ECOLE NORMALE SUPERIEURE

# Subtyping Parametric and Dependent Types

Gang CHEN
Giuseppe LONGO

LIENS - 96 - 21

Département de Mathématiques et Informatique

CNRS URA 1327

# Subtyping Parametric and Dependent Types

**Gang CHEN**
**Giuseppe LONGO**

**LIENS - 96 - 21**

Décembre 1996

Laboratoire d'Informatique de l'Ecole Normale Supérieure
45 rue d'Ulm 75230 PARIS Cedex 05

Tel : (33)(1) 44 32 00 00

Adresse électronique : ...@dmi.ens.fr

# Subtyping Parametric and Dependent Types
# An introduction*

Gang Chen
DMI
Ecole Normale Supérieure
45 rue d'Ulm
75005 Paris, France
*gang@dmi.ens.fr*

Giuseppe Longo
LIENS(CNRS) and DMI
Ecole Normale Supérieure
45 rue d'Ulm
75005 Paris, France
*longo@dmi.ens.fr*

December 5, 1996

## 1 Introduction

A type may be a subtype of another type. The intuition about this should be clear: a type is a type of data, some data then may live in a given type as well as in a larger one, up to a simple "transformation". The advantage is that those data may be "seen" or used in different contexts. The formal treatment of this intuition, though, is not so obvious, in particular when data may be programs.

In Object Oriented Programming, where the issue of "reusing data" is crucial, there has been a long-lasting discussion on "inheritance" and ... little agreement. There are several ways to understand and formalize inheritance, which depend on the specific programming environment used.

Since early work of Cardelli and Wegner, there has been a large amount of papers developing several possible functional approaches to inheritance, as subtyping. Indeed, functional subtyping captures only one point of view on inheritance, yet this notion largely motivated most of that work. Whether or not inheritance is exactly functional subtyping, this notion raised several problems and many relevant answers (see, for example, [CW85, Mit88, BL90, BTCG+91, CMMS91, CG92, PS, Tiu96, TU96]). We take the most naive approach, here, and survey two recent approaches and some of their consequences, one for the second order types (variable types), the other for types which may depend on first order variables.

*Longo's Invited Lecture at "Type Theory and Term Rewriting", Glasgow, September 1996.

## 2 System Fc

Our starting point is Girard's System F [Gir71]. Its language has two kinds of expression, *types* and *terms*, defined by the following syntax:

$$\begin{array}{lrcl} \text{(Types)} & \sigma & ::= & X \mid \sigma \to \tau \mid \forall X.\sigma \\ \text{(Terms)} & M & ::= & x \mid \lambda x\!:\!\sigma.M \mid MN \mid \lambda X.M \mid M\tau \end{array}$$

We will use:

| | |
|---|---|
| $\sigma, \tau, \rho, \mu$ for types | $M, N, P, Q, R$ for terms |
| $X, Y, Z$ for type variables | $x, y, z$ for term variables |

An *environment* $\Gamma$ is a set of term variables with their types. We write $\Gamma, x\!:\!\sigma$ to extend $\Gamma$ with a new term variable $x$ of type $\sigma$, where $x$ must not already occur in $\Gamma$.

We use the notation $\Gamma \vdash_{Fc} M : \sigma$ for type assignment in system Fc, where term equality is defined below. The following rules define valid type assignments.

**System Fc**

$$\text{(ax)} \quad \Gamma, x\!:\!\sigma \vdash_{Fc} x : \sigma$$

$$(\to \text{ intro}) \quad \frac{\Gamma, x\!:\!\sigma \vdash_{Fc} M : \tau}{\Gamma \vdash_{Fc} \lambda x\!:\!\sigma.M : \sigma \to \tau} \qquad (\to \text{ elim}) \quad \frac{\Gamma \vdash_{Fc} M : \sigma \to \tau \qquad \Gamma \vdash_{Fc} N : \sigma}{\Gamma \vdash_{Fc} MN : \tau}$$

$$(\forall \text{ intro}) \quad \frac{\Gamma \vdash_{Fc} M : \sigma}{\Gamma \vdash_{Fc} \lambda X.M : \forall X.\sigma} \qquad (\forall \text{ elim}) \quad \frac{\Gamma \vdash_{Fc} M : \forall X.\sigma}{\Gamma \vdash_{Fc} M\tau : [\tau/X]\sigma}$$

* *for $X$ not free in the type of any free term variable in $M$*

Reduction of terms is defined as usual by the closure of the following rules, where $FV(M)$ stands for the set of free type and term variables in $M$:

$$(\beta_1) \quad (\lambda x\!:\!\sigma.M)N =_{\beta_1} [N/x]M \qquad (\beta_2) \quad (\lambda X.M)\tau =_{\beta_2} [\tau/X]M$$

$$(\eta_1) \quad \lambda x\!:\!\sigma.Mx =_{\eta_1} M \quad (* \text{ for } x \notin FV(M)) \qquad (\eta_2) \quad \lambda X.MX =_{\eta_2} M \quad (* \text{ for } X \notin FV(M))$$

Equality of terms is the *intended equality*: namely the usual reflexive and transitive closure of the reduction rules above augmented by the following axiom.

$$\text{(Axiom C)}\quad \frac{M : \forall X.\sigma}{M\tau_1 = M\tau_2}$$
$^*$*for $X$ not free in $\sigma$*

It may be fair to call *intended* this extension as it formalizes an early and relevant remark of J.Y. Girard, in [Gir71]: in System F, there are no type discriminators. Indeed, (Axiom C) forces terms of universally quantified type, whose outputs live in the same type, to be constant; that is, it says that a term cannot discriminate between different types, by giving provably different output (different normal forms in the same type). The very sound idea is that in our constructive framework a type represents possibly infinite information. Thus a finite program, a term, cannot output 0 or 1, say, according to a check on the equality of possibly infinite entities. Note also that all known proper models of F realize (Axiom C), that is all known models except the term model and models of the inconsistent theory Type:Type, see [Lon95]. The extended system, call it Fc, has a *decidable* term-equality; indeed, it may be obtained by a normalizing and Church-Rosser reduction relation (ongoing work of Bellé).

# 3 Sets and Propositions

Usually, formalisations are guided by semantic intuition. We then sketch first a path through informal meaning towards a sound theory. Later we will survey the basic ideas of a mathematical model. There exist several alternative interpretations of types, which may suggest different treatment of subtyping. Types may be sets, propositions or objects of categories. We briefly discuss the first two understanding of subtyping and just mention the latter.

## 3.1 Types as Sets or as Objects

As a type is a type of data, not necessarily a structured one, the first idea is to understand subtypes as subsets and extend System Fc by suitable rules, which mimic the intended properties of subsets. This interpretation is naif, indeed impossible, yet it may guide the first few steps towards a formalization.

Viewed as a set, a type is *identically injected* into a larger one. As for universal quantification, the core construction of System F, sets provide an immediate intuition and an easy formalization. The intuitive meaning of $\forall X.\sigma$ is the following: the *intersection* of all the instances of $\sigma$, when X ranges in the intended domain. In this naif interpretation, $\forall X.\sigma$ is always a subset of $\sigma$. Conversely, $\sigma$ may also be a subset, thus identical to $\forall X.\sigma$, in the special case that X does not occur in $\sigma$. More formally,

($\forall$ left, axiom) $\quad \forall X.\sigma \ \leq \ [\rho/X]\sigma$

($\forall$ right, axiom) $\sigma \leq \forall X.\sigma$ for $X$ not free in $\sigma$

In [Mit88] these are two of the formal axioms of the subtyping system. So far so good: sets provide a good intuition and suggest an easy formalization of the crucial inclusion relations, the ones between a type and its universal quantification. This relation is crucial, as the strength of impredicative second order systems is given by the possibility of quantifying over type variables, thus one has to say something on how types and their universal quantifications relate to each other. Note finally that, over sets, the intended inclusion, in each direction, is an injection: indeed, the identical embedding.

