

# Coercive Subtyping in Lambda-Free Logical Frameworks

Robin Adams  
robin@cs.rhul.ac.uk

Royal Holloway, University of London

**Abstract.** Coercive subtyping is a powerful approach to subtyping in dependent type theories, but its theoretical properties are often difficult to prove. Lambda-free logical frameworks such as TF have shown themselves to be a powerful tool for investigating the theory of logical frameworks, thanks to the close correspondance between a lambda-free frame and a traditional framework such as LF. We show how a type theory with coercive subtyping may be defined within TF. An operation of typecasting plays the role that coercive application plays in LF. We show that the resulting systems in TF and LF are equivalent, and how several results may be proven more easily in TF and then lifted to LF.

**Keywords:** Coercive subtyping, logical frameworks, lambda-free logical frameworks, coercive application, typecasting

## 1 Introduction

When working with dependent type theories, we often find it convenient to introduce a notion of *subtyping*. Intuitively, to say the type  $A$  is a *subtype* of the type  $B$  is to say that every term of type  $A$  is also a term of type  $B$ . We often find ourselves in a situation where there are two types,  $A$  and  $B$  — such as  $\mathbb{N}$ , the type of natural numbers, and  $\mathbb{Z}$ , the type of integers — such that we would like to work as if  $A$  is a subtype of  $B$ , when it is not literally true that every term of type  $A$  is itself a term of type  $B$ .

One approach to this problem is *coercive subtyping*. We construct a function  $c : (A)B$  — say, the function that maps each natural number  $n$  to the integer  $+n$ . We declare this function to be a *coercion*, which we write as  $A <_c B : \mathbf{Type}$ , indicating that we intend to identify each term  $a : A$  with the term  $ca : B$ . Whenever a context expects a term of type  $B$ , we allow the user to enter a term  $a$  of type  $A$  instead; the machine is to proceed as if the user had entered  $ca$ .

Coercive subtyping has been implemented in proof checkers such as Coq [1], LEGO [2] and Plastic [3], and several large proof developments have made extensive use of coercions [4, 5]. The theory of coercive subtyping within the logical framework LF [6] has been developed in a series of papers by Luo, Luo, Soloviev and the present author [7–10].

Given a type theory with coercive subtyping defined within LF, there are several properties of the system that we may wish to establish, such as *coherence*

(that there is at most one coercion between any two types), *transitivity* (that the subtyping relation is transitive) and *coercion completion* (given any term involving coercive application, we can insert coercions to form a term typable without coercive application). Proving such properties of a system is often difficult; the paper [8], for example, gives the long and intricate proof that coherence implies coercion completion. A method that allows such results to be proven more easily would be very welcome.

*Lambda-free logical frameworks* have shown themselves to be powerful tools for studying the theory of logical frameworks. The term was coined by Luo to describe the framework PAL+ [11] which does not have lambda-abstraction as a primitive, instead forming abstractions via let-definitions. The concept has since been carried further; we shall use the phrase to describe a framework which does not allow abstractions as first-class objects at all. The first such framework was called TF and was constructed by Aczel [12]. The present author [13, 14] has developed the language theory of an infinite family of lambda-free frameworks. They were also employed in the paper [10].

A lambda-free logical framework deals only with objects in normal form; that is,  $\beta$ -normal,  $\eta$ -long form. Lambda-abstractions thus cannot appear as objects in their own right, but only as arguments to a variable or constant. We shall call a logical framework that is not a lambda-free frame a ‘traditional’ logical framework.

Given a traditional logical framework  $F$ , we can often find a lambda-free framework which can be embedded within  $F$  as a conservative subsystem. In fact, TF can be embedded within LF in this manner. That is, there exist translations between TF and LF that map derivable judgements to derivable judgements which are inverses to one another. These allow properties to be ‘lifted’ from TF to LF; that is, the fact that a property holds for LF can be deduced easily from the fact that it holds for TF.

In this paper, we shall show how theories with coercive subtyping may be constructed within TF, may be embedded in the corresponding systems built within LF, and how several results about coercive subtyping in TF can thus be lifted to yield results about LF.

In Section 2 of this paper, we give the necessary background on the framework LF, coercive subtyping in LF, and the lambda-free framework TF. In Section 3, we give the details of the construction in TF of theories with coercive subtyping, and define the translations between the theories in LF and TF. In Section 4, we prove two results in the lambda-free systems — that coherence implies insertion of coercions, and that coherence implies conservativity of equality judgements — and lift these results to the LF systems.

## 2 Background

### 2.1 The Logical Framework LF

The logical framework LF was introduced by Luo in [6]. It is a Church-typed version of Martin-Löf’s logical framework [15], and is intended for use as a meta-

language for specifying type theories. In particular, it contains the ability to declare computation rules in a type theory. It has been implemented in the proof checker Plastic [3].

Its grammar deals with *objects* and *kinds*. The kinds  $K$  in LF are of the following forms: **Type**, whose objects are the types of the type theory being specified;  $\text{El}(A)$ , whose objects are the terms of type  $A$ ; and  $(x : K)K'$ , whose objects are the meta-functions which, given an object  $k$  of kind  $K$ , return an object of kind  $[k/x]K'$ .

An object  $k$  in LF may be a variable, a constant, a lambda-abstraction  $[x : K]k$ , or an application  $k_1 k_2$ .

We refer to a type theory being specified in LF as the *object theory*. An object theory is specified in LF by declaring a number of *constants* and *computation rules*. We may declare a constant  $c$  of kind  $K$ , or a computation rule of the form ' $k = k' : K$  where  $x_1 : K_1, \dots, x_n : K_n$ '.

**Coercive Subtyping in LF** When working within a framework such as LF, we can include coercive subtyping in a theory as follows.

We begin with a type theory  $T$  defined in LF. We extend the grammar to include *subtyping judgements* of the form  $\Gamma \vdash A <_c B : \mathbf{Type}$  where  $A, B$  and  $c$  are objects. The intended meaning for this judgement is that  $A$  is to be made a subtype of  $B$  with  $c$  the *coercion* from  $A$  to  $B$ ; that is, each term  $a : A$  is to be identified with the term  $ca : B$ . We form a set  $\mathcal{R}$  of rules of deduction for deriving *subtyping judgements* of the form  $\Gamma \vdash A <_c B : \mathbf{Type}$ .

We extend the type theory  $T$  with the members of  $\mathcal{R}$ , as well as the *congruence rule*

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

to form the system  $T[\mathcal{R}]_0$ . As each of the new rules of deduction has a subtyping judgement as its conclusion, it is easy to see that  $T[\mathcal{R}]_0$  is a conservative extension of  $T$ .

We extend this system further with a judgement form  $\Gamma \vdash K <_c K'$ , expressing that  $c$  is a coercion from the kind  $K$  to the kind  $K'$ , and rules of deduction governing this judgement form, to form the system  $T[\mathcal{R}]_{0K}$ . Again,  $T[\mathcal{R}]_{0K}$  is a conservative extension of both  $T$  and  $T[\mathcal{R}]_0$ .

