind}^C_\Gamma(K_1', K_1) \\ \texttt{subkind}^C_{\Gamma,x:K_1}([x/K_2], K_2') & \texttt{coercion}^C_\Gamma(K_1', K_1) \equiv c_1 \\ \texttt{coercion}^C_{\Gamma,x:K_1}(K_2, K_2') \equiv c_2 & \end{array}}{\begin{array}{c}\texttt{subkind}^C_\Gamma((x{:}K_1)K_2, (x{:}K_1')K_2') \\ \texttt{coercion}^C_\Gamma((x{:}K_1)K_2, (x{:}K_1')K_2') \equiv [f{:}(x{:}K_1)K_2][x{:}K_1']c_2(f(c_1(x)))\end{array}}$$

3. $$\frac{\begin{array}{ll}\texttt{iskind}^C_{\Gamma,x:K_1}(K_2) & \texttt{subkind}^C_\Gamma(K_1', K_1) \\ \texttt{conv}^C_{\Gamma,x:K_1}([x/K_2], K_2') & \texttt{coercion}^C_\Gamma(K_1', K_1) \equiv c_1\end{array}}{\begin{array}{c}\texttt{subkind}^C_\Gamma((x{:}K_1)K_2, (x{:}K_1')K_2') \\ \texttt{coercion}^C_\Gamma((x{:}K_1)K_2, (x{:}K_1')K_2') \equiv [f{:}(x{:}K_1)K_2][x{:}K_1']f(c_1(x))\end{array}}$$

4. $$\frac{\begin{array}{ll}\texttt{iskind}^C_\Gamma(K_2) & \texttt{conv}^C_\Gamma(K_1', K_1) \\ \texttt{subkind}^C_{\Gamma,x:K_1}(K_2, K_2') & \texttt{coercion}^C_{\Gamma,x:K_1}(K_2, K_2') \equiv c_2\end{array}}{\begin{array}{c}\texttt{subkind}^C_\Gamma((x{:}K_1)K_2, (x{:}K_1')K_2') \\ \texttt{coercion}^C_\Gamma((x{:}K_1)K_2, (x{:}K_1')K_2') \equiv [f{:}(x{:}K_1)K_2][x{:}K_1']c_2(f(x))\end{array}}$$

4.3 Soundness of the algorithm

The algorithm presented above is sound.

Theorem 31. *The following hold:*

1. *If* $\texttt{valid}^C(\Gamma)$*, then* $\Gamma \vdash \mathbf{valid}$*,*
2. *If* $\Gamma \vdash k{:}K$ *and* $\texttt{whnf}^C_\Gamma(k) \equiv k'$*, then* $\Gamma \vdash k = k'{:}K$*,*
3. *If* $\Gamma \vdash k{:}K$*,* $\Gamma \vdash k'{:}K$ *and* $\texttt{conv}^C_\Gamma(k, k')$*, then* $\Gamma \vdash k = k'{:}K$*,*

4. *If* $\texttt{kcheck}^C_\Gamma(k,K)$*, then* $\Gamma \vdash k{:}K$*,*
5. *If* $\texttt{kinfer}^C_\Gamma(k) \equiv K$*, then* $\Gamma \vdash k{:}K$*,*
6. *If* $\texttt{iskind}^C_\Gamma(K)$*, then* $\Gamma \vdash K{:}\mathbf{kind}$*,*
7. *If* $\texttt{subkind}^C_\Gamma(K,K')$ *and* $\texttt{coercion}^C_\Gamma(K,K') \equiv c$*, then* $\Gamma \vdash K <_c K'$*.*

Proof: The proof is by induction on derivations of the algorithm. Only the more complex cases are described below.