The other type constructor of System F(c) is not less relevant: it is the arrow type constructor. Its set-theoretic interpretation poses no problem: (the interpretation of) $\sigma \to \tau$ is the set of functions from (the interpretation of) $\sigma$ to $\tau$. Can we inherit to arrow types the identical injection of a subset into a set? No, there is no way: suppose that $\sigma$ is a subset of $\rho$ then $\sigma \to \tau$ is not a subset, nor a superset of $\rho \to \tau$. In the set-theoretic interpretation of the functions in $\sigma \to \tau$ as graphs, that is as single-valued subsets of the cartesian product $\sigma \times \tau$, the two function sets are simply incomparable. Of course, there are maps going from one arrow set to the other (by extending functions, say), but the identical embedding is lost in either case.

Unfortunately, general Category Theory cannot help either for an interpretation: subobjects nicely generalize subsets, but, similarly as for sets, the monomorphic embedding which characterizes subobjects (see any text in Category Theory, such as [AL91], say) is not inherited at higher types, as a monomorphism is an injection in the specific case of the Category of Sets (a recent, though complex, categorical frame for subtyping has been proposed in [Jac95]). In conclusion, both naif Set and Category Theory cannot suggest any sound formalization of inclusion between arrow types. Moreover, even for universal quantification, the suggestion was very informal, as its meaning is far from granted in either approach, as we will mention later.

### 3.2 Types as Propositions

Even though naif Set Theory suggested a well motivated formalization for subtyping universally quantified types, it did not help us with the arrow types. We are then forced to try another meaning for inclusion and derive a formal treatment from it. The well-known interpretation of types as propositions may turn out to be useful. How can we interpret subtypes as *sub*propositions? By a blend of logical and naif set-theoretic meaning of *inclusion as implication or entailment*. A proposition naively determines a set, its set of validity. Then, when the intended set for $\sigma$ is a subset of the intended one for $\rho$ or $\sigma \leq \rho$, this means that $\sigma$ *implies* $\rho$. Let's then maintain this interpretation and consider inclusion as a special case of logical implication, when types are viewed as propositions, and forget that this actually originated from looking at propositions

as sets. One of the tasks of the formalization will be to determine exactly what kind of implication it is.

We check first inclusion for arrow types, as it caused problems. Since aristotelian logic it is known that if $\rho$ implies $\sigma$ then $\sigma \to \tau$ implies $\rho \to \tau$, if $\to$ is understood as linguistic implication. Assume $\sigma \to \tau$ and that $\rho$ implies $\sigma$, then, given $\rho$, go first to $\sigma$ and then from $\sigma$ to $\tau$, by using $\sigma \to \tau$. Thus, $\sigma \to \tau$ implies $\rho \to \tau$. So far as for the crucial first argument of the $\to$ type constructor. The second argument, here as before, does not pose any problem: if $\rho$ implies $\sigma$, then $\tau \to \rho$ trivially implies $\tau \to \sigma$. In conclusion, once inclusion is understood as implication (or entailement), ancient logic suggests the following rule, *contravariant* in the first argument and *covariant* in the second.

$$(\to) \qquad \frac{\sigma \leq \tau \qquad \rho \leq \rho'}{\tau \to \rho \ \leq \ \sigma \to \rho'}$$

By chance, it happens that this rule has also a clear computational meaning, in the modern terminology, in spite of the lack of set-theoretical and categorical sense. Go back again to the naif interpretation of subtypes as subsets and assume that $\rho$ is "set-theoretically" included in $\sigma$. A program of type $\sigma \to \tau$ can "a fortiori" act on any input of the smaller type $\rho$. Thus, the programs in $\sigma \to \tau$ inhabit also $\rho \to \tau$, or $\sigma \to \tau$ is included in $\rho \to \tau$. The difference with the set-theoretic and the categorical approach should be clear. In this computational perspective, programs are not single-valued set-theoretic relations, nor categorical morphisms with intended source and target, but list of symbols, acting on inputs. The understanding is naif, as we are discussing of *typed* programs and use a *type-free* understanding of their manipulation of inputs. This is fair though, as it corresponds to the practice of many typed programming languages, the ML family for example, where types are used only at compile time and not at run time, when they are actually fed with inputs. This computational intuition motivated the introduction of the $\to$-rule in [BCDC83] and [CW85].

Let's now investigate the other key case: the inclusion relation between a type and its universal quantification. No problem here: in logic, $\forall X.\sigma$ clearly implies $[\rho/X]\sigma$ for any instantiation of $X$ by $\rho$ in $\sigma$. Similarly for the reverse inclusion, when $X$ does not occur in $\sigma$, as quantification in this case is dummy. The point though will be to check whether this implication is the *special kind* of implication that interprets "inclusion" between propositions, some sort of "simple" implication representing the intended identical embedding of sets, within a logic frame. Note that, in order to work out the formal details, Set Theory cannot help, as there is no rigorous set-theoretic interpretation of system F [Rey83]. The miracle happens in a few categories, for example in the original blend of sets and categories provided by the Effective Topos, see [Hyl82], [LM91] and [BL90], an issue to be discussed later.

Before going to the formal description of subtyping as "implication", entailment to be precise, recall though that naif Set Theory suggested an understanding of inclusion as identical embedding, where possible, in particular from $\forall X.\sigma$ to $[\rho/X]\sigma$. When

subtyping is logical implication, do we still have, in general, an injective map which represents the implication from $\forall X.\sigma$ to $[\rho/X]\sigma$? We will discuss this issue in section 6.2. The reader, for the time being, may try to answer the question for exercise ... or, at least, to formalize it in Type Theory (i.e. write the implication which would say that $\lambda x{:}\forall X.\sigma.x\rho : \forall X.\sigma \to [\rho/x]\sigma$ is injective, for any given $\sigma$ and $\rho$).

# 4 Subtyping as Logical Entailment

We are now in the position to summarize in a unified approach the scattered remarks above on the meaning of subtyping. This Gentzen-style presentation of subtyping as entailment is taken from [LMS95] (see [LMS96] for a more recent version).

## 4.1 The Unlabelled System

Here is the core system:

$$\text{(ax)} \quad \sigma \vdash \sigma \qquad\qquad (\to) \quad \frac{\sigma' \vdash \sigma \qquad \tau \vdash \tau'}{\sigma \to \tau \vdash \sigma' \to \tau'}$$

$$(\forall \text{ left}) \quad \frac{[\rho/X]\sigma \vdash \tau}{\forall X.\sigma \vdash \tau} \qquad\qquad (\forall \text{ right}) \quad {}^{*}\textit{for } X \textit{ not free in } \sigma \quad \frac{\sigma \vdash \tau}{\sigma \vdash \forall X.\tau}$$

Note that only one premise is allowed in a sequent $\sigma \vdash \tau$, as even the swapping of inputs is forbidden: this will help us in determining the exact class of maps representing subtyping as constructive implication. Thus, in order to deal with nested implications, we generalize ($\forall$ right) to:

$$(\forall_{n\geq 0} \text{ right}) \quad {}^{*}\textit{for } X \textit{ not free in } \sigma \textit{ nor in } \tau_1, \ldots, \tau_n \quad \frac{\sigma \vdash \tau_1 \to \ldots(\tau_n \to \tau)\ldots)}{\sigma \vdash \tau_1 \to \ldots(\tau_n \to \forall X.\tau)\ldots)}$$

($\forall_n$ right) is a family of rules indexed by $n \geq 0$. Note that, if more than one premise was allowed, ($\forall_n$ right) would be the curried variant of ($\forall$ right) with $n$ premises.