We extend this system one more time to form the system  $T[\mathcal{R}]$ , in which, if the judgement  $\Gamma \vdash K <_c K'$  is derivable, then, given any framework function  $\Gamma \vdash f : (x : K')L$  that expects an argument of kind  $K'$ , we can provide an argument  $k$  of kind  $K$  instead, forming the object  $fk$ . This object is to be considered definitionally equal to the object  $f(ck)$ . The operation of forming the object  $fk$  is called *coercive application*.

Having constructed the system  $T[\mathcal{R}]$ , there are certain theoretical properties of the system we may be interested in, including:

**Coherence** We say that  $T[\mathcal{R}]$  is *coherent* if the following all hold:<sup>1</sup>

1. If  $\Gamma \vdash A <_c B : \mathbf{Type}$ , then  $\Gamma \vdash A : \mathbf{Type}$ ,  $\Gamma \vdash B : \mathbf{Type}$ , and  $\Gamma \vdash c : (\text{El}(A))\text{El}(B)$ .
2. The judgement  $\Gamma \vdash A <_c A : \mathbf{Type}$  is never derivable.
3. If  $\Gamma \vdash A <_c B : \mathbf{Type}$  and  $\Gamma \vdash A <_d B : \mathbf{Type}$  then  $\Gamma \vdash c = d : (\text{El}(A))\text{El}(B)$ .

**Transitivity** If  $\Gamma \vdash A <_c B : \mathbf{Type}$  and  $\Gamma \vdash B <_d C : \mathbf{Type}$  then  $\Gamma \vdash A <_{[x:\text{El}(A)]d(cx)} C : \mathbf{Type}$ .

**Coercion Insertion** If  $\Gamma \vdash_{T[\mathcal{R}]} k : K$  and  $\Gamma \vdash_T K$  kind, then there exists  $\bar{k}$  such that  $\Gamma \vdash_T \bar{k} : K$  and  $\Gamma \vdash_{T[\mathcal{R}]} k = \bar{k} : K$ .

## 2.2 Lambda-Free Logical Frameworks

*Lambda-free logical framework* are a powerful tool for studying the theory of logical frameworks. We present here a very abbreviated summary of the definition and properties of TF.<sup>2</sup> Its objects and kinds can be regarded as the objects and kinds of LF that are in  $\beta$ -normal,  $\eta$ -long form. We refer to [13] and the forthcoming [14] for a more detailed presentation.

Its syntax is organised by *arities*  $\alpha$ , which are defined by the grammar  $\alpha ::= (\alpha, \dots, \alpha)$ . We write  $\mathbf{0}$  for the arity  $()$ ,  $\mathbf{1}$  for  $(\mathbf{0})$ ,  $\mathbf{2}$  for  $(\mathbf{0}, \mathbf{0})$ , and so forth. We define *concatenation* on arities by  $(\alpha_1, \dots, \alpha_m) \wedge (\beta_1, \dots, \beta_n) \equiv (\alpha_1, \dots, \alpha_m, \beta_1, \dots, \beta_n)$ . The intuition behind arities is that an  $(\alpha_1, \dots, \alpha_n)$ -ary object is a function that takes  $n$  arguments — namely an  $\alpha_1$ -ary object,  $\dots$ , and an  $\alpha_n$ -ary object — and returns a term or type of the object theory.

Every variable and constant is associated with an arity  $\alpha$ . An *object* must have the form  $z[F_1, \dots, F_n]$ , where  $z$  is a variable or constant, and each  $F_i$  is an *abstraction* of the appropriate arity. An *abstraction* is an expression of the form  $[x_1 : K_1, \dots, x_n : K_n]M$ , where each  $x$  is a variable,  $K$  a kind, and  $M$  is an object. A kind has the form  $(x_1 : K_1, \dots, x_n : K_n)\mathbf{Type}$  or  $(x_1 : K_1, \dots, x_n : K_n)\text{El}(M)$ .

For each variable  $x$  and kind  $K$  of the same arity, we define the abstraction  $x^K$ , the  $\eta$ -long form of  $x$  considered as an abstraction of kind  $K$ .

A *context* in TF is a sequence of declarations  $x : K$ , where  $x$  is a variable and  $K$  a kind of the same arity. The judgements of TF are of three forms:  $\Gamma$  valid,  $\Gamma \vdash M : T$ , and  $\Gamma \vdash M = N : T$ , where  $\Gamma$  is a context,  $M$  and  $N$  objects, and  $T$  a base kind.

The operations of *instantiation* and *employment* play the roles in TF that are played by substitution and application in a traditional framework. We define  $\{F/x\}M$ , the result of *instantiating* the variable  $x$  with the abstraction  $F$ , and

<sup>1</sup> This terminology deviates from that of the previous literature on coercive subtyping, where the set  $\mathcal{R}$  was said to be coherent if  $T[\mathcal{R}]_0$  satisfies the three properties given.

<sup>2</sup> We are describing here the Church-typed version of TF. We can also construct a Curry-typed version of TF, which was the version used in the paper [10]. The two systems are equivalent; this was proven in [13], although mistakes have since been found in the proof given there. A corrected proof shall appear in [14].

$F \bullet G$ , the result of *employing* the abstraction  $F$  on the abstraction  $G$ . These can be thought of as the normal forms of  $[F/x]M$  and  $FG$  respectively, although they can be defined without reference to any notion of reduction.

We introduce *defined judgement forms* to denote an abstraction inhabiting a kind  $\Gamma \Vdash F : K$ ; or two abstractions being equal objects of a kind  $\Gamma \Vdash F = G : K$ . Each of these is defined as a set of judgements of the three primitive forms given above; for example,  $\Gamma \Vdash [x : \text{El}(A)]f[x] : (x : \text{El}(A'))\text{El}(B')$  would be defined to be  $\{( \Gamma \vdash A' = A : \mathbf{Type} ), ( \Gamma, x : \text{El}(A') \vdash f[x] : \text{El}(B') )\}$ .

A type theory is declared in TF in a very similar way to LF; by declaring a number of *constants*  $c : K$  and *computation rules*  $(x_1 : K_1, \dots, x_n : K_N)(M = N : T)$ .

The following properties of TF are provable. For the proofs, we refer to [13].

**Instantiation** If  $\Gamma, x : K, \Gamma' \vdash J$  and  $\Gamma \Vdash F : K$  are derivable, so is

$$\Gamma, \{F/x\}\Gamma' \vdash \{F/x\}J.$$

**Functionality** If  $\Gamma, x : K, \Gamma' \vdash M : T$  and  $\Gamma \Vdash F = G : K$  are derivable, so is

$$\Gamma, \{F/x\}\Gamma' \vdash \{F/x\}M = \{G/x\}M : \{F/x\}T.$$

**Context Conversion** If  $\Gamma, x : K, \Gamma' \vdash J$  and  $\Gamma \Vdash K = K'$  are derivable, so is

$$\Gamma, x : K', \Gamma' \vdash J.$$

**Translations** We now have two type systems, TF and LF, intended for use as logical frameworks. LF can be seen as a conservative extension of TF. We can see TF as picking out the derivable judgements of LF in which everything is in normal form.

