A Calculus of Coercive Subtyping (Extended Abstract)

Libor Škarvada[1], Matej Kollár[1], Ondřej Peterka[2], Ondřej Ryšavý[2], and Dušan Kolář[2]

[1] Masaryk University
{libor,xkollar2}@fi.muni.cz
[2] Brno University of Technology
{ipeterka,rysavy,kolar}@fit.vutbr.cz

August 24, 2009

1 Introduction

Our work that stems, in particular, from the research done by Aspinall and Compagnoni [1], and Luo [5] attempts to provide a framework for systematical studying coercive subtyping in dependent type systems. Contrary to [1] we define subtyping based on coercions instead of allowing term overloading. Contrary to [5] we implemented coercive subtyping as direct extension of λP type system instead of introducing definitional mechanism, which is more powerfull but leads to more complicated presentation of a system.

The calculus defined in this paper is based on λP system and the shape of many type rules and the idea of splitting reduction on term and type level corresponds to $\lambda \mathrm{P}_{\leq}$ system[1]. The introduced notion of coercion functions is motivated by definitions presented in [5]. Therein, a coercion κ is a definable term in the system, and is inserted if necessary to harmonize the type of expressions. It is done via introduction of abbreviations, $f(x) = f(\kappa\ a)$, for f with domain of type A' and coercion $\kappa : A \to A'$ that allows to apply the function to term $a : A$. In our work, we take coercion as definitional term that stands for a function usable for transforming terms between subtypes or computationally equivalent types. In the case of coercion for computational type equality such mapping is simply the identity function that may be removed from the term.

The contribution of the present work is in formalization of an extension of λP calculus with coercive subtyping, in which the subtyping fragment itself is more independent compared to related calculi, such as $\lambda \mathrm{P}_{\leq}$ or $\lambda \Pi$ [2]. In fact, the only dependence comes with the need to deal with type equality, but this is restricted in subtyping fragment. Because of this restriction, by allowing to define subtyping only on types in canonical form it is possible to break the dependence as a whole and show decidability of subtyping fragment independently. Moreover, it can be recognized that the subtyping fragment corresponds to simply typed calculus (considering $\leq$ as a function type constructor).

The present paper is organized as follows: In Section 2, we give a formal presentation of $\lambda \mathrm{P}_{\kappa}$ calculus, which is supplied with examples to help grasp the idea of the system. In Section 3, we state basic properties of the calculus. In particular, we sketch the proof of decidability of typechecking. In Section 4, we conclude with discussion and remarks on the presented system and provide comparison with related work. We also indicate the possible extension of the system and discuss the related issues.

2 The λP_κ calculus

In this section we define the calculus λP_κ, which is an extension of λP calculus with coercive subtyping along the line of Luo as introduced in [5] and [6].

The formal presentation begins with a grammar of *pre-terms* and judgement forms.

Definition 1 (Grammar of pre-terms). *Let x is from an enumerable set of variables, and α is from an enumerable set of type constants, then the pre-terms of the calculus may be constructed according to the following grammar:*

M, N	$::= x \mid \lambda x{:}A.M \mid MN \mid \iota_A$	*terms of the calculus*
A, B	$::= \alpha \mid \pi x{:}A.B \mid \Lambda x{:}A.B \mid A\ M$	*types of the calculus*
K	$::= \star \mid \Pi x{:}A.K$	*kinds of the calculus*

The meaning is usuall:

- A term is a term variable, an abstraction, an application, or a designated constant ι.
- A type is an atomic type, a dependent product type, a type family, or a type application.
- A kind is either the constant $\star$ or a type family. According to this a well-formed kind has always form of $\Pi x_1 : A_1 \ldots x_n : A_n.\star$.

2.1 Type System

The type system incorporates *derivation rules* (typing, subtyping, kinding, and formation rules), cosisting of *judgements*. Judgements contain *contexts*, which are (possibly empty) finite sequences of *declarations*.

