Implicit Coercions in Type Systems

Gilles Barthe

CWI, PO Box 94079, 1090 GB Amsterdam, The Netherlands.
Email:gilles@cwi.nl

Abstract. *We propose a notion of pure type system with implicit coercions. In our framework, judgements are extended with a context of coercions Δ and the application rule is modified so as to allow coercions to be left implicit. The setting supports multiple inheritance and can be applied to all type theories with Π-types. One originality of our work is to propose a computational interpretation for implicit coercions. In this paper, we demonstrate how this interpretation allows a strict control on the logical properties of pure type systems with implicit coecions.*

1 Introduction

The increasing importance of mathematical software has been accompanied by a drift of mainstream mathematics towards mathematical logic and the foundations of mathematics. Before mathematical software, formal systems were generally seen both by logicians and mathematicians as safe heavens into which mathematics could theoretically be embedded. With powerful mathematical software, there is now a genuine interest in developing mathematics within a formal system (see e.g. [7, 13]). This radical change in the relationship between mathematics and mathematical logic calls for a new strategy in the design of formal systems. New criteria such as comfort, efficiency and suitability to implementation, have to be taken into account when assessing the value of formal systems. The new challenge is to provide formal systems for *feasible formal mathematics*. Despite an early proposal by N.G. de Bruijn ([8]), much remains to be done in this direction. There are still notable differences between formal and informal mathematics:

- *at the level of reasoning:* the level of detail required in formal proofs is much greater than the level of detail in informal proofs; reasoning in a formal system requires every single step to be decomposed in terms of primitive rules.
- *at the level of language:* formal mathematics requires extreme rigour in the formulation of statements. Commonly used mathematical expressions, such as $x \in G$, where G is a group, are not always well-formed in a formal language because it is often required that the expression on the right hand side of $\in$ should be a set. Hereafter we shall refer to this problem as implicit syntax.

While the first problem has been partially solved by a variety of tools (tactics, inductionless induction, partial reflection and decision procedures), the problem of implicit syntax has received little attention in the context of proof-checking[1].

The goal of this paper is to contribute to the study of implicit syntax in proof-checking. In this paper, we focus on one specific aspect of implicit syntax, namely *implicit coercions*; by implicit coercions, we refer to a grammatical convention which allows to apply a map $f : A \to B$ to an element a of A' whenever there is a coercion from A' to A. We propose a notion of pure type system with implicit coercions (PTSC for short) whose judgements are of the form $\Delta|\Gamma \vdash M : A$ where Δ is a set of legal terms. Elements of Δ, which are called *coercions*, specify which are the arguments that can be omitted in an expression. A typical derivation is

$$\frac{i : \mathbb{N} \to \mathbb{Z}| \vdash 3 : \mathbb{N} \quad i : \mathbb{N} \to \mathbb{Z}| \vdash \mathsf{minus} : \mathbb{Z} \to \mathbb{Z}}{i : \mathbb{N} \to \mathbb{Z}| \vdash \mathsf{minus}\ 3 : \mathbb{Z}}$$

The derivation is valid because $i : \mathbb{N} \to \mathbb{Z}$ is assumed as an implicit coercion (of course, there are suitable rules to introduce coercions in a context). One of the novelties of our approach is to give a computational interpretation of implicit coercions. We define a (conditional) reduction relation $\to_\epsilon$ which makes coercions explicit. There are several advantages in having such a relation:

1. the equational theory of the type system is rich enough to identify terms which should be identified (such as minus 3 and minus (i 3));
2. expliciting a term is viewed as a computational process interacting with β-reduction;
3. by identifying suitable terms, $\to_\epsilon$ forces pure type systems with implicit coercions to be conservative extensions of pure type systems.

We shall show that under certain conditions $\to_\epsilon$ is normalising (i.e. the use of implicit coercions can be removed from any derivation) and confluent (i.e. there is essentially an unique way of making a term explicit). The relevance of these properties will be discussed in Section 3.2.

Related work The use of implicit coercions or subtyping in proof-checking has been considered by several authors (see [1, 3, 4] for the former and [2, 5, 14, 15] for the latter). In [4], the author reports on a medium-scale example of formalisation of mathematics using implicit coercions. See also [10, 16] for work on overloading and implicit syntax respectively.