To make this precise, we introduce *translations* NF from LF to TF, and lift from TF to LF. The action of NF can be thought of as reducing every object to its normal form. We write out the definitions of this translation NF — in some detail here, as they are different from the details given in [13].<sup>3</sup>

We begin by associating arities with the objects and kinds of LF. Assume that we have declared a type theory in LF, and let  $\Gamma$  be a context. We define a partial function  $\text{Ar}$  which assigns an arity to objects and kinds as follows.

- If  $c$  has been declared with kind  $K$ , then  $\text{Ar}_\Gamma(c) \simeq \text{Ar}_\langle \rangle(K)$ .
- If  $x : K$  is an entry in  $\Gamma$ , say  $\Gamma \equiv \Gamma_1, x : K, \Gamma_2$ , then  $\text{Ar}_\Gamma(x) \simeq \text{Ar}_{\Gamma_1}(K)$ .
- $\text{Ar}_\Gamma([x : K]k) \simeq (\text{Ar}_\Gamma(K)) \hat{\wedge} \text{Ar}_{\Gamma, x:K}(k)$
- If  $\text{Ar}_\Gamma(k) \equiv (\alpha) \hat{\wedge} \beta$ , and  $\text{Ar}_\Gamma(k') \equiv \alpha$ , then  $\text{Ar}_\Gamma(kk') \equiv \beta$ .
- $\text{Ar}_\Gamma(\mathbf{Type}) \equiv \mathbf{0}$ .
- If  $\text{Ar}_\Gamma(k) \equiv \mathbf{0}$ , then  $\text{Ar}_\Gamma(\text{El}(k)) \equiv \mathbf{0}$ .
- $\text{Ar}_\Gamma((x : K)L) \simeq (\text{Ar}_\Gamma(K)) \hat{\wedge} \text{Ar}_{\Gamma, x:K}(L)$ .

We shall call an object or kind of LF  $X$  *well-arityed* with respect to  $T$  and  $\Gamma$  if and only if  $\text{Ar}_\Gamma(X)$  is defined. Note that, if  $X$  is well-arityed, then every constant in  $X$  is declared in  $T$  and every free variable in  $X$  is declared in  $\Gamma$ .

We now define  $\text{NF}_\Gamma(X)$  for each well-arityed  $X$  as follows.

<sup>3</sup> These details were given for the Curry-typed version of TF in [10]; the development for Church-typed TF follows the same lines. The formal details for both systems, and the relations between them, shall be given in [14].

- If  $c$  has been declared with kind  $K$ , then  $\text{NF}_\Gamma(c) \equiv c^K$ .
- If  $x$  has been declared with kind  $K$ , then  $\text{NF}_\Gamma(x) \equiv x^K$ .
- $\text{NF}_\Gamma([x : K]k) \equiv [x : \text{NF}_\Gamma(K)]\text{NF}_{\Gamma, x:K}(k)$ .
- $\text{NF}_\Gamma(kk') \equiv \text{NF}_\Gamma(k) \bullet \text{NF}_\Gamma(k')$ .
- $\text{NF}_\Gamma(\mathbf{Type}) \equiv \mathbf{Type}$
- $\text{NF}_\Gamma(\text{El}(k)) \equiv \text{El}(\text{NF}_\Gamma(k))$
- $\text{NF}_\Gamma((x : K)L) \equiv (x : \text{NF}_\Gamma(K))\text{NF}_{\Gamma, x:K}(L)$

We can define  $\text{NF}(\Gamma)$  and  $\text{NF}(J)$  for  $\Gamma$  a context and  $J$  a judgement of LF in the obvious way.

We shall not write the definition of ‘lift’ here; it consists of mapping objects and abstractions of TF to objects of LF, and kinds of TF to kinds of LF, in the natural way.

These two translations are both sound, and are inverses to one another, as shown by the following theorem:

**Theorem 1** *Let  $T$  be a type theory specified in LF. Let  $\text{NF}(T)$  be the TF specification defined as follows: for every constant declaration  $c : K$  in  $T$ , we declare  $c : \text{NF}_\emptyset(K)$  in  $\text{NF}(T)$ ; and for every computation rule declaration ‘ $k = k' : K$ ’ where  $x_1 : K_1, \dots, x_n : K_n$  in  $T$ , we declare  $(\text{NF}(x_1 : K_1, \dots, x_n : K_n))(\text{NF}_{x::K}(k) = \text{NF}_{x::K}(k')) : \text{NF}_{x::K}(K)$  in  $\text{NF}(T)$ . Then*

1. *If  $J$  is derivable in  $T$ , then  $\text{NF}(J)$  is derivable in  $\text{NF}(T)$ .*
2. *For  $J$  an  $\text{NF}(T)$  judgement,  $J$  is derivable in  $\text{NF}(T)$  if and only if  $\text{lift}(J)$  is derivable in  $T$ .*
3. (a)  $\text{NF}_\Gamma(\text{lift}(M)) \equiv M$   
(b)  $\text{NF}_\Gamma(\text{lift}(K)) \equiv K$
4. (a) *If  $\Gamma \vdash k : K$  is derivable in  $T$ , then so is  $\Gamma \vdash k = \text{lift}(\text{NF}_\Gamma(k)) : K$ .*  
(b) *If  $\Gamma \vdash K$  kind is derivable in  $T$ , then so is  $\Gamma \vdash K = \text{lift}(\text{NF}_\Gamma(K))$ .*

It is thanks in particular to part 2 of this theorem that we can view LF as being a conservative extension of TF.

**Proof** Parts 1 and one direction of part 2 are provable by induction on derivations. Part 3 is proven by simultaneous induction on  $M$  and  $K$ , and part 4 by simultaneous induction on  $k$  and  $K$ . The details are similar to those presented in [10]. The other direction of part 2 then follows: if  $\text{lift}(J)$  is derivable in  $T$ , then  $\text{NF}(\text{lift}(J))$  is derivable in  $\text{NF}(T)$ , and  $J \equiv \text{NF}(\text{lift}(J))$ .  $\square$

### 3 Lambda-Free Frameworks with Coercive Subtyping

In this paper, we shall construct lambda-free systems that are embeddable in the systems  $T[\mathcal{R}]_0$  and  $T[\mathcal{R}]$  associated with coercive subtyping. The situation shall be as portrayed in Fig. 1. It is straightforward to form the system  $\text{NF}(T)[\text{NF}(\mathcal{R})]_0^{\text{TF}}$  that is embeddable in  $T[\mathcal{R}]_0$ , and it turns out to be unnecessary to form a system embeddable in  $T[\mathcal{R}]_{0K}$ .