(whnf.1) Assume $\Gamma \vdash f(k){:}K^*$. Also assume:

$$\begin{array}{ll} \texttt{whnf}^C_\Gamma(f) \equiv [x{:}A]f' & \texttt{kinfer}^C_\Gamma(f) \equiv (x{:}A)B \\ \texttt{kinfer}^C_\Gamma(k) \equiv K & \texttt{conv}^C_\Gamma(A,K) \\ \texttt{whnf}^C_\Gamma([k/x]f') \equiv k' & \end{array}$$

By the I.H. we get both $\Gamma \vdash k{:}K$ and $\Gamma \vdash f{:}(x{:}A)B$ and so, by Theorem 13, $\Gamma \vdash (x{:}A)B{:}\mathbf{kind}$ and then, by Lemma 17, $\Gamma \vdash A{:}\mathbf{kind}$. Therefore, we can use the I.H. to get:

$$\Gamma \vdash f = f{:}[x{:}A']f'{:}(x{:}A)B \;\; \Gamma \vdash A = K{:}\mathbf{kind}$$

We can then use the Split Lemma to show $\Gamma \vdash [x{:}A]f'{:}(x{:}A)B$ and so Lemma 18 gives us $\Gamma, x{:}A \vdash f'{:}B$. Then the I.H. gives us:

$$\Gamma, x{:}A \vdash [k/x]f' = k'{:}[k/x]B$$

and the result soon follows using the (β)-rule.

(conv.5) This result follows if we use the admissible rule $\dfrac{\Gamma \vdash K = K'{:}\mathbf{kind} \;\; \Gamma, x{:}K \vdash \mathbf{J}}{\Gamma, x{:}K' \vdash \mathbf{J}}$

(conv.7) This rule follows similarly, using the I.H, the (η)-rule and Theorem 20.

Corollary 32. *The following are also true.*

1. *If* $\texttt{kcheck}^C_\Gamma(k,K)$ *and* $\texttt{whnf}^C_\Gamma(k) \equiv k'$*, then* $\Gamma \vdash k = k'{:}K$*,*
2. *If* $\texttt{kcheck}^C_\Gamma(k,K)$*,* $\texttt{kcheck}^C_\Gamma(k',K)$ *and* $\texttt{conv}^C_\Gamma(k,k')$*, then* $\Gamma \vdash k = k'{:}K$*,*

Proof: Immediate, using Theorem 31.

5 Conclusion and related work

Reported here is the development of some basic but important meta-theoretic properties of the framework with coercive subtyping and a sound type-checking algorithm for coercive subtyping. The presentation of the framework in this paper also allows us to show more clearly what sort of connection there is between subtyping and coherence problems. Such a connection for subtyping in the second order lambda-calculus (system F) was investigated in [LMS95,LMS], where the importance of cut-elimination (ie, elimination of transitivity) is shown to be of particular importance. This is also the case in our work reported here.

Traditional meta-theoretic studies on dependent type theories have mostly been considered for type systems with an underlying conversion relation (e.g., the presentation of Pure Type Systems) rather than those with equality judgements (e.g., the presentation of Martin-Löf's type theory and that used for coercive subtyping in this paper). The relationship between these two kinds of presentations of type theories has been a subtle and difficult problem. Coquand in [Coq91] deals with this problem and Goguen in his PhD thesis [Gog94] has developed a theory of typed operational semantics for the meta-theory of type theory. The approach in this paper is to develop the meta-theory of coercive subtyping by studying the system with judgemental equality directly and try, as far as we can, to separate the meta-theory of the LF with subtyping (at the kind level) from that of the underlying type theory. This is reflected in our treatment and presentation of the meta-theoretic results in this paper. For example, the existence of the coercion insertion map (Theorem 29) is an important factor that allows important computational properties of the extended system to be studied independently from the underlying type theory. In other words, many results can take the form that 'if the underlying type theory T has properties that ..., then T[$\mathcal{C}$] has the property that ...'. The results in this paper has laid down a basis for such a development.