These four rules are, basically, all we need. The reader may wonder what happened to a fundamental property of subtyping, that is, to transitivity. Indeed, in [LMS95] and [LMS96] it is *proven* that $\vdash$ is a partial order, thus, in particular that it is transitive and anti-symmetric. But transitivity is just a (cut) rule:

$$\text{(cut)} \quad \frac{\sigma \vdash \tau \qquad \tau \vdash \rho}{\sigma \vdash \rho}$$

The proof of transitivity is then a proof of admissibility for the rule (cut). That is, each time the premises are derivable, then the consequence is derivable too, see below. Or, equivalently, that the system extended with (cut) has the cut-elimination property. Note that the rules above are all derivable in System F. However, the proof that one can "eliminate cuts", in general, is not inherited to subsystems (a reduction path may lay outside the subsystem). Thus, we cannot use the normalization theorem of System F to prove it (see the full version of [LMS95] for a direct proof and a detailed discussion).

In summary, the entailment system for subtyping is given by the rules:
(ax), ($\rightarrow$), ($\forall$ left), ($\forall_{0\leq n}$ right).

### 4.2 The Labelled System

Recall now that we are in a constructive frame. That is, proofs are not platonic observation of an outside ontology, but actual constructions; in particular, they are programs that transform proofs as inputs to proofs as outputs. This is the essence of Type Theory as formalized Intuitionistic Logic. Let's then write the explicit terms as programs in the entailment rules. We consider terms as "labels" of the deduction rules.

**System $Co^{\vdash}$ (labeled)**

(ax) $\qquad x{:}\sigma \;\vdash_{co}\; x : \sigma$

$$(\rightarrow) \quad \frac{x'{:}\sigma' \;\vdash_{co}\; M : \sigma \qquad y{:}\tau \;\vdash_{co}\; N : \tau'}{x{:}\sigma \rightarrow \tau \;\vdash_{co}\; \lambda x'{:}\sigma'.[xM/y]N \;:\; \sigma' \rightarrow \tau'}$$

$$(\forall \text{ left}) \quad \frac{y{:}[\rho/X]\sigma \;\vdash_{co}\; M : \tau}{x{:}\forall X.\sigma \;\vdash_{co}\; [x\rho/y]M : \tau}$$

For $0 \leq k \leq n$

$$(\forall_{k,n}) \quad \frac{x{:}\sigma \;\vdash_{co}\; \lambda x_1{:}\tau_1 \ldots \lambda x_k{:}\tau_k.M \;:\; (\tau_1 \rightarrow \ldots(\tau_n \rightarrow \tau)\ldots)}{x{:}\sigma \;\vdash_{co}\; \lambda x_1{:}\tau_1 \ldots \lambda x_n{:}\tau_n.\lambda X.M x_{k+1} \ldots x_n \;:\; (\tau_1 \rightarrow \ldots(\tau_n \rightarrow \forall X.\tau)\ldots)}$$

This presentation of the last rule of $Co^{\vdash}$ in [LMS95] is taken from [Tiu96].

We already mentioned that the other approach dealing explicitly with the subtyping relation between a type and its universal quantification has been proposed by Mitchell

in his seminal paper [Mit88]. The latter is an axiomatic approach, based on seven axioms and rules, which include transitivity among the assumption, and it is essentially equivalent to the one above, see [LMS95]. The advantages of a Gentzen style approach should be clear: we may derive transitivity and we can explicitly construct terms, as our deductions are typing rules, in the usual sense. This will also allow to state and prove a new fact: the coherence of the system, in a categorical sense.

# 5 Coherence, Completeness and Transitivity

(The results reviewed in this section are all from [LMS95] and [LMS96], unless otherwise stated). The terms given by the typing system above have several strong restrictions. They are clearly terms of System F, in normal form. They are "linear" in the precise and restrictive sense that each "abstrated" variable, that is each variable in the assumption, appears free exactly once in the consequence. Moreover, each sequent has exactly one assumption. We call *coercions* the typed terms in the systems, with reference to the meaning of this term in the practice of computing.

Are these coercions needed and useful? Note that we are in Type Theory, thus it make no sense to say, when $\sigma$ is different from $\tau$, that "$\sigma$ can be *identically* embedded into $\tau$": there is no identity of type $\sigma \to \tau$, if $\sigma$ is different from $\tau$. Naif Set Theory suggested an axiomatization, but then, in the account above, we had to forget the vague hints coming from it, as Set Theory fails to give (even naively) a sound meaning to contravariance of the arrow type in the first argument. We had then to switch to Types as Propositions, where the meaning of a constructive approach is exactly given by the presence of typed terms. Thus coercions are needed, in Type Theory. (Note that also in categories one needs a non-identical morphism to go from $\sigma$ to $\tau$, if $\sigma$ is different from $\tau$.) The point is to prove how "simple" these maps are.

Recall now that we are thinking to System Fc, namely to system F extended with the constructive equality given by adding (Axiom C). Up to a "subtyping version" of (Axiom C), it is then possible to make some use of coercions as they may be shown to be unique, if any, between two types. Thus, a type is a subtype of another in "at most one way"; this corresponds also to the intuition of computing, as we only want one way to coerce integers, say, into reals.

Let's first state the subtyping version of (Axiom C): following an idea for equality in presence of subtyping in [CMMS91], extend (Axiom C) to the following rule.

(eq appl2 co)
$$\frac{y_1 : [\tau_1/X]\sigma \vdash_{co} N_1 : \mu \qquad y_2 : [\tau_2/X]\sigma \vdash_{co} N_2 : \mu}{x : \forall X.\sigma \vdash_{co} [x\tau_1/y_1]N_1 =_{\eta co} [x\tau_2/y_2]N_2 : \mu}$$

Intuitively, (eq appl2 co) says that if two different instances of the same type are coerced to the same type $\mu$, then these coercions equate the two coerced values, in the

two instances, of each term in $\forall X.\sigma$. (Note that there is an underlining commuting diagram, in the categorical sense of "coherence"). It is easy to observe that, when $[\tau_1/X]\sigma = \mu = [\tau_2/X]\sigma$, and thus when $\sigma$ does not contain $X$ free, (eq appl2 co) collapses to (Axiom C), for coercions.

We can then state the Coherence theorem for coercions.

**Theorem 1 (Coherence of $Co^{\vdash}$ derivations)** *Consider two derivations $x:\sigma \vdash_{co} M:\tau$ and $x:\sigma \vdash_{co} N:\tau$. Then, $x:\sigma \vdash_{co} M =_{\eta co} N$.*

In system F extended with subtyping, coherence has been investigated also in [CG92], from a different perspective though.

One may wonder then whether these coercions, which seem so few, are "sufficiently many", namely whether system $Co^{\vdash}$ is enough expressive as to capture all "intended" subtyping relations. How to formalize this? There are two (equivalent) ways: a syntactic and a semantic one.

As for the syntax, what kind of computations should "all" coercions be? We already said that in Type Theory they cannot be identities. There is no doubt though that coercions should make little or no transformation on terms: at most they should update the type of the inputs and, viewed as type-free computations, they should compute like the identity. Indeed, in [Mit88] they are called "retyping" functions. Here comes, of course, the ML-like practice: at run time, when computing, we only want type-free terms.

It is very easy to show that the typable terms in $Co^{\vdash}$ erase to the identity. Namely, that, when all type information is inductively erased, one gets type-free terms $\eta$-equivalent to the identity. The point is that also the converse holds:

**Theorem 2 (Completeness)** *Let $M$ be a term in $\beta$-nf such that $x:\sigma \vdash_F M:\tau$ and $erase(M) \longrightarrow_\eta x$. Then, $x:\sigma \vdash_{co} M:\tau$.*

As for the semantic completeness, this may be obtained by factorizing via Mitchell's system, [Mit88], where a model-theoretic completeness theorem is given via that provably equivalent system to $Co^{\vdash}$.