Definition 2 (Context Declaration). *For every type constant α, kind K, type A, variable x, and coercion κ, a context declaration has one of the following forms:*

$\alpha : K$	*type constant α is in kind K*
$\kappa : \alpha \leq A$	*type constant α is a subtype of A in kind $\star$ with coercion κ.*
$x : A$	*variable x has type A.*

As we do not employ polymorphism, there are no type variables. Therefore, in declaration $\alpha : K$, the symbol α can denote only a new type constant.

Definition 3 (Context). *Context is a finite sequence of context declarations.*

For the sake of cleanliness, any declaration may appear at most once in a given context. Thus, we rule out contexts like $\langle x{:}A, x{:}B\rangle$, or even $\langle x{:}A, x{:}A\rangle$.

For the presentation of the inference system we adapted Harper's equational formulation of LF [3]. Unlike the original system, we use only equational fragment and define a system of abbreviations that allows us to simulate the rest of the rules. The transformation to the Harper's presentation is easy.

Definition 4 (Judgement Forms). *Let Γ be a context, K be a kind, A and B be types, κ be a coercion, and M be term, then there are three basic forms of judgements:*

$\Gamma \vdash K = K'$	*K and K' are equal kinds in context Γ*
$\Gamma \vdash \kappa : A \leq B$	*A is a subtype of B in context Γ with κ being the witnessing coercion between A and B.*
$\Gamma \vdash M = N : A$	*terms M and N are equal in type A.*

For readability, we use additional judgement forms, which are abbreviations of the basic ones:

$\Gamma \vdash K$	K is a kind, abbreviation for $\Gamma \vdash K = K$
$\Gamma \vdash A = A' : K$	A and A' are equal types in kind K, abbreviation for $\Gamma \vdash \iota_A : A \leq A'$ where ι_A is a special distinguished coercion — identity coercion on A
$\Gamma \vdash A : K$	A is a type of kind K, abbreviation for $\Gamma \vdash A = A : K$
$\Gamma \vdash M : A$	M is a term of type A, abbreviation for $\Gamma \vdash M = M : A$

In judgements of form $\Gamma \vdash \kappa : A \leq B$ we have designated ι symbols that specify the unique identity coercions. If $C : \star$, then ι_C is the identity on C. If $C = (\Lambda x{:}A.B) : (\Pi x{:}A.K)$, then ι_C is a mapping returning a (dependent) coercion on lower type B, $\iota_C\ a = \iota_{B[x:=a]}$. Put loosely, if $C : \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.\star$, then ι_C is n-times lifted identity on a standard type.

For brevity, we allow to write two or more similar judgements in compact form, eg. $\Gamma \vdash M, N : A$ stands for two judgements $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$.

Rules for kind and context formation allow to create a well–defined context. They include the only axiom of the system, (F-EMPTY) rule, which simultaneously states that $\star$ is a well–formed kind and the empty context is a well–formed context.

Definition 5 (Formation Rules).

$$\text{F-EMPTY}\quad \frac{}{\langle\rangle \vdash \star} \qquad \text{F-}\Pi\quad \frac{\Gamma, x{:}A \vdash K = K' \qquad \Gamma \vdash A = A' : \star}{\Gamma \vdash \Pi x{:}A.K = \Pi x{:}A'.K'}$$

$$\text{F-TERM}\quad \frac{\Gamma \vdash A : \star}{\Gamma, x{:}A \vdash \star} \qquad \text{F-TYPE}\quad \frac{\Gamma \vdash K}{\Gamma, \alpha{:}K \vdash \star} \qquad \text{F-SUBT}\quad \frac{\Gamma \vdash A : K}{\Gamma, \kappa{:}\alpha{\leq}A \vdash \star}$$

Similarly to [1], there are two possible ways of introducing type constants to the context. The rule (F-TYPE) allows one to insert to the context a new type constant that inhabits the given kind, e.g. $\Gamma,\ seq : \Pi x{:}nat.\star \vdash \star$. The other way is supposed to be employed if one needs to declare a subtype of an existing type. The rule (F-SUBT) enables declaring a new subtype and its accompanying coercion function, e.g. $\Gamma, n : nat, \kappa : nlist \leq seq\ n \vdash \star$.