Contents of the paper The paper is organised as follows: in the next section, we give an informal motivation of the syntax of pure type systems with implicit coercions by giving an abstract definition of implicit syntax. In section 3, we present the syntax for pure type systems with implicit coercions. In order to look at interesting examples, we consider pure type systems with Σ-types. In section 4, we exemplify the use of our syntax in the formalisation of algebra. In section 5, we study the basic meta-theory of implicit coercions and show that

[1] Some of the concepts involved in implicit syntax such as overloading and argument synthesis have been thoroughly investigated in the context of programming languages ([9, 18, 19]).

pure type systems with implicit coercions provide an implicit syntax for pure type systems. Possible extensions to our work are discussed in section 6. Section 7 contains some final remarks.

Acknowledgements I would especially like to thank P. Aczel for many discussions on classes and on the formalisation of mathematics. The paper has also benefited from comments from A. Bailey, A. Saibi and the anonymous referees. This work was partially carried out at the Universities of Manchester and Nijmegen with the financial support of the Esprit project 'Types: Types for programs and proofs'.

2 What is implicit syntax?

In this section, we give an abstract definition of the concept of implicit syntax. There are two fundamental assumptions about implicit syntax:

1. it is meant to improve (not to increase) the expressivity of a formal system;
2. it should not affect the theory of the formal system.

To fix ideas, we shall make the ideas precise in the abstract setting of formal systems.

Definition 1 *A formal system as a triple* $(A, =_A, \mathsf{Thm})$ *where* A *is a set of expressions,* $=_A$ *is an equality relation on* A *and* Thm *is a binary relation on* A.

For example, every pseudo-context Γ of a pure type system λS determines a formal system $\mathsf{F}_{\lambda S}(\Gamma)$ by taking A to be the set of Γ-terms, $=_A$ to be β-equality and Thm_A to be the typing relation :.

Definition 2 *An implicit syntax for a formal system* $\mathbb{A} = (A, =_A, \mathsf{Thm}_A)$ *consists of a formal system* $\mathbb{B} = (B, =_B, \mathsf{Thm}_B)$ *and a map* $e : B \rightarrow A$ *such that:*

1. $A \subseteq B$;
2. *for every* $b \in B$, $b =_B e(b)$;
3. *for every* $a_1, a_2 \in A$, $a_1 =_A a_2 \quad \Leftrightarrow \quad a_1 =_B a_2$;
4. *for every* $a_1, a_2 \in A$, $(a_1, a_2) \in \mathsf{Thm}_A \quad \Leftrightarrow \quad (a_1, a_2) \in \mathsf{Thm}_B$;
5. *for every* $b_1, b_2 \in B$, $(b_1, b_2) \in \mathsf{Thm}_B \quad \Leftrightarrow \quad (e(b_1), e(b_2)) \in \mathsf{Thm}_A$.

The definition is meant to capture idea is that B should contain more terms than A (requirement 1) and that every term in B could be translated into a term in A with the same meaning (requirement 2). Moreover, $=_B$ (resp. Thm_B) should coincide with $=_A$ (resp. Thm_A) on A (requirements 3 and 4) and e should preserve the logical structure of the formal system (requirement 5).

The emphasis of this paper will be on showing that PTSCs are an implicit syntax for PTSs (this will be stated precisely in Section 9). We believe this perspective to be fundamental for proof-checking as it provides a means to ensure that PTSCs have a suitable logical interpretation.

3 Pure type systems with implicit coercions

In this section, we define a deductive system for pure type systems with implicit coercions. In order to treat interesting examples, we consider an extension of pure type systems with Σ-types. However, our approach is independent from type constructors (we only need a function space former) and does not require the presence of Σ-types.

3.1 Syntax

Definition 3 *1. A pure type system is specified by a quadruple*

$$\lambda S = (\mathsf{Sort}, \mathsf{Axiom}, \mathsf{Rule}_\Pi, \mathsf{Rule}_\Sigma)$$

where Sort *is a set,* Axiom $\subseteq$ Sort$\times$Sort *and* Rule$_\Pi$, Rule$_\Sigma$ $\subseteq$ Sort$\times$Sort$\times$Sort.

2. The set of pseudo-terms *of a pure type system* $\lambda S = (\mathsf{Sort}, \mathsf{Axiom}, \mathsf{Rule}_\Pi, \mathsf{Rule}_\Sigma)$ *is given by the following abstract syntax:*

$$T = V|\mathsf{Sort}|\Pi V : T.T|\lambda V : T.T|TT|\Sigma V : T.T|\mathsf{pair}(T,T)|\mathsf{fst}\ T|\mathsf{snd}\ T$$

where V is a fixed set of variables.