Completeness, besides its relevance, has an indirect application: transitivity. We already mentioned that the strength of the Gentzen style approach, as a type assignment system, was going to allow us to *derive* a key property of partial orders, transitivity, instead of assuming it.

Recall that, in the unlabeled system, transitivity may be written as the fact that the rule:

$$(\text{cut}) \quad \frac{\sigma \vdash_{co} \tau \qquad \tau \vdash_{co} \rho}{\sigma \vdash_{co} \rho}$$

is admissible.

At first thinking a labeled version could be the following:

$$\frac{x:\sigma \vdash_{co} M:\tau \qquad y:\tau \vdash_{co} N:\rho}{x:\sigma \vdash_{co} [M/y]N:\rho}$$

But the term $[M/y]N$ may not be a coercion; in particular, it may not be in $\beta$-nf. However, *without* making use of the strong normalization property of System F, it is easy to show that, under the assumptions in the rule, $[M/y]N$ normalizes (just use the strong linearity of coercions: coercions contain one lambda-abstraction for the unique occurrence of the unique free variable ...).

The existence and unicity of the $\beta$-nf of $[M/y]N$ now motivates the following alternative version of the labeled (cut)-rule, where $\text{nf}_\beta([M/y]N)$ means the $\beta$-nf of $[M/y]N$:

$$(\text{cut}) \qquad \frac{x:\sigma \vdash_{co} M:\tau \qquad y:\tau \vdash_{co} N:\rho}{x:\sigma \vdash_{co} \text{nf}_\beta([M/y]N):\rho}$$

The theorem below proves than that $x:\sigma \vdash_{co} \text{nf}_\beta([M/y]N):\rho$ is actually a coercion, that is, derivable in $Co^{\vdash}$; or, in other words, that the rule above is admissible.

**Theorem 3 (Transitivity)** *(cut) is an admissible rule in* $Co^{\vdash}$.

Proof: (Sketch) Just observe that if the assumptions erase to the identity, then also the consequence erases to the identity. By the completeness theorem we are done.

**Definition (Bicoercibility)** *Two types* $\sigma$ *and* $\tau$ *are defined to be bicoercible, written* $\sigma \overset{\cong}{_b} \tau$, *iff* $\sigma \vdash_{co} \tau$ *and* $\tau \vdash_{co} \sigma$.

For example, $\tau \overset{\cong}{_b} \forall X.\tau$ for $X$ not free in $\tau$. Now, recall that, in general, two objects or types $A$ and $B$ are *isomorphic*, $A \cong B$, if there are maps $f:A \to B$ and $g:B \to A$ such that $g \circ f = id$ and $f \circ g = id$. Thus the following can be easily shown:

**Corollary (Anti-symmetry)** *If* $\sigma \overset{\cong}{_b} \tau$ *then* $\sigma \cong \tau$ *in* $Co^{\vdash}$.

Note that bicoercibility is strictly stronger than isomorphism. In [Tiu95] it is shown that bicoercibility is decidable, while subtyping is not (see [TU96] and [Wel95]).

As a final remark, note that it is easy to extend $Co^{\vdash}$with base types. Just extend the language of $Co^{\vdash}$ with fresh type constants $\kappa_1, \kappa_2, \kappa_3, \ldots$. For example, these could be base types such as *bool*, *int*, *real*. To assert that a subtyping relation holds between some of these base types, between $\kappa_i$ and $\kappa_j$ say, add a fresh term constant $c_{i,j}$ to the language and add the following Gentzen-style rule asserting that $\kappa_i$ is a subtype of $\kappa_j$ via coercion $c_{i,j}$:

$$(\kappa_i \leq \kappa_j) \quad \frac{x : \sigma \vdash M : \kappa_i}{x : \sigma \vdash c_{i,j}\, M : \kappa_j}$$

Then $Co^{\vdash}+\rho$ may be shown to be a *conservative* extension of $Co^{\vdash}$, which satisfies (the extended versions of) the Coherence, Completeness and Transitivity Theorems.

# 6 Models and Parametricity

## 6.1 Intersection as Product

In the motivating discussion for the entailment system for subtyping above, we started with the issue of "injectivity" of subtypes into types. Indeed, both the set-theoretic identical embedding and the categorical monomorphism represent "injective" maps. And ... this is the reason why both Set Theory and Category Theory failed to provide the right abstract setting for subtyping, even in the propositional case: both forms of injectivity are lost at arrow types. However, consider just the subtyping relation in ($\forall$ left) axiom or rule. It was an easy remark that, over sets, the naif interpretation of "for-all" as intersection yields injective embeddings: just the identity between an intersection of sets and one component of the intersection.

This interpretation of "for-all" as intersection is not sound, though: it could just informally guide the formalization. The technical reasons are given in [Rey83]. The intuition is that the intersection is "too small" or too much information is lost: from an intersection of sets, one cannot reconstruct the sets which gave it (the components of the intersection). Recall instead the constructive meaning of $\forall X.\sigma$. A proof-term $M : \forall X.\sigma$ is a computation or function that takes *any type* $\rho$ to a proof $M\rho : [\rho/X]\sigma$. Thus, from the terms $M : \forall X.\sigma$ one can reconstruct the terms $M\rho$ in *each* of its components $[\rho/X]\sigma$. Even in set-theoretic terms, the understanding of such an $M$ as a function $f$ and $\sigma$ as a functional $G$ in $X$ may find a better representation as 'indexed product". Let then $G$ be a function(al) from a set $Tp$ of "semantic types" to the category Set. The set-theoretic indexed product is defined as the set or class

$$\prod_{X \in Tp} G(X) = \{\ f \mid f(X) \in G(X),\ \text{for } X \in Tp\}$$

There is a problem though: this is ... "too big". $Tp$ should interpret the set of Types, that is $\forall X.\sigma : Types$ should mean $\prod_{X \in Tp} G(X) \in Tp$, while there is no non-trivial set $Tp$ closed under products indexed over itself.

The solution comes from categorical logic, since the work of Lawvere, where universal quantification is (soundly) interpreted as a dependent or indexed product in suitable Toposes (see [AL91] for details in second order Type Theory, the case we are dealing with). As a matter of fact, in Category Theory, the intended interpretation of "for-all" as product is similar to the set-theoretic one above: categories just add more structure, which reduces the "size" of $\prod_{X \in Tp} G(X)$.

The connection between intersection and products is given in Hyland's Effective Topos, [Hyl82], [Hyl88]. In short, in that specific model, the categorical interpretation of second order universal quantification, as "indexed product" (or adjoint to the diagonal functor), which is formally sound, happens to be (isomorphic to) an intersection, by a non-trivial result, see [LM91]. In a sense, by the validity of a strong proof-principle in that Topos, the Uniformity Principle, the intersection is so informative, or the product is so small, that they can both interpret second order universal quantification. By this, that model is the only known interpretation, up to now, of the subtyping rules for "arrow" and "for all", see [BL90],[CL91],[LMS95]. The point though is that the embedding in ($\forall$ left) axiom or rule, is not interpreted by an identical injection, since we are in a category and there is no identity between different objects. It is even not a monomorphism. Yet it is "computed" by the identical function, a true *coercion* in the sense above, indeed the closest we could go to the set-theoretic intuition. The further advantage is that this interpretation is finally sound and it is inherited at arrow types too.

### 6.2 Effective Parametricity

In the interpretation of Types as Propositions, which motivated our approach to subtyping as entailment, the injectivity of the coercions in the rules left for universal quantification may seem a minor curiosity, in particular because, also when Types are Propositions, injectivity is lost at arrow types. However, it is not so, as that particular injectivity sets an unexpected bridge towards an apparently unrelated topic: Parametricity.