Forming the final system  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$  is trickier. We observe that coercive application in  $T[\mathcal{R}]$  allows us to perform *typecasting*. If  $k : A$  and  $A < B$ , we

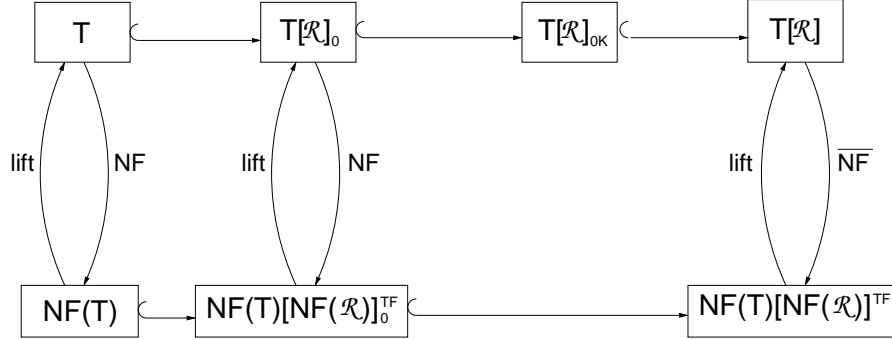


Fig. 1. Systems for Coercive Subtyping

can write down an expression that denotes ‘the object  $k$  considered as an object of type  $B$ ’ or ‘the object  $A$  cast to be of type  $B$ ’:  $([x : B]x)k$ . So our system  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$  must contain some way of performing typecasting. Surprisingly, it turns out that this is enough: if we add typecasting to  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$ , the resulting system is strong enough for us to define sound, mutually inverse translations  $\overline{\text{NF}}$  and  $\text{lift}$  between  $T[\mathcal{R}]$  and  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$ . (The close connection between coercive application and typecasting was first noted by Aczel in a private communication to Luo, reported in [7].)

The translation from  $T[\mathcal{R}]$  to  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$  cannot agree with  $\text{NF}$  on every object, however; for, in the example above,  $\text{NF}([x : B]x)k \equiv \text{NF}(k)$ , yet  $k$  and  $([x : B]x)k$  have different typing properties in  $T[\mathcal{R}]$ . For this reason, we use a new symbol  $\overline{\text{NF}}$  for the new translation. (We shall define it so that  $\overline{\text{NF}}([x : B]x)k$  is the result of typecasting  $\overline{\text{NF}}(k)$  to have type  $\overline{\text{NF}}(B)$ .)

In this section, we shall show how a type theory  $S$  specified in TF may be extended to incorporate coercive subtyping. As in LF, we proceed in stages, first forming the system  $S[\mathcal{Q}]_0^{\text{TF}}$  by including subtyping judgements, then forming  $S[\mathcal{Q}]^{\text{TF}}$  by including typecasting. We can then define the translations between the LF system  $T[\mathcal{R}]$  and the TF system  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$ . The correspondence between the two is not quite as close as between  $T$  and  $\text{NF}(T)$ , but it is still enough for many results to be lifted.

For the rest of this paper, we use  $T$  and  $\mathcal{R}$  for theories and subtyping rules in LF, and  $S$  and  $\mathcal{Q}$  for theories and subtyping rules in TF.

### 3.1 Subtyping Judgements

Let  $S$  be any type theory declared in TF. The construction of the system  $S[\mathcal{Q}]_0^{\text{TF}}$ , which extends  $S$  with subtyping judgements, is almost identical to the corresponding construction in the LF world. The coercions involved shall be 1-ary abstractions. (We choose to use Curry-typed rather than Church-typed abstractions for coercions, as this makes the theory slightly simpler; we could have used either.)

We define a *subtyping judgement* to be an expression of the form  $\Gamma \vdash A <_{[x]M} B : \mathbf{Type}$ , where  $A, B$  and  $M$  are objects, and  $x$  a base variable. This judgement expresses that  $A$  is to be considered as a subtype of  $B$ , with an arbitrary object  $N : \text{El}(A)$  being identified with the object  $\{N/x\}M : \text{El}(B)$ .

Let  $\mathcal{Q}$  be a set of rules of deduction, whose premises may be either judgements of TF or subtyping judgements, and whose conclusions are all subtyping judgements. Then  $S[\mathcal{Q}]_0^{\text{TF}}$  is the result of extending  $S$  with the rules in  $\mathcal{Q}$ , as well as the *congruence rule*:

$$\text{(cong)} \quad \frac{\Gamma \vdash A <_{[x]M} B : \mathbf{Type} \quad \Gamma, x : \text{El}(A) \vdash M = M' : \text{El}(B) \quad \Gamma \vdash A = A' : \mathbf{Type} \quad \Gamma \vdash B = B' : \mathbf{Type}}{\Gamma \vdash A' <_{[x]M'} B' : \mathbf{Type}}$$

The translations NF and lift extend in the obvious manner to the new systems, mapping subtyping judgements to subtyping judgements.

It is easy to see that  $S[\mathcal{Q}]_0^{\text{TF}}$  is a conservative extension of  $S$ , as the conclusion of each of the new rules of deduction is a subtyping judgement.

Let  $T$  be a type theory and  $\mathcal{R}$  a set of subtyping rules in LF. Let us write  $\text{NF}(\mathcal{R})$  to denote the set of rules of deduction formed by applying NF to the judgements in each rule of deduction in  $\mathcal{R}$ . Then all the properties stated in Theorem 1 for  $T$  and  $\text{NF}(T)$  also hold for  $T[\mathcal{R}]_0$  and  $\text{NF}(T)[\text{NF}(\mathcal{R})]_0^{\text{TF}}$ . To prove this, all that is needed is to check that properties 1 and 2 hold for subtyping judgements, which is done by induction on derivations. We should enjoy this tight relationship between the TF and LF worlds for as long as we can; it is the last time that things will be this neat!

### 3.2 Typecasting in a Lambda-Free Framework

Let  $S$  be a type theory specification in TF, and let  $\mathcal{Q}$  be a set of rules of deduction for subtyping judgements as in Section 3.1. The system  $S[\mathcal{Q}]^{\text{TF}}$  is formed by extending  $S[\mathcal{Q}]_0^{\text{TF}}$  as follows.

We add one new constructor to the grammar: if  $M$  and  $A$  are objects, then  $M_A$  is an object. The object  $M_A$  is intended to denote the result of *typecasting*  $M$  to have type  $A$ .

We add the rules of deduction shown in Fig. 2 to the system. The first two allow any object of type  $A$  to be cast to be of type  $B$  whenever  $A < B$ . The second two allow the *trivial* case: any object of type  $A$  may be cast to be of type  $A$ . (The reason for including trivial typecasting shall be revealed below, under ‘Coercive Employment’.)