3. A pseudo-coercion *is a pair of the form $(\lambda y : B.t, B \rightarrow C)$ where $\lambda y : B.t$ and $B \rightarrow C$ are pseudo-terms. Sets of pseudo-coercions are usually denoted by Δ.*

4. The closure *Δ^+ of a set Δ of pseudo-coercions: it is the least set such that*

$$\lambda x : A_1.c_n(c_{n-1}(\ldots(c_1 x)\ldots)) : A_1 \rightarrow B_n \quad \in \Delta^+$$

whenever $(c_i : A_i \rightarrow B_i) \in \Delta$ for $i = 1, \ldots, n$ and $A_{i+1} =_\beta B_i$ for $i = 1, \ldots, n-1$.

5. Let Δ be a set of pseudo-coercions. The relation $\rightarrow_{\epsilon(\Delta)}$ is defined on pseudo-terms as the compatible closure of $t\ u \rightarrow_{\epsilon(\Delta)} t\ (i\ u)$, where it is assumed that $i \in \Delta^+$. $\twoheadrightarrow_{\epsilon(\Delta)}$ (resp. $=_{\epsilon(\Delta)}$) is defined as the reflexive, transitive (resp. reflexive, symmetric and transitive) closure of $\rightarrow_{\epsilon(\Delta)}$.

6. A pseudo-context *is a sequence of the form*

$$\lambda y : B_1.t_1 : B_1 \rightarrow C_1, \ldots, \lambda y : B_m.t_m : B_m \rightarrow C_m | x_1 : A_1, \ldots, x_n : A_n$$

where the $\lambda y : B_i.t_i : B_i \rightarrow C_i$'s are pseudo-coercions, the x_i's are variables and the A_i's are pseudo-terms. Pseudo-contexts are usually written as $\Delta|\Gamma$.

7. A judgement *is a triple of the form $(\Delta|\Gamma, M, A)$ where $\Delta|\Gamma$ is a pseudo-context and M, A are pseudo-terms.*

8. The derivability relation $\vdash$ is defined by the rules of Table 1.

Few explanations seem in order to justify our syntax: all the rules except (Entry), (Method) and the conversion rules are straightforward adaptations of the rules for pure type systems. The (Method) rule introduces implicit syntax in the system by allowing to apply $f : \Pi x : A.B$ to elements of several types (in

(Axiom)	$\mid\vdash c : s$	if $(c, s) \in \mathsf{Axiom}$
(Start)	$\dfrac{\Delta \mid \Gamma \vdash A : s}{\Delta \mid \Gamma, x : A \vdash x : A}$	if $x \notin \Gamma$
(Weakening)	$\dfrac{\Delta \mid \Gamma \vdash t : A \quad \Delta \mid \Gamma \vdash B : s}{\Delta \mid \Gamma, x : B \vdash t : A}$	if $x \notin \Gamma$
(Product)	$\dfrac{\Delta \mid \Gamma \vdash A : s_1 \quad \Delta \mid \Gamma, x : A \vdash B : s_2}{\Delta \mid \Gamma \vdash \Pi x : A.B : s_3}$	if $(s_1, s_2, s_3) \in \mathsf{Rule}_\Pi$
(Application)	$\dfrac{\Delta \mid \Gamma \vdash t : \Pi x : A.B \quad \Delta \mid \Gamma \vdash u : A}{\Delta \mid \Gamma \vdash tu : B[u/x]}$	
(Abstraction)	$\dfrac{\Delta \mid \Gamma, x : A \vdash t : B \quad \Delta \mid \Gamma \vdash \Pi x : A.B : s}{\Delta \mid \Gamma \vdash \lambda x : A.t : \Pi x : A.B}$	
(Sum)	$\dfrac{\Delta \mid \Gamma \vdash A : s_1 \quad \Delta \mid \Gamma, x : A \vdash B : s_2}{\Delta \mid \Gamma \vdash \Sigma x : A.B : s_3}$	if $(s_1, s_2, s_3) \in \mathsf{Rule}_\Sigma$
(Pairing)	$\dfrac{\begin{array}{c}\Delta \mid \Gamma \vdash t_1 : A \quad \Delta \mid \Gamma \vdash t_2 : B[t_1/x] \\ \Delta \mid \Gamma \vdash \Sigma x : A.B : s\end{array}}{\Delta \mid \Gamma \vdash \mathsf{pair}(t_1, t_2) : \Sigma x : A.B}$	
(First Projection)	$\dfrac{\Delta \mid \Gamma \vdash t : \Sigma x : A.B}{\Delta \mid \Gamma \vdash \mathsf{fst}\ t : A}$	
(Second Projection)	$\dfrac{\Delta \mid \Gamma \vdash t : \Sigma x : A.B}{\Delta \mid \Gamma \vdash \mathsf{snd}\ t : B[\mathsf{fst}\ t/x]}$	
(Entry)	$\dfrac{\Delta \mid \Gamma \vdash c : C \quad \Delta' \mid \Gamma' \vdash t : A \to B}{\Delta, t : A \to B \mid \Gamma \vdash c : C}$	PROVISO
(Method)	$\dfrac{\Delta \mid \Gamma \vdash t : \Pi x : A.B \quad \Delta \mid \Gamma \vdash u : A'}{\Delta \mid \Gamma \vdash tu : B[i\ u/x]}$	if $(i, A'' \to A_0) \in \Delta^+$ with $A =_\beta A''$
(β-conversion)	$\dfrac{\Delta \mid \Gamma \vdash c : C \quad \mid \Gamma' \vdash C : s \quad \mid \Gamma' \vdash C' : s}{\Delta \mid \Gamma \vdash c : C'}$	if $C =_\beta C'$ and $\Gamma \to_{\epsilon(\Delta)} \Gamma'$
(ϵ-conversion)	$\dfrac{\Delta \mid \Gamma \vdash c : C \quad \Delta \mid \Gamma \vdash C' : s}{\Delta \mid \Gamma \vdash c : C'}$	if $C \downarrow_{\epsilon(\Delta)} C'$