We owe to Strachey, in the sixties, the distinction between "parametric" or uniform and "ad hoc" polymorphism, according to how polymorphic functions depend on their type parameters. Reynolds later proposed a formalization of the early ideas of Strachey: in his (rather complex) approach, parametricity is formalized as a notion of invariance by means of "relations". The idea is that the independence of computations w.r.t. (the internal structure of) types is expressed by their invariance w.r.t. *arbitrary relations* on types. The pivotal result is the "abstraction theorem", which essentially says that a

polymorphic function takes related input types to related output values. This approach has been given a categorical presentation in [MR92] and further developed by many, see for example [ACC93], [Has93].

The first link to our approach is that (Axiom C) can be (easily) *derived* in Reynolds' relational extension of system F, see [ACC93], [Lon95].

The other connection is given by an alternative, but related (by (Axiom C)), approach to parametricity. This is given by the injectivity of the intended coercion in ($\forall$ left). Recall the structure of this coercion:

$$(\forall \text{ left}) \quad \frac{y : [\rho/X]\sigma \ \vdash_{co} \ M : \tau}{x : \forall X.\sigma \ \vdash_{co} \ [x\,\rho/y]M : \tau}$$

Then, for each $\sigma$ and $\rho$, the coercion from $\forall X.\sigma$ to $[\rho/X]\sigma$ is the following map (just take $\tau$ to be $[\rho/X]\sigma$ in the rule):

$$x : \forall X.\sigma \ \vdash_{co} \ x\,\rho : [\rho/X]\sigma$$

Is this injective? And how is this related to parametricity?

The reader was already asked to write, at least, the exact statement of injectivity, for the coercion above. If he/she did so, he/she would now recognize that, by the following theorem, for each $\sigma$ and $\rho$, with no restriction on $\sigma$ nor $\rho$, the coercion above is injective. But, beyond the (minor) point of injectivity, one should understand the theorem for what it really says about terms parametrized by types (or polymorphic).

**Theorem 4 (Genericity)** *Let* $\Gamma \ \vdash_F \ M : \forall X.\sigma$ *and* $\Gamma \ \vdash_F \ N : \forall X.\sigma$. *Then, if* $\Gamma \vdash_{Fc} M\rho = N\rho : [\rho/X]\sigma$ *for some* $\rho$, *one has* $\Gamma \vdash_{Fc} M = N : \forall X.\sigma$.

The proof requires about 15 pages and is given in [LMS93]. It is based in a close proof-theoretic analysis of the derivation of the assumption and in the proof that, in that derivation, types are "generic"; namely, that the "value" of a type appearing in a term is not used in any computation from that term. Note that any term of System F yields a term of universally quantified type and, thus, satisfies an obvious variant of the theorem in terms of substitution.

Is this a form of "parametricity", in the sense of Strachey? Yes, as it says that polymorphic terms have a "uniform behaviour" w.r.t. input types. Indeed, the meaning (and the strength) of the theorem is the following: if two polymorphic functions of the same type agree on one input, then they agree on all inputs. In other words, *any type is generic* (or behaves like a variable). In our understanding, the Genericity Theorem says that we cannot use the possibly infinite information carried by a type, as predicate or (structured) set. This corresponds to the effective nature of Intuitionistic Logic: terms may compute only on finite information. Thus, computations deal with types as "black holes": if two terms compute to the same value on a given black hole, they

will compute in the same way on all other black holes. This suggests that Genericity expresses a form of *Effective* Parametricity. (Or a strong uniformity for polymorphic functions, similar to the one for functions of complex variables: if they are known on the border of a sphere, then they are known everywhere.)

In conclusion, an easy to state injectivity property, which is trivial in Set Theory, false in Category Theory, yields an informative result in Type Theory: the effectiveness and uniformity of the dependence of terms on types. (For a discussion on model theoretic issues, in particular concerning the interpretation of Genericity in the Effective Topos, one should consult [Lon95]; some Proof Theory is discussed in [FL96].)

# 7 Subtyping dependent types (some motivations)

System F extends simply typed $\lambda$-calculus, $\lambda_{\rightarrow}$, by adding variable types and in particular, functions which are mappings from types to terms:

$$\lambda X.M: \quad Types \quad \mapsto \quad Terms$$

On the other hand, dependent type systems have functions which give types as outputs. The first-order dependent type system $\lambda\Pi$ (also called $\lambda P$) is the extension of $\lambda_{\rightarrow}$ by functions which maps terms to types:

$$\Lambda x:\rho.\tau: \quad Terms \quad \mapsto \quad Types$$

A function of this kind is called a type family, or simply, a family, which lives in the "kind" $\Pi x{:}\rho.K$ where $K$ is the kind for $\tau$. There is a special kind $\star$ representing the class of all types. Suppose that a type family $L$ lives in the kind $\Pi x{:}\rho.\star$, then for $y:\rho$, $Ly:\star$ is a type. This shows that a type may contain terms with free variables. So an ordinary $\lambda$-term may have a $\pi$-type instead of an arrow type: $\lambda x{:}\rho.M:\pi x{:}\rho.\tau$. Note also that there are no unique principal types, in general, because of the type conversion

$$(\Lambda x:\rho.\tau)M \quad \rightarrow_{\beta_3} \quad \tau[x:=M]$$

and the fact that $N:\sigma_1 \ \wedge \ \sigma_1 =_\beta \sigma_2 \ \Rightarrow \ N:\sigma_2$, where $\beta = \beta_1 \cup \beta_3$.

Since first order variables have a clear mathematical meaning and use,dependent types have been used as the basis of many proof development systems. Recently, researchers in this area strongly feel the need of combining subtyping with dependent types. Here is an example given by Pfenning [Pfe93] concerning economic encoding of logics. Consider the set (or the type) of well formed formulas in propositional calculus defined by the syntax:

$$\tau \ ::= \ Atom \mid \neg\tau \mid \tau \wedge \tau \mid \tau \vee \tau \mid \tau \Rightarrow \tau \mid$$

and a subset of these formulas defined as:

$$\tau_1 \ ::= \ Atom \mid \neg\tau_1 \mid \tau_1 \vee \tau_1$$

As pointed out by Pfenning, without subtyping, the representation of subsets of formulas like $\tau_1$ is awkward and will lead to inefficient implementations of proof search procedures. To overcome the problem, he proposed the subsorting relation, which could be viewed as a restricted form of subtyping. With subsorting and bounded intersection, the above formula could have a nice declaration as

$$\begin{array}{rll} Atom <: \tau_1 & -- & Atom\ is\ a\ subsort\ of\ the\ type\ formula\ \tau_1 \\ \tau_1 <: \tau & -- & \tau_1\ is\ a\ subsort\ of\ \tau \end{array}$$

$$\begin{array}{lll} \neg & : \tau \to \tau & \neg \ : \tau_1 \to \tau_1 \\ \wedge & : \tau \to \tau \to \tau & \vee \ : \tau_1 \to \tau_1 \to \tau_1 \\ \vee & : \tau \to \tau \to \tau & \\ \Rightarrow & : \tau \to \tau \to \tau & \end{array}$$

where $\tau <: \rho$ denotes that $\tau$ is a subsort of $\rho$. Thus, for example, assume that $v_1, v_2$ are variables of atomic type, then $\neg v_1 \vee v_2$ is in type $\tau_1$, so, by the subsorting relation, it is in the type $\tau$.