These additional primitives are enough for us to define a judgement form that corresponds to subkinding, and an operation that corresponds to coercive application, and so to construct translations between  $T[\mathcal{R}]$  and  $\text{NF}(T)[\text{NF}(\mathcal{R})]_0^{\text{TF}}$ .

All the properties of TF listed in Section 2.2 hold for  $T[\mathcal{R}]$  as well; they can be proved by the same method.

$$\begin{array}{c}
\text{(cast)} \frac{\Gamma \vdash M : \text{El}(A) \quad \Gamma \vdash A <_{[x]N} B : \mathbf{Type}}{\Gamma \vdash M_B : \text{El}(B)} \\
\text{(cast\_def)} \frac{\Gamma \vdash M : \text{El}(A) \quad \Gamma \vdash A <_{[x]N} B : \mathbf{Type}}{\Gamma \vdash M_B = \{M/x\}N : \text{El}(B)} \\
\text{(tcast)} \frac{\Gamma \vdash M : \text{El}(A)}{\Gamma \vdash M_A : \text{El}(A)} \quad \text{(tcast\_def)} \frac{\Gamma \vdash M : \text{El}(A)}{\Gamma \vdash M_A = M : \text{El}(A)}
\end{array}$$

**Fig. 2.** Rules of Deduction for Typecasting

**Kindcasting** We extend typecasting to the kinds of higher arity. Given an abstraction  $F$  of kind  $K$ , we shall define the object  $F_{K'}$ , the result of casting  $F$  to have kind  $K'$ .

The formal definition is as follows. Given an abstraction  $F$  and kind  $K$ , both of the same arity  $\alpha$ , we define the  $\alpha$ -ary abstraction  $F_K$  by recursion on  $\alpha$  thus:

$$\begin{aligned}
M_{\text{El}(A)} &\equiv M_A & M_{\mathbf{Type}} &\equiv M \\
([x : K]G)_{(y.K')L'} &\equiv [y : K'](\{(y^{K'})_K/x\}G)_{L'}
\end{aligned}$$

*Example* Suppose we have declared a coercion  $\mathbb{N} <_I \mathbb{Z}$ , and defined a squaring function  $\text{square} : (\mathbb{Z})\mathbb{N}$ . Then we can cast  $\text{square}$  to have kind  $(\mathbb{N})\mathbb{Z}$ ; the result is  $\text{square}_{(\mathbb{N})\mathbb{Z}} \equiv [x : \mathbb{N}](\text{square}[x]_{\mathbb{Z}})_{\mathbb{Z}}$ . That is, the function that takes a natural number, casts it to be an integer, squares that integer, then casts the resulting natural number as an integer.

**Subkinding Judgements** There is no need to add subkinding judgements as a primitive judgement form; we have seen that we can define kindcasting without need such a judgement. In fact, we can use kindcasting to *define* a judgement form for subkinding. We shall say that  $K$  is a subkind of  $K'$  if, given an abstraction  $F$  of kind  $K$ , the attempt to cast  $F$  to be of kind  $K'$  succeeds.

Formally, we define the judgements  $\Gamma \Vdash K \leq K'$  and  $\Gamma \Vdash K \leq_{[x]F} K'$  as follows.

$$\begin{aligned}
(\Gamma \Vdash K \leq K') &\equiv (\Gamma, x : K \Vdash (x^K)_{K'} : K') \\
(\Gamma \Vdash K \leq_{[x]F} K') &\equiv (\Gamma, x : K \Vdash (x^K)_{K'} = F : K')
\end{aligned}$$

The relation produced is reflexive (because of our decision to include trivial typecasting); we can derive  $K \leq K'$  when  $K$  is a subkind of  $K'$  or when  $K = K'$ . This is therefore a weaker relation than the one used in the system  $T[\mathcal{R}]_{0K}$ , but it is adequate for our purposes.

Many of the properties of subtyping and typecasting carry over to subkinding and kindcasting. In particular, the analogues of the rules for typecasting are

admissible:

$$\frac{\Gamma \Vdash F : K \quad \Gamma \Vdash K \leq K'}{\Gamma \Vdash F_{K'} : K'} \quad \frac{\Gamma \Vdash F : K \quad \Gamma \Vdash K \leq_{[x]G} K'}{\Gamma \Vdash F_{K'} = \{F/x\}G : K1}$$

$$\frac{\Gamma \Vdash F : K}{\Gamma \Vdash F_K : K} \quad \frac{\Gamma \Vdash F : K}{\Gamma \Vdash F_K = F : K}$$

The proofs for the first two are straightforward; the last two are proven by induction on the arity of  $F$  and  $K$ .

**Coercive Employment** Kindcasting allows us to define the operation of *coercive employment*, which shall play the role of coercive application. Suppose we wish to employ an abstraction  $F$ , which expects arguments of kind  $K$ , on an abstraction  $G$ , which may be of kind  $K$  or a subkind of  $K$ . We first *cast*  $G$  to be of kind  $K$ , then employ  $F$  on the result. (It is for this reason that we included trivial typecasting: without it, we would need some method of deciding whether or not to use kindcasting on  $G$ .)

Formally, we define the abstraction  $F \diamond G$  by:  $([x : K]F') \diamond G \equiv \{G_K/x\}F'$ .

### 3.3 The Translation $\overline{\text{NF}}$

The defined judgement form for subkinding, and the defined operation of coercive application, allow us now to define the translation  $\overline{\text{NF}}$  from  $T[\mathcal{R}]$  to  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$ . As was discussed above, this translation necessarily disagrees with  $\text{NF}$  on most objects. The principal difference between the definitions of  $\text{NF}$  and  $\overline{\text{NF}}$  is that  $\text{NF}$  maps application to employment, while  $\overline{\text{NF}}$  maps application to coercive employment.