Next definition gives a collection of kinding rules. The purpose of these rules is to state the equality on types.

Definition 6 (Kinding Rules).

$$\text{K-VAR}\quad \frac{\Gamma, \alpha{:}K, \Gamma' \vdash \star}{\Gamma, \alpha{:}K, \Gamma' \vdash \alpha : K} \qquad \text{K-CONV}\quad \frac{\Gamma \vdash A = B : K \qquad \Gamma \vdash K = K'}{\Gamma \vdash A = B : K'}$$

$$\text{K-SYM}\quad \frac{\Gamma \vdash A = B : K}{\Gamma \vdash B = A : K} \qquad \text{K-TRANS}\quad \frac{\Gamma \vdash A = B : K \qquad \Gamma \vdash B = C : K}{\Gamma \vdash A = C : K}$$

$$\text{K-}\Lambda\quad \frac{\Gamma, x{:}A_1 \vdash B_1 = B_2 : K \qquad \Gamma \vdash A_1 = A_2 : \star}{\Gamma \vdash \Lambda x{:}A_1.B_1 = \Lambda x{:}A_2.B_2 : \Pi x{:}A_1.K}$$

$$\text{K-APP}\quad \frac{\Gamma \vdash A = A' : \Pi x{:}B'.K \qquad \Gamma \vdash M = M' : B \qquad \Gamma \vdash \kappa : B \leq B'}{\Gamma \vdash A\ M = A'\ M' : K[x := M]}$$

$$\text{K-}\beta\quad \frac{\Gamma, x{:}A \vdash B : K \qquad \Gamma \vdash M : A' \qquad \Gamma \vdash \kappa : A' \leq A}{\Gamma \vdash (\Lambda x{:}A.B)\ M = B[x := M] : K[x := M]}$$

The kind conversion rule (K-CONV) is used to close kinding judgements under a conversion of well-formed kinds. The only general form of Π-kind is $\Pi x_1{:}A_1.\ldots\Pi x_k{:}A_k[x_1,\ldots,x_{k-1}].\star$. Therefore for checking kind equality it is sufficient to check equality of all argument types. This principle is used in rule (F-Π). The rule (K-Λ) is utilized for examining equality of arbitraty type families. The rule (K-APP) serves for the application of a type family to a well-typed term. By requiring a coercion function κ to exist, we allow here the type of the argument to be a subtype of an anticipated type in a type-family argument. The same applies also for (K-β). Rule (K-β) captures the notion of β–equality on the level of types. In this rule it is acceptable to supply a term as a function argument, whose type is not equal to the expected argument type. Instead a subtype is admitted and the coercion function stands for witnessing this subsumption.

The following example demonstrates the application of (K-APP) rule. The example however does not consider subtyping of argument types.

Example 1 (Equivalence of type families). Let us call "tall matrices" those whose width is smaller than their height, $tm = \Lambda h{:}nat \Lambda w{:}less\, h.matrix\, h\, w$. Let Γ be a context $\langle nat : \star,\ tm : \Pi x{:}nat.less\ x \Rightarrow \star,\ m : nat,\ n : less\ m\rangle$. Assume that we derive $m' : nat$, $n' : less\ m$, and also $m = m'$ and $n = n'$ (in the omitted part of the derivation below). Then the rule (K-APP) is used twice:

$$\dfrac{\dfrac{\Gamma \vdash tm : \Pi x{:}nat.less\ x \Rightarrow \star \qquad \dfrac{\vdots}{\Gamma \vdash m = m' : nat}}{\Gamma \vdash tm\ m = tm\ m' : less\ m \Rightarrow \star}\ \text{K-APP} \qquad \dfrac{\vdots}{\Gamma \vdash n = n' : less\ m}}{\Gamma \vdash tm\ m\ n = tm\ m'\ n' : \star}\ \text{K-APP}$$

The subtyping judgements $\Gamma \vdash \kappa : A \leq B$ consist of coercion term κ, which annotates the underlying subtyping relation.