Table 1. RULES FOR DERIVATIONS IN PURE TYPE SYSTEMS WITH IMPLICIT COERCIONS

fact to all the types which are linked to A by a pseudo-coercion). Note that the predicate of the conclusion is $B[i\ u/x]$ rather than $B[u/x]$ because we do not know if the latter is legal. The (Entry) rule enables new coercions to be introduced provided a certain PROVISO is satisfied. The role of the PROVISO is discussed in Subsection 3.2. As for the conversion rules, there are two rules: one for β-conversion and one for ϵ-conversion. The choice for these rules is given in Subsection 3.3.

3.2 The coherence and conservativity properties

In Section 2, we made two fundamental assumptions for implicit syntax and formalised these assumptions in Definition 2. Here we see how to instantiate the definition to the framework of pure type systems with implicit coercions.

In our context, the translation map from implicit to explicit syntax has an obvious candidate namely ϵ-reduction. We would like that for every legal contexts

$\Delta|\Gamma$ and $|\Gamma'$ such that $\Gamma \twoheadrightarrow_{\epsilon(\Delta)} \Gamma'$, the set of legal $\Delta|\Gamma$-terms (with $\beta\epsilon(\Delta)$-equality) is an implicit syntax for the set of legal Γ'-terms (with β-equality). This is a consequence of the following two properties:

1. for every derivation $\Delta|\Gamma \vdash M : A$, there exists a derivation $|\Gamma' \vdash M' : A'$ with $\Gamma \twoheadrightarrow_{\epsilon(\Delta)} \Gamma'$, $M \twoheadrightarrow_{\epsilon(\Delta)} M'$ and $A \twoheadrightarrow_{\epsilon(\Delta)} A'$;
2. for every derivations $\Delta|\Gamma \vdash M : A$, $|\Gamma' \vdash M' : A'$ and $|\Gamma'' \vdash M'' : A''$ such that $\Gamma \twoheadrightarrow_{\epsilon(\Delta)} \Gamma', \Gamma''$, $M \twoheadrightarrow_{\epsilon(\Delta)} M', M''$ and $A \twoheadrightarrow_{\epsilon(\Delta)} A', A''$ one has $\Gamma' =_\beta \Gamma''$, $M' =_\beta M''$ and $A' =_\beta A''$.

We respectively call them the *conservativity* property and the *coherence* property. The role of the proviso is to ensure that both properties hold. In order to simplify the problem, we require coercions to be closed.

Definition 4 - *A pseudo-coercion $\lambda x : A.t : A \to B$ is* simple *if $\lambda x : A.t$ and $A \to B$ are closed.*
- *A set Δ of pseudo-coercions is* coherent *if all coercions are simple and*
 1. $\forall (\lambda x : A.i, A \to B), (\lambda x' : A'.i', A' \to B') \in \Delta^+.$
 $$A =_\beta A' \quad \wedge \quad B =_\beta B' \quad \Rightarrow i\,[x'/x] =_\beta j$$
 2. $\forall (\lambda x : A.i, A \to B) \in \Delta^+.\quad A =_\beta B \quad \Rightarrow \quad i =_\beta x$

The entry rule is now formulated as

$$\frac{\Delta|\Gamma \vdash c : C \qquad | \vdash t : A \to B}{\Delta, t : A \to B|\Gamma \vdash c : C} \text{ if } \Delta \cup \{t, A \to B\} \text{ is coherent.}$$

Note that for the sake of simplicity we require coercions to be fully explicit, i.e. to be derivable in the empty context.