We then define  $\overline{\text{NF}}_\Gamma(k)$ ,  $\overline{\text{NF}}_\Gamma(K)$ ,  $\overline{\text{NF}}_\Gamma(\Gamma)$  and  $\overline{\text{NF}}_\Gamma(J)$  for every well-arity object  $k$ , kind  $K$ , context  $\Gamma$  and judgement  $J$ . For kinds, contexts, and the judgement forms of  $T$ , the definition is exactly as for  $\text{NF}$ . For objects, the only clause that is different is

$$\overline{\text{NF}}_\Gamma(kk') \equiv \overline{\text{NF}}_\Gamma(k) \diamond \overline{\text{NF}}_\Gamma(k') .$$

For the new judgement forms of  $T[\mathcal{R}]$ , we have

$$\overline{\text{NF}}(\Gamma \vdash A <_c B : \mathbf{Type}) = \{\overline{\text{NF}}(\Gamma) \vdash \overline{\text{NF}}_\Gamma(A) <_{\overline{\text{NF}}_{\Gamma, x:A}(cx)} \overline{\text{NF}}_\Gamma(B) : \mathbf{Type}\}$$

$$\overline{\text{NF}}(\Gamma \vdash K <_c K') = \{\overline{\text{NF}}(\Gamma) \Vdash \overline{\text{NF}}_\Gamma(K) <_{\overline{\text{NF}}_{\Gamma, x:K}(cx)} \overline{\text{NF}}_\Gamma(K')\}$$

We cannot yet prove this translation sound, but we can show how it behaves with respect to substitution:

**Lemma 1** *Let  $k$  and  $K$  have the same arity w.r.t.  $\Gamma$ , and let  $k'$  and  $K'$  be well-arity w.r.t.  $\Gamma, x : K$ . Then*

$$\{\overline{\text{NF}}_\Gamma(k)/x\}\overline{\text{NF}}_{\Gamma, x:K}(k') \equiv \overline{\text{NF}}_\Gamma([k/x]k')$$

$$\{\overline{\text{NF}}_\Gamma(k)/x\}\overline{\text{NF}}_{\Gamma, x:K}(K') \equiv \overline{\text{NF}}_\Gamma([k/x]K')$$

**Proof** This is proven by induction on  $k'$  and  $K'$ . We need to prove the following auxiliary facts first:  $\{F/x\}(G_K) \equiv (\{F/x\}G)_{\{F/x\}K}$ , and  $\{F/x\}(G \diamond H) \equiv \{F/x\}G \diamond \{F/x\}H$ .  $\square$

### 3.4 The Translation lift

The translation lift extends naturally from  $T$  to  $T[\mathcal{R}]$ . We have only to decide how to handle the new primitives, subtyping and typecasting. We map subtyping judgements to subtyping judgements and typecasting to coercive application of identity functions:

$$\begin{aligned} \text{lift}(M_A) &\equiv ([x : \text{El}(\text{lift}(A))]x)\text{lift}(M) \\ \text{lift}(\Gamma \vdash A <_{[x]M} B : \mathbf{Type}) &\equiv (\text{lift}(\Gamma) \vdash \text{lift}(A) <_{[x:\text{lift}(A)]\text{lift}(M)} \text{lift}(B) : \mathbf{Type}) \end{aligned}$$

The following lemma shows how lift behaves with respect to substitution:

**Lemma 2** *Let  $F$  be an abstraction and  $x$  a variable of the same arity. For any expression (object, abstraction, kind, context or judgement)  $X$  of  $T[\mathcal{R}]$ ,  $[\text{lift}(F)/x]\text{lift}(X) \rightarrow_{\beta} \text{lift}(\{F/x\}X)$ .*

The proof is by induction on  $X$ .

With this lemma, we can show that lift is sound:

**Theorem 2** *If  $J$  is derivable in  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$ , then  $\text{lift}(J)$  is derivable in  $T[\mathcal{R}]$ .*

**Proof** An induction on derivations, using Lemma 2 and Subject Reduction.  $\square$

It is not true that  $\overline{\text{NF}}$  is a left-inverse to lift up to syntactic identity, but the two translations are inverses of one another up to judgemental equality in each system:

### Theorem 3

1. *If  $\Gamma \vdash k : K$  is derivable in  $T[\mathcal{R}]$ , then so is  $\Gamma \vdash k = \text{lift}(\overline{\text{NF}}_{\Gamma}(k)) : K$ .*
2. *If  $\Gamma \vdash M : T$  is derivable in  $S[\mathcal{Q}]^{\text{TF}}$ , then so is  $\Gamma \vdash M = \overline{\text{NF}}_{\text{lift}(\Gamma)}(\text{lift}(M)) : T$ .*

**Proof** We must prove part 1 simultaneously with the statement: If  $\Gamma \vdash K$  kind is derivable in  $T[\mathcal{R}]$ , then so is  $\Gamma \vdash K = \text{lift}(\overline{\text{NF}}_{\Gamma}(K))$ . The two statements are proven by induction on  $k$  and  $K$ . Part 2 is proven by induction on  $M$ .  $\square$

We have shown that  $\text{NF}$  and  $\overline{\text{NF}}$  do not agree on all the objects typable in  $T[\mathcal{R}]$ ; however, using the previous theorem, we can prove immediately that  $\text{NF}$  and  $\overline{\text{NF}}$  agree (up to judgemental equality in  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$ ) on the objects typable in  $T$ . The soundness of  $\overline{\text{NF}}$  will follow.

**Theorem 4** *If  $\Gamma \vdash_T k : K$ , then  $\overline{\text{NF}}(\Gamma) \Vdash \overline{\text{NF}}_{\Gamma}(k) = \text{NF}_{\Gamma}(k) : \overline{\text{NF}}_{\Gamma}(K)$  is derivable in  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$ .*

**Proof** Suppose  $\Gamma \vdash_T k : K$ . Then  $\Gamma \vdash_T k = \text{lift}(\text{NF}_\Gamma(k)) = \text{lift}(\overline{\text{NF}}_\Gamma(k)) : K$ . Therefore, using the soundness of NF,

$$\text{NF}(\Gamma) \Vdash \text{NF}_\Gamma(\text{lift}(\text{NF}_\Gamma(k))) = \text{NF}_\Gamma(\text{lift}(\overline{\text{NF}}_\Gamma(k))) : \text{NF}_\Gamma(K)$$

is derivable in  $\text{NF}(T)$ , hence in  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$ . Therefore, as NF is a left inverse to lift up to equality (which can be proven in  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$  just as it was in  $\text{NF}(T)$ ), we have

$$\overline{\text{NF}}(\Gamma) \Vdash_{\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}} \text{NF}_\Gamma(k) = \overline{\text{NF}}_\Gamma(k) : \overline{\text{NF}}_\Gamma(K) . \quad \square$$

### Corollary 5

1. For any judgement  $J$  of  $T[\mathcal{R}]$ ,  $\text{NF}(J)$  is derivable in  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$  if and only if  $\overline{\text{NF}}(J)$  is derivable.
2. If  $J$  is derivable in  $T[\mathcal{R}]$ , then  $\overline{\text{NF}}(J)$  is derivable in  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$ .

**Proof** Part 1 is proven by induction on the length of the judgement  $J$ , using Theorem 4. Part 2 is proven by induction on derivations, making use of part 1 in the case of a rule associated with a declaration of  $T$  or a rule in  $\mathcal{R}$ .  $\square$

## 4 Lifting Results

Using these translations, we can “lift” many results from the TF world to the LF world. That is, we can prove properties of the TF systems then, using the properties of the translations that we have established, deduce that the corresponding properties hold of the the LF systems. This is often a profitable approach as many properties are simpler to prove for the TF systems, since they have fewer constructors in their grammar and fewer rules of deduction.

In this section, we shall prove two properties of lambda-free systems, and show how each can be lifted to the LF systems.