For coercions on standard types we use the following abbreviations:

$\kappa_1 \circ \kappa_2$	for	$\lambda x{:}A.\kappa_1(\kappa_2\, x)$	where $\kappa_1 : A' \leq A''$, $\kappa_2 : A \leq A'$ are coercions
$(\circ\, \kappa)$	for	$\lambda f{:}(\pi y{:}A'.B)\lambda x{:}A.f(\kappa\, x)$	where $\kappa : A \leq A'$ is a coercion
$(\kappa\, \circ)$	for	$\lambda f{:}(\pi y{:}A.B)\lambda x{:}A.\kappa(f\, x)$	where $\kappa : B \leq B'$ is a coercion

Subtyping rules introduce coercion judgements. In these rules, the symbol ι may appear in the position of a coercion function, and indicates that two types are equal. The typing rule (T-ι) defined later enables simplifying coercion terms containing identity coercions.

Definition 7 (Subtyping Rules).

S-VAR
$$\dfrac{\Gamma, \kappa{:}\alpha{\leq}A, \Gamma' \vdash \star}{\Gamma, \kappa{:}\alpha{\leq}A, \Gamma' \vdash \kappa : \alpha \leq A}$$

S-π1
$$\dfrac{\Gamma \vdash \kappa : A \leq A' \qquad \Gamma, x{:}A \vdash B : \star}{\Gamma \vdash (\circ\, \kappa) : (\pi x{:}A'.B) \leq (\pi x{:}A.B)}$$

S-π2
$$\dfrac{\Gamma \vdash B, B' : \star \qquad \Gamma, x{:}A \vdash \kappa : B \leq B'}{\Gamma \vdash (\kappa\, \circ) : (\pi x{:}A.B) \leq (\pi x{:}A.B')}$$

S-APP
$$\dfrac{\Gamma \vdash \kappa : C \leq C' \qquad \Gamma \vdash C, C' : \Pi x{:}A.K \qquad \Gamma \vdash M : A}{\Gamma \vdash \kappa\ M : C\ M \leq C'\ M : K}$$

S-TRANS1
$$\dfrac{\Gamma \vdash B, C, D : \star \qquad \Gamma \vdash \kappa_1 : B \leq C \qquad \Gamma \vdash \kappa_2 : C \leq D}{\Gamma \vdash \kappa_2 \circ \kappa_1 : B \leq D}$$

S-TRANS2
$$\dfrac{\Gamma \vdash B, C, D : \Pi x{:}A.\star \qquad \Gamma \vdash \kappa_1 : B \leq C \qquad \Gamma \vdash \kappa_2 : C \leq D}{\Gamma \vdash (\lambda x{:}A.\kappa_2 x \circ \kappa_1 x) : B \leq D}$$

Rule (S-VAR) allows to use the declaration asserted previously into the context. Rule (S-π1) provides us with a way to subtype function types by assuming contravariant typing of the function argument. Correspondingly, rule (S-π2) expresses that subtyped function types are covariant in their result types.

The following example demonstrates the use of rule (S-π1), which allows subtyping an argument type between (dependent) function types in a contravariant manner.

Example 2 (Contravariant subtyping). Consider a context $\Gamma \equiv \langle nat : \star,\ list : nat \Rightarrow \star,\ \kappa : even \leq nat\rangle$.

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \kappa : even \leq nat}\ \textsc{s-var} \qquad \dfrac{\vdots}{\Gamma, y{:}even \vdash list\ y : \star}\ \textsc{k-app}}{\Gamma \vdash (\circ\,\kappa) : \pi x{:}nat.list\ x \leq \pi y{:}even.list\ y}\ \textsc{s-}\pi 1$$

Transitivity as subsumed by (S-TRANS) rules is necessary, if considering subtyping in several arguments of a function type. It immediatelly involves both (S-π1) and (S-π2) rules as demonstrated in the following example:

Example 3 (Multiple contravariant subtyping). Let us consider context $\Gamma \equiv \langle nat : \star,\ matrix : \Pi h{:}nat.\Pi w{:}nat.\star,\ \kappa : even \leq nat\rangle$. For the sake of space the following abbreviations for type expressions are used:

$$\begin{aligned}
\tau_{nn} &\equiv \pi h{:}nat.\pi w{:}nat.matrix\ h\ w \to nat\\
\tau_{n} &\equiv \pi w{:}nat.matrix\ h\ w \to nat\\
\tau_{e} &\equiv \pi w{:}even.matrix\ h\ w \to nat\\
\tau_{en} &\equiv \pi h{:}even.\pi w{:}nat.matrix\ h\ w \to nat\\
\tau_{ee} &\equiv \pi h{:}even.\pi w{:}even.matrix\ h\ w \to nat
\end{aligned}$$

$$\dfrac{\dfrac{\Gamma \vdash \kappa : even \leq nat}{\Gamma \vdash (\circ\,\kappa) : \tau_{nn} \leq \tau_{en}}\ \textsc{s-}\pi 1 \qquad \dfrac{\dfrac{\Gamma, h{:}even \vdash \kappa : even \leq nat}{\Gamma, h{:}even \vdash (\circ\,\kappa) : \tau_n \leq \tau_e}\ \textsc{s-}\pi 1}{\Gamma \vdash ((\circ\kappa)\circ) : \tau_{en} \leq \tau_{ee}}\ \textsc{s-}\pi 2}{\Gamma \vdash ((\circ\kappa)\circ)\circ(\circ\kappa) : \tau_{nn} \leq \tau_{ee}}\ \textsc{s-trans1}$$

Example 4 (Subtyping of type families). Let us consider context $\Gamma \equiv \langle nat : \star,\ \kappa_{en} : even \leq nat,\ 0_e : even,\ 0_n : nat,\ bag : nat \Rightarrow \star,\ \kappa_{lb} : list \leq bag, \ldots\rangle$ and assume we have derived judgement $\Gamma \vdash 0_n = \kappa_{en}0_e : nat$. Then, using the rule (S-APP), we can derive judgement

$$\Gamma \vdash \kappa_{lb}\ 0_n : list\ 0_n \leq bag\ 0_n : \star$$

Using rule (K-APP) and the fact that $0_n = \kappa_{en}0_e$ we derive $\Gamma \vdash bag\ 0_n = bag\ 0_e : \star$, which is the same as

$$\Gamma \vdash \iota : bag\ 0_n \leq bag\ 0_e : \star$$

(recall that the former is just an abbreviation of the latter). Then using rule (S-TRANS1), we get

$$\Gamma \vdash \iota \circ \kappa_{lb}\ 0_n : list\ 0_n \leq bag\ 0_e$$

Finally, the last definition of the section introduces typing judgements. These rules assert equality on terms under the typing assumptions. This fragment depends on subtyping fragment as we allow the application of a function to the argument whose type is a subtype of the type expected by the function.

Definition 8 (Typing Rules).

$$\textsc{t-var}\ \frac{\Gamma, x{:}A, \Gamma' \vdash \star}{\Gamma, x{:}A, \Gamma' \vdash x : A} \qquad \textsc{t-conv}\ \frac{\Gamma \vdash M_1 = M_2 : A_1 \qquad \Gamma \vdash A_1 = A_2 : \star}{\Gamma \vdash M_1 = M_2 : A_2}$$

$$\textsc{t-sym}\ \frac{\Gamma \vdash M_1{=}M_2 : A}{\Gamma \vdash M_2{=}M_1 : A} \qquad \textsc{t-trans}\ \frac{\Gamma \vdash M_1{=}M_2 : A \qquad \Gamma \vdash M_2{=}M_3 : A}{\Gamma \vdash M_1{=}M_3 : A}$$

$$\textsc{t-}\iota\ \frac{\Gamma \vdash N_1{=}N_2 : A \qquad \Gamma \vdash A : \star}{\Gamma \vdash \iota_A\ N_1{=}N_2 : A}$$

$$\textsc{t-}\beta\ \frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash N : A' \qquad \Gamma \vdash \kappa : A' \leq A}{\Gamma \vdash (\lambda x{:}A.M)\, N{=}M[x := \kappa\ N] : B[x := N]}$$