3.3 The conversion rule

The conversion rule is split in two (see Table 1). There is a β-conversion rule which allows to convert fully explicit types (i.e. types which are derivable in a context with no pseudo-coercions) and an ϵ-conversion rule which allows to convert types which are related by ϵ-reduction. There are two reasons for such a choice:

- it seems natural to postpone computations until the term is fully explicit. With this view, reduction is a succession of two processes, *explicitation* (i.e. ϵ-reduction) and *computation* (i.e. β-reduction).
- it is unclear what may be the effects of a very general conversion rule, such as

$$\frac{\Delta|\Gamma \vdash c : C \;\; \Delta|\Gamma \vdash C' : s}{\Delta|\Gamma \vdash c : C'} \text{ if } C =_{\beta\epsilon(\Delta)} C'$$

4 Implicit coercions at work

In this section, we exemplify the use of implicit coercions in the formalisation of mathematics.

4.1 Formalising algebra with implicit coercions

The Calculus of Constructions with strong sums $CC\Sigma$ has two sorts, $*$ and $\Box$, related by the axiom $* : \Box$. The rules for products are $(*, *)$, $(\Box, *)$, $(*, \Box)$, $(\Box, \Box)$. The rules for sums are $(*, *, *)$ and $(\Box, *, \Box)$. The system is Church-Rosser, strongly normalising, consistent and has decidable type-checking. In $CC\Sigma$, it is possible to define several basic algebraic types, such as sets, groupoids, monoids... We give some of these definitions here. For the sake of simplicity, we take $Set = *$.

$$\begin{aligned}
Gpd &= \Sigma T : Set.\Sigma o : T \rightarrow T \rightarrow T.\mathsf{AxGpd}\ o \\
AbGpd &= \Sigma T : Set.\Sigma o : T \rightarrow T \rightarrow T.(\mathsf{AxGpd}\ o) \wedge (\mathsf{Comm}\ o) \\
Mon &= \Sigma T : Set.\Sigma o : T \rightarrow T \rightarrow T.\Sigma e : T.\mathsf{AxMon}\ o\ e \\
AbMon &= \Sigma T : Set.\Sigma o : T \rightarrow T \rightarrow T.\Sigma e : T.(\mathsf{AxMon}\ o\ e) \wedge (\mathsf{Comm}\ o) \\
Grp &= \Sigma T : Set.\Sigma o : T \rightarrow T \rightarrow T.\Sigma e : T.\Sigma i : T \rightarrow T.\mathsf{AxGrp}\ o\ e\ i \\
AbGrp &= \Sigma T : Set.\Sigma o : T \rightarrow T \rightarrow T.\Sigma e : T.\Sigma i : T \rightarrow T.(\mathsf{AxGrp}\ o\ e\ i) \wedge (\mathsf{Comm}\ o)
\end{aligned}$$

where Comm o is the proposition stating that o is commutative and AxGpd, AxMon and AxGrp respectively state the axioms of groupoids, monoids and groups. The canonical maps between these types, as shown in figure 1, yield a coherent set Δ of coercions. The context of coercions Δ can be used to formalise algebra.

$$\begin{tikzcd}
Grp \arrow[d, "\mathsf{GrptoMon}"'] & AbGrp \arrow[l, "\mathsf{AbGrptoGrp}"'] \arrow[d, "\mathsf{AbGrptoAbMon}"] \\
Mon \arrow[d, "\mathsf{MontoGpd}"'] & AbMon \arrow[l, "\mathsf{AbMontoMon}"'] \arrow[d, "\mathsf{AbMontoAbGpd}"] \\
Gpd \arrow[d, "\mathsf{GpdtoSet}"'] & AbGpd \arrow[l, "\mathsf{AbGpdtoGpd}"'] \\
Set &
\end{tikzcd}$$

Fig. 1. Basic coercions for algebra

For example, we can apply comp $\equiv \lambda G : Gpd.\mathsf{fst}\ (\mathsf{snd}\ G)$ to monoids, groups... If moreover we define Op2 as $\lambda T : Set.T \rightarrow T \rightarrow T$ then comp G is of type Op2 G whenever $G : Monoid$, $G : Group$...