### 4.1 Insertion of Coercions

This result says that every typable object in  $S[\mathcal{Q}]^{\text{TF}}$  is equal to some object typable in  $S$ . Thus, we have not introduced any new objects in moving from  $S$  to  $S[\mathcal{Q}]^{\text{TF}}$ . The proof consists of showing we can replace any typecast term  $M_A$  with  $C \bullet M$ , where  $C$  is the appropriate coercion.

**Theorem 6 (Insertion of Coercions)** *Suppose  $S[\mathcal{Q}]^{\text{TF}}$  is coherent. If  $\Gamma \vdash_{S[\mathcal{Q}]^{\text{TF}}} M : T$ , then there exist  $\overline{\Gamma}$ ,  $\overline{M}$  and  $\overline{T}$  such that  $\overline{\Gamma} \vdash_S \overline{M} : \overline{T}$  and*

$$\Vdash_{S[\mathcal{Q}]^{\text{TF}}} \Gamma = \overline{\Gamma} \quad \Gamma \Vdash_{S[\mathcal{Q}]^{\text{TF}}} T = \overline{T} \quad \Gamma \vdash_{S[\mathcal{Q}]^{\text{TF}}} M = \overline{M} : T$$

*Further, if  $M$  does not involve typecasting, then  $\overline{M} \equiv M$ ; similarly for  $\Gamma$  and  $T$ .*

**Proof** Let  $\mathcal{Q}'$  be the following set of rules. For each rule  $R \in \mathcal{Q}$  with conclusion  $\Gamma \vdash A <_{[x]M} B : \mathbf{Type}$ , we put in  $\mathcal{Q}'$  the rule formed from  $R$  by adding the premises  $\Gamma \vdash A : \mathbf{Type}$ ,  $\Gamma \vdash B : \mathbf{Type}$ , and  $\Gamma, x : \text{El}(A) \vdash M : \text{El}(B)$ . As  $S[\mathcal{Q}]^{\text{TF}}$  is coherent, the systems  $S[\mathcal{Q}]^{\text{TF}}$  and  $S[\mathcal{Q}']^{\text{TF}}$  have the same derivable judgements.

We prove the following three statements:

1. Suppose  $\Gamma \vdash_{S[\mathcal{Q}']^{\text{TF}}} M : T$ , and  $\bar{\Gamma} \vdash_S \bar{T}$  kind,  $\Vdash_{S[\mathcal{Q}']^{\text{TF}}} \Gamma = \bar{\Gamma}$ , and  $\Gamma \Vdash_{S[\mathcal{Q}']^{\text{TF}}} T = \bar{T}$ . Then there exists  $\bar{M}$  such that  $\bar{\Gamma} \vdash_S \bar{M} : \bar{T}$  and  $\Gamma \vdash_{S[\mathcal{Q}']^{\text{TF}}} M = \bar{M} : T$ . If  $M$  does not involve typecasting, then  $\bar{M} \equiv M$ .
2. Suppose  $\Gamma \Vdash_{S[\mathcal{Q}']^{\text{TF}}} T$  kind and  $\bar{\Gamma}$  valid in  $S$ . Then there exists  $\bar{T}$  such that  $\Gamma \Vdash_S \bar{T}$  kind and  $\Gamma \Vdash_{S[\mathcal{Q}']^{\text{TF}}} T = \bar{T}$ . If  $T$  does not involve typecasting, then  $\bar{T} \equiv T$ .
3. If  $\Gamma$  valid in  $S[\mathcal{Q}']^{\text{TF}}$ , then there exists  $\bar{\Gamma}$  such that  $\bar{\Gamma}$  valid in  $T$  and  $\Vdash_{S[\mathcal{Q}']^{\text{TF}}} \Gamma = \bar{\Gamma}$ . If  $\Gamma$  does not involve typecasting, then  $\bar{\Gamma} \equiv \Gamma$ .

The first and second are proven simultaneously by induction on derivations. In the case where  $M$  is the result of typecasting using a subtyping judgement  $\Gamma \vdash A <_{[x]N} B : \mathbf{Type}$ , this judgement must have been formed by a member  $R$  of  $\mathcal{Q}'$ , possibly followed by some applications of the congruence rule. We apply the induction hypothesis to the new premises of  $R$ .

The third statement follows easily, and the result follows.  $\square$

**Corollary 7** *Let  $T$  be a type theory specification on LF, and  $\mathcal{R}$  a set of subtyping rules for  $T$  such that  $T[\mathcal{R}]$  and  $\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}$  are coherent. If  $\Gamma \vdash_{T[\mathcal{R}]} k : K$ , then there exist  $\bar{\Gamma}$ ,  $\bar{k}$  and  $\bar{K}$  such that  $\bar{\Gamma} \vdash_T \bar{k} : \bar{K}$ ,  $\Vdash_{T[\mathcal{R}]} \Gamma = \bar{\Gamma}$ ,  $\Gamma \vdash_{T[\mathcal{R}]} K = \bar{K}$ , and  $\Gamma \vdash_{T[\mathcal{R}]} k = \bar{k} : K$ .*

**Proof** Suppose  $\Gamma \vdash_{T[\mathcal{R}]} k : K$ . Then, by the soundness of  $\overline{\text{NF}}$ ,

$$\overline{\text{NF}}(\Gamma) \Vdash_{\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}} \overline{\text{NF}}_{\Gamma}(k) : \overline{\text{NF}}_{\Gamma}(K) .$$

Therefore, by the theorem, there exist  $\Delta$ ,  $F$  and  $L$  such that

$$\begin{array}{ll} \Delta \Vdash F : L & \text{in } \text{NF}(T) \\ \Vdash \overline{\text{NF}}(\Gamma) = \Delta & \text{in } \text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}} \\ \overline{\text{NF}}(\Gamma) \Vdash \overline{\text{NF}}_{\Gamma}(K) = L & \text{in } \text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}} \\ \overline{\text{NF}}(\Gamma) \Vdash \overline{\text{NF}}_{\Gamma}(k) = F : \overline{\text{NF}}_{\Gamma}(K) & \text{in } \text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}} \end{array}$$

Let  $\bar{\Gamma} \equiv \text{lift}(\Delta)$ ,  $\bar{k} \equiv \text{lift}(F)$ , and  $\bar{K} \equiv \text{lift}(L)$ . Then, by the soundness of lift, we have

$$\begin{array}{l} \bar{\Gamma} \vdash_T \bar{k} : \bar{K} \\ \Vdash_{T[\mathcal{R}]} \text{lift}(\overline{\text{NF}}(\Gamma)) = \bar{\Gamma} \\ \text{lift}(\overline{\text{NF}}(\Gamma)) \vdash_{T[\mathcal{R}]} \text{lift}(\overline{\text{NF}}_{\Gamma}(K)) = \bar{K} \\ \text{lift}(\overline{\text{NF}}(\Gamma)) \vdash_{T[\mathcal{R}]} \text{lift}(\overline{\text{NF}}_{\Gamma}(k)) = \bar{k} : \text{lift}(\overline{\text{NF}}_{\Gamma}(K)) \end{array}$$

The desired results follow using the fact that lift is a left inverse to  $\overline{\text{NF}}$ .  $\square$

The pattern of this proof is typical of the method for lifting results from TF to LF.