$$\textsc{t-}\lambda\ \frac{\Gamma, x{:}A_1 \vdash M_1{=}M_2 : B \qquad \Gamma \vdash A_1{=}A_2 : \star}{\Gamma \vdash \lambda x{:}A_1.M_1{=}\lambda x{:}A_2.M_2 : \pi x{:}A_1.B}$$

$$\textsc{t-app}\ \frac{\Gamma \vdash M_1{=}M_2 : \pi x{:}A.B \qquad \Gamma \vdash N_1{=}N_2 : A' \qquad \Gamma \vdash \kappa : A' \leq A}{\Gamma \vdash M_1\, N_1{=}M_2\, N_2 : B[x := N_1]}$$

$$\textsc{t-c}\ \frac{\Gamma \vdash \kappa : C \leq C' \qquad \Gamma \vdash C, C' : \star}{\Gamma \vdash \kappa : C \to C'}$$

Rules (T-VAR), (T-SYM), (T-TRANS), and (T-λ) are standard. Rule (T-ι) defines equality under ι-contraction. It means that ι can be safely removed from the term as it does not represent a significant computational meaning. Rule (T-β) introduces β-reduction into equality judgements. There is, however, a difference from the usual β-reduction. A suitable coercion function is substituted with the argument during redex elimination by β-reduction. It means that the following two terms are equal: $(\lambda x{:}nat.twice\ x)\ e = twice\ (\kappa\ e)$, if $twice : nat \to even$, $e : even$, and $\kappa : even \leq nat$. Note that coercion is not inserted to types in (T-β) nor (T-APP) rules. Rule (T-C) makes posible to transform a coercion to an ordinary term.

The subtyping introduced via (T-APP) rule is shown in the following example.

Example 5 (Coercion in application). Let $\Gamma \equiv \langle nat : \star,\ \kappa : even \leq nat,\ list : nat \Rightarrow \star,\ listMake : \pi x{:}nat.list\ x,\ e : even\rangle$, then:

$$\frac{\Gamma \vdash listMake : \pi x{:}nat.list\ x \qquad \Gamma \vdash e : even \qquad \Gamma \vdash \kappa : even \leq nat : \star}{\Gamma \vdash listMake\ e : list\ e}\ \textsc{t-app}$$

2.2 Typed Reduction

The reduction relation on terms and types can be defined by means of the typing rules (T-ι) and (T-β) (reduction on terms), and the kinding rule (K-β) (reduction on types). This means that reduction requires typing to work and the only correct notion of reduction is *typed reduction*. While the typing rules are the tool of inference the reduction should be understand in the computation sense giving the operational semantics of the calculus.

Definition 9 (Reduction λP^R_κ). *Reduction ($\triangleright$) is defined as usual with respect to the following typed contraction schemes:*

$$(\beta_1) \qquad \frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash N : A' \qquad \Gamma \vdash \kappa : A' \leq A}{\Gamma \vdash^R (\lambda x{:}A.M)N \rightsquigarrow_\beta M[x{:=}\kappa\ N] : B[x{:=}N]}$$

$$(\beta_2) \qquad \frac{\Gamma, x : A \vdash B : K \qquad \Gamma \vdash M : A' \qquad \Gamma \vdash \kappa : A' \leq A}{\Gamma^R \vdash (\Lambda x{:}A.B)N \rightsquigarrow_\beta B[x{:=}N] : K[x{:=}N]}$$

$$(\iota_1) \qquad \frac{\Gamma \vdash A : \star \qquad \Gamma \vdash M : A}{\Gamma \vdash \iota_A\ M \rightsquigarrow_\iota M : A}$$

$$(\iota_2) \qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash A : \star \qquad \Gamma, x : A \vdash B : K}{\Gamma \vdash \iota_{\Lambda x:A.B}\ M \rightsquigarrow_\iota \iota_{B[x:=M]} : K}$$

The following demonstrates the interplay between β and ι reductions.