4.2 Typical features of implicit coercions

They include:

- *uniformity:* we do not have any restriction on the domain and codomain of a coercion. This enables us to treat in an identical manner canonical coercions of a different nature, such as the one from naturals to integers or the one from groups to sets;
- *multiple inheritance:* there can be several coercions maps with the same domain and there might be more than one path between two types. For example, one can have four coercion maps $f : A \rightarrow B$, $g : B \rightarrow C$, $h : A \rightarrow B'$ and $i : B' \rightarrow C$ provided $g \circ f$ and $i \circ h$ are extensionally equal.
- *top-down introduction of coercions:* it is possible to introduce a coercion $f : A \rightarrow B$ and then a coercion $g : B \rightarrow C$. In fact, coercions can be introduced in any order. This solves the problem of "super-type" which occurs when coercions are required to be built up in a tree-like manner. In our syntax, there is no problem in defining the natural, then the integers and declaring a coercion from natural to integers, then build the rationals and declare a coercion from integers to rationals...
- *splitting a coercion:* it is possible to "split" a coercion $f : A \rightarrow B$ into two coercions $g : A \rightarrow C$ and $h : C \rightarrow B$ provided f and $h \circ g$ are extensionally equal. This allows to postpone the introduction of new notions until they are needed. For example, one does not need to introduce the notion of monoid before the notion of group in order to split the coercion from groups to groupoids into a coercion from groups to monoids and from monoids to groupoids;
- *back and forth coercions:* it is possible to have two coercions $f : A \rightarrow B$ and $g : B \rightarrow A$ provided the maps are mutually inverse. Back and forth coercions allow for equivalent representations of a same mathematical object to be used without any major bureaucratic difficulty. This is very convenient for re-usability as experience shows that different users chose different but equivalent representations of a same mathematical object. However, the absence of η-conversion limits significantly the usefulness of back and forth coercions, as seen in the next subsection.

4.3 Limitations of implicit coercions

Our syntax for implicit coercions suffers from some limitations and should be considered as a preliminary step towards a theory of implicit syntax. We try to discuss some of these limitations briefly.

Re-usability The definition of the closure of a set of coercions does not allow for an immediate re-use of methods as can be seen in the following example. Assume we have a two types $\mathsf{ColourPoint}$ and Point with an implicit coercion $i : \mathsf{ColourPoint} \rightarrow \mathsf{Point}$. If we have a map $\mathsf{move} : \mathsf{Nat} \times \mathsf{Point} \rightarrow \mathsf{Point}$ and $c : \mathsf{ColourPoint}$, then we will not have $\mathsf{movepair}(n, c) : \mathsf{Point}$ for every natural number n. This choice has been made deliberately for the simplicity of the syntax. Besides, we can always define another implicit coercion from $\mathsf{Nat} \times \mathsf{ColourPoint}$ to $\mathsf{Nat} \times \mathsf{Point}$.

Efficient proof-checking The conversion rules are rather inefficient for proof-checking because they require computations to be postponed until terms are fully explicit. We conjecture it can be solved by considering a more general form of conversion. In fact, the essential property to prove the coherence and conservativity properties is that for every application of conversion

$$\frac{\Delta|\Gamma \vdash c : C \quad \Delta|\Gamma \vdash C' : s}{\Delta|\Gamma \vdash c : C'}$$

and derivations $|\Gamma_1 \vdash C_1 : s_1$ and $|\Gamma_1' \vdash C_1' : s_1'$ with $\Gamma \twoheadrightarrow_{\epsilon(\Delta)} \Gamma_1, \Gamma_1'$ and $C \twoheadrightarrow_{\epsilon(\Delta)} C_1, C_1'$, one has $C_1 =_\beta C_1'$.

The coherence requirement The definition of coherent set of pseudo-coercions requires equality up to β-conversion. In practice, natural sets of pseudo-coercions do not respect equality up to β-conversion. For example, the swapping maps $\mathsf{swap}_1 : (A \times B) \rightarrow (B \times A)$ and $\mathsf{swap}_2 : (B \times A) \rightarrow (A \times B)$ where A and B are closed types do not form a coherent set of coercions. However, this fact is closely related to the choice of β-equality as the primitive notion of equality for pure type systems. In our view, it is a problem of pure type systems not of implicit coercions.

Polymorphic and general coercions The restriction to simple coercions is a serious one. In practice, one might want to consider polymorphic coercions (of closed type $\Pi \mathbf{x} : \mathbf{A}.B \rightarrow C$) or even general coercions (of possibly open type $\Pi \mathbf{x} : \mathbf{A}.B$. For example, one might want to define the coercion $\mathsf{collapse} : \Pi T : \mathsf{Type}.List\ T \rightarrow Multiset\ T$ which transforms a list of elements of an arbitrary set into a multi-set by forgetting the ordering.

Unfortunately, the formulation of the proviso for polymorphic coercions becomes quite intrinsic and is left as a subject for future work.