The type-checking algorithm presented here uses the notion of typed reduction, rather than the untyped reduction. We have explained why this is the case by considering an example in Section 4.1. However, this example also suggests that there is a possibility to study more efficient reduction strategies for coercive subtyping. Notice that the following three judgements can be derived:

$$\begin{aligned}
&\Gamma \vdash [x{:}K_1]([y{:}K_2]y)(x){:}(K_1)K_2\\
&\Gamma \vdash [x{:}K_1]x{:}(K_1)K_1\\
&\Gamma \vdash [y{:}K_2]y{:}(K_2)K_2
\end{aligned}$$

Now $\Gamma \vdash (K_1)K_1 <_{c^A} (K_1)K_2$ and $\Gamma \vdash (K_2)K_2 <_{c^B} (K_1)K_2$ where

$$\begin{aligned}
c^A &\equiv [f{:}(x{:}K_1)A][x{:}K_1]c(f(x))\\
c^B &\equiv [f{:}(x{:}K_2)B][x{:}K_1]f(c(x)).
\end{aligned}$$

It can then be shown that $c^A([x{:}K_1]x)$ and $c^B([y{:}K_2]y)$ are convertible. We hope that some generalisation of this example will lead to a strategy that requires a lot less typing. The consideration of the fully typed algorithm in this paper is a necessary step towards further study of, for example, more efficient reduction strategies.

A lot of work has been presented already on subtyping. People such as Cardelli ([Car88,Car89]) and Aspinall and Compagnoni ([AC96]) present systems with subtyping that do not use coercions. Bailey [Bai96] and Saibi [Sai97] have already implemented coercive subtyping into current proof assistant programs as notational abbreviations. Our approach should lead to better understanding of such implementations and possibly better implementation of type-checking algorithms.

References

[AC96] D. Aspinall and A. Compagnoni. Subtyping dependent types. *Proc. of LICS96*, 1996.

[Bai93] Anthony Bailey. Representing algebra in LEGO. Master's thesis, Department of Computer Science, University of Edinburgh, 1993.

[Bai96] A. Bailey. Lego with implicit coercions. 1996. Draft.

[Car88] L. Cardelli. Type-checking dependent types and subtypes. *Lecture Notes in Computer Science*, 306, 1988.

[Car89] L. Cardelli. Typeful programming. Lecture notes for the IFIP State of the Art Seminar on Formal Description of Programming Concepts, Rio de Janeiro, Brazil, 1989.

[Coq91] Th. Coquand. An algorithm for testing conversion in Type Theory. In G. Huet and G. Plotkin, editors, *Logical Frameworks*. Cambridge University Press, 1991.

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

[H$^+$96] G. Huet et al. *The Coq Proof Assistant Reference Manual*. INRIA-Rocquencourt, February 1996.

[Hue89] Gérard Huet. The constructive engine. In R. Narasimhan, editor, *A Perspective in Theoretical Computer Science*. World Scientific Publishing, 1989. Commemorative Volume for Gift Siromoney.

[Jon95] Alex Jones. The formalization of linear algebra in LEGO: The decidable dependency theorem. Master's thesis, Department of Mathematics, University of Manchester, 1995.

[LMS] Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. Coherence and transitivity of subtyping as entailment. *To appear in Journal of Logic and Computation.*

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

[LP92] Zhaohui Luo and Robert Pollack. LEGO proof development system: User's manual. Technical Report LFCS Report ECS-LFCS-92-211, Department of Computer Science, University of Edinburgh, 1992.

[Luo93] Z. Luo. Program specification and data refinement in type theory. *Mathematical Structures in Computer Science*, 3(3), 1993.

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

[Luo96] Z. Luo. Coercive subtyping in type theory. *Proc. of CSL'96, the 1996 Annual Conference of the European Association for Computer Science Logic, Utrecht. LNCS 1258*, 1996.

[Luo97] Z Luo. Coercive suptyping. Draft submitted for publication, 1997.

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

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