$$\begin{aligned}
&\Gamma \vdash (\lambda x{:}A\lambda y{:}B\lambda z{:}A \to B \to C.z\ x\ y)M\ N\ (\lambda x{:}A'.\lambda y{:}B.g_{x,y})\\
&\quad \triangleright_\star \kappa_{C' \leq C}((\lambda x{:}A'.\lambda y{:}B'.g_{x,y})(\iota_A M)(\iota_B N)))\\
&\quad \triangleright_\star \kappa_{C' \leq C}((\lambda x{:}A'.\lambda y{:}B.g_{x,y})\ M\ N)\\
&\quad \triangleright_\star \kappa_{C' \leq C} g_{x,y}\ (\kappa_{A \leq A'}\ M)\ N
\end{aligned}$$

Any application of β reduction introduces coercions that stick to the arguments when doping substitution in function body. Using ι reduction the identity coercions can be eliminated from terms therefore we polish the term from the evidently unnecessary applications. Because of existence of ι terms we need corresponding rule to eliminate ι from terms by means of computation. On the other hand, it is possible to check whether two types are equal $A = A'$ or one is a subtype of another $A < A'$ in the common manner by checking $A \leq A'$.

Becase the reduction of terms of the calculus requires the typing context we need to show that the defined typed reduction is sound with typing rules.

Proposition 1. *Let M_1 and M_2 are terms, A, A_1 and A_2 are types, and K is a kind.*

– *If $\Gamma \vdash M_1 \rhd M_2 : A$ then $\Gamma \vdash M_1 = M_2 : A$.*
– *If $\Gamma \vdash A_1 \rhd A_2 : K$ then $\Gamma \vdash A_1 = A_2 : K$.*

Also the desirable property is that the reduction is complete with respect to term and type equality as captured by the type system.

Proposition 2. *Let M_1, M_2 and M' are terms, A, A_1, A_2 and A' are types, and K is a kind.*

– *If $\Gamma \vdash M_1 = M_2 : A$ then there exists M' such that $\Gamma \vdash M_1 \rhd M' : A$ and $\Gamma \vdash M_2 \rhd M' : A$.*
– *If $\Gamma \vdash A_1 = A_2 : K$ then there exists A' such that $\Gamma \vdash A_1 \rhd A' : K$ and $\Gamma \vdash A_2 \rhd A' : K$.*

3 Properties of λP_κ

In [4] we dealt with a restricted version of our calculus we showed that a restricted version of our calculus has standard useful properties like strong normalization, or Church-Rosser property. This is because the only essential difference from Aspinall's calculus $\lambda P_\leq$ is in explicit coercions supplied in judgements, and this additional information does not spoil the properties of the calculus.

We also need to deal with equational judgements which allows us to consider typed reductions on terms and types. Typing information in reductions is necessary to supply right coercions in expressions during the evaluation.

Proposition 3 (Strong Normalization). *If $\Gamma \vdash M : A$ then M is strongly normalizing.*

The important property is the decidability of the calculus. In [4] we showed that if we restrict the subtype relation $\leq$ to types of kind $\star$, the resulting calculus has decidable type derivability.

We expect that the same holds for λP_κ (with the subtype relation extended to all types). The reason for this belief is that (one of) the hard parts of the typing process is the derivation of subtyping judgements—and this is simplified due to the presence of coercion terms. Moreover, the coercion terms can be kept in simple form, thanks to which the equality of coercion terms can be decided.

References

1. D. Aspinall and A. Compagnoni. Subtyping dependent types (summary. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 86–97. IEEE Computer Society Press, 1996.
2. G. Chen. Dependent type system with subtyping. *Journal of Computer Science and Technology*, 13(6), 1998.
3. R. Harper. An equational formulation of lf. Technical Report ECS-LFCS-88-67, University of Edingurgh, 1988.

4. M. Kollar, O. Peterka, O. Rysavy, and D. Kolar. A calculus of coercive subtyping. Technical report, FI MUNI, Brno, to appear Sep2009.
5. Z. Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(1):105–130, 1999.
6. Z. Luo and S. Soloviev. Dependent coercions. In *Proceedings of 8th conference on Category Theory and Computer Science (CTCS'99)*, 1999.