5 The coherence and conservativity properties

In this section, we prove that implicit coercions have the coherence and conservativity properties. Before we establish some preliminary results.

5.1 The rule $(Entry - A)$

The set of derivable judgements remains unchanged if one considers the restricted entry rule $(Entry - A)$

$$\frac{\Delta| \vdash c : s \mid \vdash t : A \rightarrow B}{\Delta, t : A \rightarrow B| \vdash c : s} \quad \text{if } (c, s) \in \mathsf{Axiom} \text{ and } \Delta \cup \{t, A \rightarrow B\} \text{ is coherent}$$

The set of derivable judgements remains unchanged if we replace $(Entry)$ by $(Entry - A)$. The proof proceeds by induction on the derivations and uses induction loading: we prove that if $\Delta|\Gamma \vdash M : A$ is derivable and $\Delta' \supseteq \Delta$ is coherent, then $\Delta'|\Gamma \vdash M : A$ in the system with $(Entry - A)$.

5.2 Normal forms

We introduce the notion of $\epsilon(\Delta)$-normal form. Because of possible loops in the graph of coercions, we are forced to consider a slightly weaker notion than usual. We start with some preliminary results.

Definition 5 *$M \twoheadrightarrow_{\epsilon(\Delta|\Gamma)} N$ if there exists A such that $\Delta|\Gamma \vdash M, N : A$ and $M \twoheadrightarrow_{\epsilon(\Delta)} N$.*

The following fact is easy to establish but nevertheless important.

Lemma 6 *If $\Delta|\Gamma \vdash M : A$ and $\Delta|\Gamma \vdash N : B$ with $M \twoheadrightarrow_{\epsilon(\Delta)} N$, then $\Delta|\Gamma \vdash N : A$.*

The above lemma gives an alternative definition of $\twoheadrightarrow_{\epsilon(\Delta|\Gamma)}$.

Definition 7 *A term M is in $\epsilon(\Delta|\Gamma)$-normal form if*

- *there exists A such that $\Delta|\Gamma \vdash M : A$;*
- *if $M \twoheadrightarrow_{\epsilon(\Delta|\Gamma)} P$, then there exists N such that $P \twoheadrightarrow_{\epsilon(\Delta|\Gamma)} N$ and $N \twoheadrightarrow_{\beta} M$.*

We will show that a term is in $\epsilon(\Delta|\Gamma)$-normal form if it is typable in a context without coercions. As usual, we say M has $\epsilon(\Delta|\Gamma)$-normal form N if N is in $\epsilon(\Delta|\Gamma)$-normal form, $\Delta|\Gamma \vdash M : A$ and $M \twoheadrightarrow_{\epsilon(\Delta|\Gamma)} N$. The notion of normal form and reduction on contexts is defined recursively. We write, somewhat loosely, $\Gamma, x : A \twoheadrightarrow_{\epsilon(\Delta)} \Gamma', x : A'$ if $\Gamma \twoheadrightarrow_{\epsilon(\Delta)} \Gamma'$ and $A \twoheadrightarrow_{\epsilon(\Delta|\Gamma)} A'$.

5.3 Coherence

We show that that the coherence property holds.

Proposition 8 (Coherence) *Let $\Delta|\Gamma \vdash M : A$ be a derivable judgement. Let M_1, M_2 be $\epsilon(\Delta|\Gamma)$-normal forms for M. Then $M_1 =_\beta M_2$.*

Proof: the proposition is proved by induction on the structure of the terms. The only interesting case is when $M = M_1 M_2$. By induction hypothesis, M_1 and M_2 have at most one $\epsilon(\Delta|\Gamma)$-normal form up to convertibility. Assume M has two $\epsilon(\Delta|\Gamma)$-normal forms N and P. We show they are β-convertible. First, note that there exist coercions $i_1, \ldots, i_n, j_1, \ldots, j_m$ such that $N = N_1\ (i_1\ (\ldots\ (i_n\ N_2)\ldots))$ and $P = P_1\ (j_1\ (\ldots\ (j_p\ P_2)\ldots))$, where N_k and P_k are $\epsilon(\Delta|\Gamma)$-normal forms of M_k $(k = 1, 2)$. By generation lemma, $\Delta|\Gamma \vdash N_1', P_1' : \Pi x : A.B$ and $\Delta|\Gamma \vdash N_2', P_2' : A'$ with A' linked to A. Again by generation, $\Delta|x : A' \vdash i_1\ (\ldots\ (i_n\ x)\ldots) : A' \rightarrow A$ and $\Delta|x : A' \vdash j_1\ (\ldots\ (j_p\ x)\ldots) : A' \rightarrow A$. Both terms have a $\epsilon(\Delta, x : A')$-normal form, say I and J respectively, with $\Delta|x : A' \vdash I, J : A' \rightarrow A$. By coherence, we know $I =_\beta J$. Hence $I[N_2'/x] =_\beta J[P_2'/x]$ and $N_1'\ (I[N_2'/x]) =_\beta P_1'\ (J[P_2'/x])$.