There is an implicit use of intersection type in the declaration. The declaration of $\neg$ and $\wedge$ should be transformed to:

$$\begin{array}{lll} \neg & : (\tau \to \tau) \cap (\tau_1 \to \tau_1) \\ \vee & : (\tau \to \tau \to \tau) \cap (\tau_1 \to \tau_1 \to \tau_1) \end{array}$$

Pfenning's study is within the proof environment Elf which is an implementation of Edinburgh LF. Other groups working on proof systems, based on dependent type theory, also found the need of using subtyping. The motivating examples are similar. An early work can be found in [Coq92] in the ALT group. The groups LEGO, Coq and Nuprl are studying implementations of abstract algebra. All of them have proposed extensions of type systems by some sort of subtyping: ZhaoHui Luo [Luo96] has studied the Coercive subtyping extension to LEGO; Jason Hickey in the Nuprl group [Hic95] has combined object-calculus and dependent types and proposed a form of subtyping based on the inheritance mechanism of objects; Courant [Cou95] in the Coq group is working on an extension of Calculus of Construction by subtyping: $CC_{\leq}$. The strong interests in this area are mainly due to the fact that proof development systems are attacking the problem of scale. As said by ZhaoHui Luo[Luo96]: "The lack of useful subtyping mechanisms in dependent type theories with inductive types and the associated proof development systems is one of the obstacles in their applications to large-scale formal development."

# 8 Subtyping in $\lambda P_{\leq}$

Aspinall and Compagnoni [AC96] have studied $\lambda P_{\leq}$ as a subtyping extension to the pure dependent type system $\lambda\Pi$. Its subtyping system is as follows:

$$\text{S-CONV}\quad \frac{\Gamma \vdash \tau, \rho : K \quad \tau =_\beta \rho}{\Gamma \vdash \tau \leq \rho} \qquad \text{S-}\pi\quad \frac{\Gamma \vdash \tau' \leq \tau, \quad \Gamma, x : \tau' \vdash \rho \leq \rho' \quad \Gamma \vdash \pi x{:}\tau.\rho : \star}{\Gamma \vdash \pi x{:}\tau.\rho \leq \pi x{:}\tau'.\rho'}$$

$$\text{S-}\Lambda\quad \frac{\Gamma, x : \tau \vdash \rho \leq \rho' \quad \Gamma \vdash \Lambda x{:}\tau.\rho : K}{\Gamma \vdash \Lambda x{:}\tau.\rho \leq \Lambda x{:}\tau.\rho'} \qquad \text{S-TRANS}\quad \frac{\Gamma \vdash \tau \leq \rho \quad \Gamma \vdash \rho \leq \sigma}{\Gamma \vdash \tau \leq \sigma}$$

$$\text{S-VAR}\quad \frac{\Gamma \vdash \star \quad \alpha \leq \Gamma(\alpha) : K \ \mathit{in}\ \Gamma}{\Gamma \vdash \alpha \leq \Gamma(\alpha)} \qquad \text{S-APP}\quad \frac{\Gamma \vdash \tau \leq \rho \quad \Gamma \vdash \rho M : K}{\Gamma \vdash \tau M \leq \rho M}$$

Compared to the case of second order polymorphism, there are several new features in this formulation of subtyping relation:

**Kinding condition** Kinding requirements, such as $\Gamma \vdash \tau, \rho : K$ and $\Gamma \vdash \pi x{:}\tau.\rho : \star$, are used to make sure that the types and type families involved in subtyping are well-formed. Without them, we may have undesirable subtyping judgements, e.g. $(\pi x{:}\tau.\rho)M \leq (\pi x{:}\tau'.\rho')M$, where the expressions at two sides of $\leq$ are not well-formed since $\pi$-types are not type families and they can not be applied to terms.

The kinding premises in the subtyping rules complicates the study of the system because of the dependencies between typing, kinding, context formation and subtyping.

**Conversion rule** The analysis of $\lambda P_{\leq}$ is challenging, principally because it introduces the conversion rule S-CONV guaranteeing that $\beta$-convertible types occupy the same equivalent class in the subtype relation, and the rule S-APP for subtyping family applications. These rules are responsible for the failure of cut-elimination at type level, that is, for derivations with cut that can not be transformed to cut-free derivations, as it can be seen from the following example of cut application:

$$\dfrac{\dfrac{\sigma =_\beta \pi x{:}\tau.\rho}{\Gamma \vdash \sigma \leq \pi x{:}\tau.\rho}\ \text{S-CONV} \quad \dfrac{\Gamma \vdash \tau' \leq \tau \quad \Gamma, x : \tau' \vdash \rho \leq \rho'}{\Gamma \vdash \pi x{:}\tau.\rho \leq \pi x{:}\tau'.\rho'}\ \text{S-}\pi}{\Gamma \vdash \sigma \leq \pi x{:}\tau'.\rho'}\ \text{S-TRANS}$$

**Type family** Type families are objects living in kinds of the form: $\Pi x_1{:}\tau_1 ... \Pi x_n{:}\tau_n.\star$ A type family is either a type variable, or an $\Lambda$-abstraction of the form $\Lambda x{:}\tau.\rho$, or an application $\tau M$. Strictly speaking, if $\tau, \rho$ are type families of the same kind, the relation $\tau \leq \rho$ is not a subtyping relation since $\tau, \rho$ are not types (although, for the convenience, we still use the terminology of

subtyping). They are functionals, or type constructors mapping a number of terms to a type:

$$TypeFamily: \quad Term_1 \times ... \times Term_n \mapsto Type$$

From logic point of view, $\tau, \rho$ of the above kind are first-order predicates with $n$ parameters. Subtyping between $\tau, \rho$ are actually an extension of logical implication to the relation between n-ary predicates in the sense that : " for all values $M_1, ..., M_n$ such that $\tau(M_1, ..., M_n)$ implies $\rho(M_1, ..., M_n)$, then $\tau \leq \rho$". As $\tau, \rho$ are functionals, they may take the form of $\Lambda$-abstractions:

$$\Lambda x_1{:}\tau_1 ... \Lambda x_n{:}\tau_n.\rho$$

Therefore we can have a logic understanding of the subtyping between family abstractions and between applications of type families:

$$\frac{\Gamma \vdash \tau' \leq \tau, \qquad \Gamma, x:\tau \vdash \rho \leq \rho'}{\Gamma \vdash \Lambda x{:}\tau.\rho \leq \Lambda x{:}\tau'.\rho'} \qquad \frac{\Gamma \vdash \tau \leq \rho}{\Gamma \vdash \tau M \leq \rho M}$$

Subtyping relation defined in $\lambda P_{\leq}$ (or in $\lambda\Pi_{\leq}$) is actually weaker than this general form by the restriction that $\tau' = \tau$. This is because the proof techniques used in these work have not achieved such a generality.

$\lambda P_{\leq}$ has the desired meta-theoretic properties: Confluence, Strong normalization, Subject reduction and Decidability of subtyping. One of the main technical contribution of $\lambda P_{\leq}$ is its algorithmic subtyping system defined on $\beta_3$-normalized types. Reflexivity and transitivity are all admissible in this system. The algorithmic system is the key step towards the proofs of subject reduction and the decidability.

# 9 Subtyping in $\lambda\Pi_{\leq}$

The study of $\lambda\Pi_{\leq}$[Che96] begins with two motivations:
1. could kinding premises be removed from the subtyping rules?
2. is there a subtyping system with cut-elimination at the type level?

$\lambda P_{\leq}$ is a pure dependent type system with subtyping. Proofs in $\lambda P_{\leq}$ are quite delicate due to the circularity of the subtyping and typing system. In practical application, one will need to add other type constructions such as inductive types, intersection types, overloaded types etc. A system with complicated proofs will be difficult to extend. Without kinding premises, then subtyping rules are more "independent", allowing a relatively easy study.

Cut-elimination is an essential property in the study of subtyping. Proofs of subject reduction and decidability all depends on this property. It would be desirable to lift the cut-elimination to the type level instead of being restricted to normalized expressions.