This result was first proved for LF by Soloviev and Luo (it is an easy consequence of the main theorem in [8]). The proof is long and difficult, and this result is a good example of how the use of lambda-free logical frameworks leads to results being easier to prove. (If we tried to prove Theorem 6 for LF by the same technique, the induction would fail for the cases of the rules governing application and coercive application.)

It follows immediately from Theorem 6 that  $S[\mathcal{Q}]^{\text{TF}}$  is a conservative extension of  $S$ :

**Corollary 8** *If  $J$  is a judgement of  $S$  derivable in  $S[\mathcal{Q}]^{\text{TF}}$ , then  $J$  is derivable in  $S$ .*

This result cannot be lifted directly to the LF systems; in general,  $T[\mathcal{R}]$  is not literally a conservative extension of  $S$ . (If  $\Gamma \vdash_{T[\mathcal{R}]} A < B : \mathbf{Type}$ , then  $\Gamma, y : \text{El}(A) \vdash_{T[\mathcal{R}]} ([x : \text{El}(B)]x)y : \text{El}(B)$ . This judgement fits the syntax of  $S$ , but is not derivable in  $S$ .)

However, a weaker version of this result can be lifted: the equational theory of the objects typable in  $T$  is the same in  $T$  and  $T[\mathcal{R}]$ .

**Corollary 9** *If  $\Gamma \vdash_T k : K$ ,  $\Gamma \vdash_T k' : K$ , and  $\Gamma \vdash_{T[\mathcal{R}]} k = k' : K$ , then  $\Gamma \vdash_T k = k' : K$ .*

**Proof** Suppose the hypotheses hold. By the soundness of  $\overline{\text{NF}}$ ,

$$\overline{\text{NF}}(\Gamma) \vdash_{\text{NF}(T)[\text{NF}(\mathcal{R})]^{\text{TF}}} \overline{\text{NF}}_{\Gamma}(k) = \overline{\text{NF}}_{\Gamma}(k') : \overline{\text{NF}}_{\Gamma}(K) .$$

By the soundness of NF, we have that  $\text{NF}_{\Gamma}(k)$  and  $\text{NF}_{\Gamma}(k')$  are typable in  $\text{NF}(T)$ , hence do not involve typecasting. Therefore, by Corollary 8,

$$\begin{aligned} \text{NF}(\Gamma) \vdash_{\text{NF}(T)} \text{NF}_{\Gamma}(k) &= \text{NF}_{\Gamma}(k') : \text{NF}_{\Gamma}(K) \\ \therefore \Gamma \vdash_T k = \text{lift}(\text{NF}_{\Gamma}(k)) &= \text{lift}(\text{NF}_{\Gamma}(k')) = k' : K . \end{aligned} \quad \square$$

## 5 Conclusion

We have shown how to type theories defined in a lambda-free logical framework may be extended with coercive subtyping. We have further shown how several metatheoretic properties may be proven more easily for such systems, and the results then lifted to systems defined within a traditional logical framework.

There is still work to be done on the theory of coercive subtyping. Several questions remain to be resolved before the theory can be considered wholly satisfactory. As one example, we have seen how important coherence is for the systems  $T[\mathcal{R}]$  and  $S[\mathcal{Q}]^{\text{TF}}$  to enjoy nice metatheoretic properties. What is the relationship between coherence in  $T[\mathcal{R}]_0$  and coherence in  $T[\mathcal{R}]$ ? It is possible to construct counterexamples in which  $T[\mathcal{R}]_0$  is coherent but  $T[\mathcal{R}]$  is not, yet these

all feel artificial in some sense. Are there any restrictions that can be placed on the rules  $\mathcal{R}$  that will ensure coherence in  $T[\mathcal{R}]_0$  entails coherence in  $T[\mathcal{R}]$ ?

The author does not have an answer to this question yet, but feels strongly that lambda-free frameworks may well help to investigate this and similar problems. The work in this paper shows that lambda-free frames are a more flexible tool than may have been suspected. It seems ever more likely that there is a wide variety of features of logical frameworks that lambda-free frames can help investigate.

**Acknowledgements** Many thanks to Zhaohui Luo and Sergei Soloviev for fruitful discussions during the writing of this paper.

## References

1. Barras, B., et al.: The Coq Proof Assistant Reference Manual. (May 2000)
2. Luo, Z., Pollack, R.: LEGO proof development system: User's manual. Technical Report ECS-LFCS-92-211, LFCS, Department of Computer Science, University of Edinburgh (May 1992)
3. Callaghan, P., Luo, Z.: An implementation of LF with coercive subtyping and universes. *Journal of Automated Reasoning* **27**(1) (2001) 3–27
4. Bailey, A.: The Machine-Checked Literate Formalisation of Algebra in Type Theory. PhD thesis, University of Manchester (1998)
5. Geuvers, H., Pollack, R., Wiedijk, F., Zwanenburg, J.: A constructive algebraic hierarchy in Coq. *Journal of Symbolic Computation* **34**(4) (2002) 271–286 Special issue on the integration of automated reasoning and computer algebra systems.
6. Luo, Z.: Computation and Reasoning: A Type Theory for Computer Science. Number 11 in International Series of Monographs on Computer Science. Oxford University Press (1994)
7. Luo, Z.: Coercive subtyping. *Journal of Logic and Computation* **9**(1) (1999) 105–130
8. Soloviev, S., Luo, Z.: Coercion completion and conservativity in coercive subtyping. *Annals of Pure and Applied Logic* **113**(1–3) (2002) 297–322
9. Luo, Z., Luo, Y.: Transitivity in coercive subtyping. *Information and Computation* **197**(1–2) (2005) 122–144
10. Luo, Z., Adams, R.: Structural subtyping for inductive types with functorial equality rules. *Mathematical Structures in Computer Science* (2006) To appear.
11. Luo, Z.: PAL+: A lambda-free logical framework. *Journal of Functional Programming* **13**(2) (2003) 317–338
12. Aczel, P.: Yet another logical framework. Unpublished (2001)
13. Adams, R.: A Modular Hierarchy of Logical Frameworks. PhD thesis, University of Manchester (2004)
14. Adams, R.: Lambda-free logical frameworks. Unpublished. Preprint available at <http://www.cs.rhul.ac.uk/~robin/lfreepdf> (2007)
15. Nordström, B., Petersson, K., Smith, J.: Programming in Martin-Löf's Type Theory: an Introduction. Oxford University Press (1990)