$\lambda\Pi_{\leq}$ has a set of subtyping rules which can be seen as a non-trivial modification to the algorithmic subtyping rules of $\lambda P_{\leq}$. The subtyping rules are:

$$\text{S-}\pi \quad \frac{\Gamma \vdash \tau' \leq \tau \quad \Gamma, x:\tau' \vdash \rho \leq \rho'}{\Gamma \vdash \pi x{:}\tau.\rho \leq \pi x{:}\tau'.\rho'} \qquad \text{S-}\Lambda \quad \frac{\Gamma, x:\tau \vdash \rho \leq \rho'}{\Gamma \vdash \Lambda x{:}\tau.\rho \leq \Lambda x{:}\tau.\rho'}$$

$$\text{S-ApR} \quad \frac{M_1 =_\beta M_1' \ \cdots \ M_n =_\beta M_n'}{\Gamma \vdash \alpha M_1..M_n \leq \alpha M_1'..M_n'} \qquad \text{S-ApT} \quad \frac{\Gamma \vdash \Gamma(\alpha) M_1..M_n \leq \tau}{\Gamma \vdash \alpha M_1..M_n \leq \tau}$$

$$\text{S-ApSL} \quad \frac{\Gamma \vdash [M_1/x]\rho M_2..M_n \leq \sigma}{\Gamma \vdash (\Lambda x{:}\tau.\rho) M_1..M_n \leq \sigma} \qquad \text{S-ApSR} \quad \frac{\Gamma \vdash \sigma \leq [M_1/x]\rho M_2..M_n}{\Gamma \vdash \sigma \leq (\Lambda x{:}\tau.\rho) M_1..M_n}$$

This system is equivalent to $\lambda P_{\leq}$ in the following sense, where $\vdash_{\lambda\Pi_{\leq}}, \vdash_{\lambda P_{\leq}}, \vdash_{\lambda P_{\leq}^{alg}}$ denote judgements in $\lambda\Pi_{\leq}$, $\lambda P_{\leq}$ and the algorithmic version of $\lambda P_{\leq}$ respectively:

$$\Gamma \vdash_{\lambda\Pi_{\leq}} \tau \leq \rho \wedge \Gamma \vdash \tau, \rho : \star \ \Leftrightarrow \ \Gamma \vdash_{\lambda P_{\leq}} \tau \leq \rho \ \Leftrightarrow \ \Gamma \vdash_{\lambda P_{\leq}^{alg}} \tau^{\beta_3} \leq \rho^{\beta_3} \wedge \Gamma \vdash \tau, \rho : \star$$

This result implies the admissibility of transitivity in $\lambda\Pi_{\leq}$. Note that no kinding premises is present in the rules, so the subtyping system is independent from the typing system. These rules can be turned into an ordered rewriting system to check subtyping. As subtyping is defined on types rather than on $\beta_3$ normalized types, the algorithm is more efficient.

The $\lambda\Pi_{\leq}$ realizes the desired properties like subject reduction, strong normalization and decidability. The key step of the proof of decidability of subtyping is the proof of $\beta\Gamma$-strong-normalization, to which we have developed an easy and general proof applicable also to $\lambda P_{\leq}, F^{\omega}$ and other systems.

It is expected that the technique can be easily adapted for future extensions and that the algorithm can be used in real implementations for extending the type theory based proof systems with subtyping.

## 10 Semantical Analysis of Subtyping Relation

Recall that the logical interpretation of a subtyping relation $\Gamma \vdash \tau \leq \rho$ consists of two points:
1. $\tau$,$\rho$ are logical propositions and $\tau$ implies $\rho$;
2. any proof $M$ of $\tau$ is also a proof of $\rho$ in the sense that there is a coercion function

$c$ from $\tau$ to $\rho$ such that: (a) $cM$ is a proof of $\rho$; (b) $c$ is "almost" an identity function, formally, $erase(c)$ is equal to the identity function $\lambda x.x$ in type-free $\lambda$-calculus.

This interpretation captures two aspects of subtyping: 1. subtyping as logical implication; 2. subtyping as set inclusion. So the subtyping relation $\tau \leq \rho$ can be understood as "any proof of the proposition $\tau$ can be used as a proof of $\rho$ with straightforward modifications in the type labels". If we take $\tau, \rho$ as specifications for programs, then the subtyping relation means that any programs satisfying the specification of $\tau$ can be straightforwardly adapted for the specification $\rho$. The second point in the above logical interpretation of subtyping has made it clear what this straightforward modification is.

In this section, we mainly analyse the subtyping from the logical implication point of view.

In $\lambda P_{\leq}$ and $\lambda\Pi_{\leq}$, there are subtyping relations between type families. Type families are not types, so they can neither be interpreted as sets nor as formulas. But we use subtyping rules only in the subsumption rule for checking subtyping relations between "true" types, that is, we only want to check $\Gamma \vdash \tau \leq \rho$ such that $\Gamma \vdash \tau, \rho : \star$. In $\lambda\Pi_{\leq}$, one can observe that all judgements in the derivation for such "true" subtyping judgement contains only types of kind $\star$. Therefore, subtyping between type families is not directly used, especially the rule S-A of $\Lambda$-abstraction can be eliminated.

Now we give a logical interpretation for subtyping rules in $\lambda\Pi_{\leq}$ except S-$\Lambda$.

**Conversion-based rules** Observe that rules S-ApR, S-ApSL and S-ApSR concern only the $\beta$-conversion. In logic, two $\beta$-convertible formulas are simply identified. So these rules are easily justified.

**Subtyping $\pi$-types** The logical interpretation of a $\pi$ type $\pi x{:}\tau.\rho$ is the first-order universal quantified formula $\forall x{:}\tau.\rho$. Therefore, the subtyping relation $\Gamma \vdash \pi x{:}\tau.\rho \leq \pi x{:}\tau'.\rho'$ can be understood as the implication between unversal quantified first-order propositions: $\Gamma \vdash \forall x{:}\tau.\rho \Rightarrow \forall x{:}\tau'.\rho'$, where the context $\Gamma$ is used to record the free variables in these formulas.

**Using context variables** The rule S-ApT is an application of subtyping relation defined in the context. For example, if $\alpha \leq \tau : \Pi x_1{:}\tau_1 ... x_n{:}\tau_n.\star$ is defined in the context, then $\alpha$ and $\tau$ represent $n$-ary predicates such that for any $n$ terms $M_1, ..., M_n$

$$\alpha(M_1, ..., M_n) \quad \Rightarrow \quad \tau(M_1, ..., M_n)$$

That is, the relation $\alpha \leq \tau$ in the context can be interpreted as the logical implication

$$\forall x_1{:}\tau_1 ... x_n{:}\tau_n.(\alpha(x_1 ... x_n) \Rightarrow \tau(x_1 ... x_n))$$

The rule S-ApT is just a reformulation of this statement in order to preserve the cut-elimination property.

## 11 Conclusion and Future Work

In this short survey, we have presented subtyping for two type systems: $Co^{\vdash}$ for System-F(c) and $\lambda P_{\leq}, \lambda\Pi_{\leq}$ for $\lambda\Pi$.

There are several common features in these studies. From the syntactic point of view, both in $Co^{\vdash}$ and $\lambda\Pi_{\leq}$ transitivity is admissible. From the semantical side, the interpretation of subtyping as logical entailment with coherent coercions has been first developed in the study of $Co^{\vdash}$ and then applied to $\lambda\Pi_{\leq}$. We are currently studying the coercion version of $\lambda\Pi_{\leq}$ and its properties. Another direction is to use the technique developed in these works to establish subtyping extensions of Calculus of Constructions or other systems in the $\lambda$-cube. The aim is that these extensions by subtyping could be used in the designs of programming languages and in the proof development environments.

**Aknowledgements.** Giuseppe Longo would like to thank Kathleen Milsted and Sergei Soloviev for the longlasting and stimulating collaboration; part of his work was supported by Digital Equipement Corporation, by a generous consulting contract. David Aspinall and Adriana Compagnoni had several clarifying discussions on subtyping dependent types with Gang Chen; his work in Paris was supported by Programme PRA M4, Association Franco-Chinoise pour la Recherche Scientifique et Technique, and Bourse du Ministère des Affaires Etrangères du Gouvernement Français.

## References

[AC96] David Aspinall and Adriana Compagnoni. Subtyping dependent types, 1996. to appear in LICS96.

[ACC93] M. Abadi, L. Cardelli, and P.-L. Curien. Formal parametric polymorphism. In Dezani, Ronchi, and Venturini, editors, *Böhm Festschrift*. 1993.

[AL91] Andrea Asperti and Giuseppe Longo. *Categories, Types and Structures: an introduction to Category Theory for the working computer scientist*. M.I.T.- Press, 1991. (pp. 1–300).

[Bar92] Henk Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

[BCDC83] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J.Symbolic Logic*, 48:931–940, 1983.

[BL90] Kim B. Bruce and Giuseppe Longo. A modest model of records, inheritance, and bounded quantification. *Information and Computation*, 87:196–240, 1990.

[BM92] Kim Bruce and John Mitchell. PER models of subtyping, recursive types and higher-order polymorphism. In *Proceedings of the Nineteenth ACM Symposium on Principles of Programming Languages*, Albequerque, NM, January 1992.

[BTCG+91] V. Breazu-Tannen, T. Coquand, C.A. Gunter, , and A. Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93:172–221, 1991. Part of this work appeared in the *Proc. of the 4th International Workshop on Database Programming Languages, Wordkshop in Computing series, Springer Verlag.*

[C+86] Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Car87] Luca Cardelli. Typechecking dependent types and subtypes, December 1987.

[CG92] P. L. Curien and G. Ghelli. Coherence of subsumption, minimum typing and the type checking in $F_{\leq}$. *Mathematical Structures in Computer Science*, 2(1), 1992.

[CGL95] Giuseppe Castagna, Giorgio Ghelli, and Giuseppe Longo. A calculus for overloaded functions with subtyping. *Information and Computation*, 117(1):115–135, 15 February 1995. revised version of Technical Report 92-4 Laboratoire d'Informatique, Ecole Normale Supérieure - Paris.

[Che96] Gang Chen. Dependent type system with subtyping, 1996. Mémoire de DEA, Laboratoire d'Informatique ENS, and Université Paris 7.

[CL91] L. Cardelli and G. Longo. A semantic basis for Quest. *Journal of Functional Programming*, 1(4):417–458, 1991.

[CMMS91] L. Cardelli, S. Martini, J.C. Mitchell, and A. Scedrov. An extension of system F with subtyping. In T. Ito and A.R. Meyer, editors, *Theoretical Aspects of Computer Software*, pages 750–771. Springer-Verlag, September 1991. LNCS 526.

[CNSvS94] Thierry Coquand, Bengt Nordström, Jan M. Smith, and Björn von Sydow. Type theory and programming. *Bulletin of the European Association for Theoretical Computer Science*, 52:203–228, February 1994.

[Com94] Adriana B. Compagnoni. Subtyping in $F^{\omega}_{\wedge}$ is decidable. Technical Report ECS-LFCS-94-281, LFCS University of Edinburgh, January 1994.

[Coq92] Thierry Coquand. Pattern matching with dependent types. In *Proceedings on Types for Proofs and Programs*, pages 71–83, 1992.

[Cou95] Judicaelë Courant. Rapport de magistère, Novembre 1995.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, December 1985.

[DFH+93] Gilles Dowek, Amy Felty, Hugo Herbelin, Gérard Huet, Chet Murthy, Catherine Parent, Christine Paulin-Mohring, and Benjamin Werner. The Coq proof assistant user's guide. Rapport Techniques 154, INRIA, Rocquencourt, France, 1993. Version 5.8.

[FL96] T. Fruchart and G. Longo. Carnap's remarks on impredicativity and the gener icity theorem, 1996. in preparation.

[Gir71] Girard, J.-Y. Une Extension de l'Interprétation de Gödel à l'Analyse, et son Application à l'Élimination des Coupures dans l'Analyse et la Théorie des Types. In Jens Erik Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92, Amsterdam, 1971. North-Holland.

[Has93] Ryu Hasegawa. Categorical data types in parametric polymorphism. 1993.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[Hic95] Jason J. Hickey. Formal abstract data types, December 1995. Unpublished draft.

[Hyl82] J.M.E. Hyland. The effective topos. In A.S.Troelstra and D.S. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*, pages 165–216. North-Holland, Amsterdam, 1982.

[Hyl88] J.M.E. Hyland. A small complete category. *Annals of Pure and Applied Logic*, 40, 1988.

[Jac95] B. Jacobs. Subtyping and bounded quantification from a fibred perspective. In *Conference on Mathematical Foundations of Programming Semantics*. New Orleans, U.S.A, 1995.

[LM91] G. Longo and E. Moggi. Constructive natural deduction and its $\omega$-set interpretation. *Mathematical Structures in Computer Science*, 1(2):215–253, 1991.

[LMS93] Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. The Genericity Theorem and effective Parametricity in Polymorphic lambda-calculus. *Theoretical Computer Science*, 121:323–349, 1993.

[LMS95] Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. A Logic of Subtyping, extended abstract. In *Logic in Computer Science, LICS'95*, San Diego, June 1995. Complete version available on ftp.ens.fr/pub/dmi/users/longo/.

[LMS96] Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. Coherence and transitivity of subtyping as entailment. Technical report, 1996. submitted for publication , available on ftp.ens.fr/pub/dmi/users/longo/.

[Lon95] Giuseppe Longo. Parametric and type-dependent polymorphism. *Fundamenta Informaticae*, 22(1-2):69–92, 1995.

[LP92] Zhaohui Luo and Robert Pollack. The LEGO proof development system: A user's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh, May 1992.

[Luo96] Zhaohui Luo. Coercive subtyping in type theory, 1996. draft.

[Mit88] J.C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76:211–249, 1988.

[MR92] Q. Ma and J.C. Reynolds. Types, abstraction, and parametric polymorphism, part 2. In S. Brookes et al., editor, *Mathematical Fundations of Programming Semantics*, volume 598, pages 1–40. 1992.

[Pfe89] Frank Pfenning. Elf: A language for logic definition and verified metaprogramming. In *Fourth Annual Symposium on Logic in Computer Science*, pages 313–322, Pacific Grove, California, June 1989. IEEE Computer Society Press.

[Pfe93] Frank Pfenning. Refinement types for logical frameworks. In *Informal Proceedings of the 1993 Workshop on Types for Proofs and Programs*, May 1993.

[PS] B. Pierce and M. Steffen. Higher-order subtyping. To appear in: *Theoretical Computer Science*, 1996. A preliminary version appeared in the proceedings of the *IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET)*, June 1994, and as University of Edinburgh technical report ECS-LFCS-94-280 and Universität Erlangen-Nürnberg Interner Bericht IMMD7-01/94, January 1994.

[Rey83] J.C. Reynolds. Types, abstractions and parametric polymorphism. In R.E.A. Mason, editor, *Information Processing '83*, pages 513–523. North-Holland, 1983.

[Tiu95] J. Tiuryn. Equational axiomatization of bicoercibility for polymorphic types. In *Proc. 15th Conference Foundations of software Technology and Theoretical Computer Science*, LNCS, pages 166–179. Springer-Verlag, 1995.

[Tiu96] J. Tiuryn. A sequent calculus for subtyping polymorphic types. unpublished, 1996.

[TU96] J. Tiuryn and P. Urzyczyn. The subtyping problem for second-order types is undecidable. In *Proc. LICS 96*, 1996. To appear.

[Wel95] J. Wells. The undecidability of mitchell's subtyping relation. Technical report, Dept., Boston University, December 1995.