5.4 Conservativity

To prove conservativity, we use induction loading.

Proposition 9 (Conservativity) *Assume $\Delta|\Gamma \vdash M : A$ is a derivable judgement. Then*

- *Γ, M and A have an $\epsilon(\Delta|\Gamma)$-normal form;*
- *for every $\epsilon(\Delta|\Gamma)$-normal forms Γ', M' and A' of Γ, M and A, the judgement $|\Gamma' \vdash M' : A'$ is derivable.*

The result is proved by induction on the derivations.

5.5 Decidability of type-checking

Definition 10 - *Let Δ be a set of pseudo-coercions. A pure type system has* decidable type-checking *for Δ if for every context $\Delta|\Gamma$ and pair of pseudo-terms (M, A), it is decidable if $\Delta|\Gamma \vdash M : A$ is derivable.*
- *A pure type system has decidable type-checking if it has decidable type-checking for all sets of pseudo-coercions.*
- *A pure type system has decidable type-checking for the standard syntax (STC) if it has decidable type-checking for the empty set of pseudo-coercions.*

The latter property is named so because derivation in the context without coercions correspond exactly to derivations in the standard syntax.

Lemma 11 *Assume $\Delta|\Gamma \vdash M : A$, $\Gamma' \twoheadrightarrow_{\epsilon(\Delta)} \Gamma$, $M' \twoheadrightarrow_{\epsilon(\Delta)} M$ and $A' \twoheadrightarrow_{\epsilon(\Delta)} A$. Then $\Delta|\Gamma' \vdash N' : A'$.*

We can advocate Proposition 9 and Lemma 11 to prove decidability of type-checking.

Proposition 12 *$\Delta|\Gamma \vdash M : A$ is derivable iff Δ is a coherent set of coercions and there exist Γ', M' and A' such that $|\Gamma' \vdash M' : A'$ is derivable, $\Gamma \twoheadrightarrow_{\epsilon(\Delta)} \Gamma'$, $M \twoheadrightarrow_{\epsilon(\Delta)} M'$ and $A \twoheadrightarrow_{\epsilon(\Delta)} A'$.*

One strategy to check whether $\Delta|\Gamma \vdash M : A$ is derivable is therefore to compute all possible legal $\epsilon(\Delta)$ reductions of Γ, M and A. This is achieved by defining for every term M its set $\mathsf{Exp}_{\Delta}(M)$ of possible explicitations of M relative to a set of pseudo-coercions Δ. In the sequel, we let $\Delta^{\bullet}$ be the smallest subset of Δ^{+} containing:

- all the pseudo-coercions $c : A \rightarrow B$ of Δ such that $A \neq_{\beta} B$;
- all the pseudo-coercions $\lambda x : A_1.c_n(c_{n-1}(\ldots(c_1 x)\ldots) : A_1 \rightarrow B_n$ where for $i = 1, \ldots, n$, $c_i : A_i \rightarrow B_i$ are pseudo-coercions in Δ and
  - $A_{i+1} =_{\beta} B_i$ for $i = 1, \ldots, n-1$,
  - $A_i \neq_{\beta} B_j$ for $i \leq j$.

In other words, $\Delta^\bullet$ is the set of pseudo-coercions which do not contain any loop. $\mathsf{Exp}_\Delta(M)$ is defined inductively on the structure of the terms:

$$\begin{aligned}
\mathsf{Exp}_\Delta(x) &= \{x\}\\
\mathsf{Exp}_\Delta(s) &= \{s\}\\
\mathsf{Exp}_\Delta(\Pi x : A.B) &= \{\Pi x : A'.B' \mid A' \in \mathsf{Exp}(A) \ \wedge \ B' \in \mathsf{Exp}_\Delta(B)\}\\
\mathsf{Exp}_\Delta(\Sigma x : A.B) &= \{\Sigma x : A'.B' \mid A' \in \mathsf{Exp}(A) \ \wedge \ B' \in \mathsf{Exp}_\Delta(B)\}\\
\mathsf{Exp}_\Delta(\lambda x : A.b) &= \{\lambda x : A'.b' \mid A' \in \mathsf{Exp}(A) \ \wedge \ b' \in \mathsf{Exp}_\Delta(b)\}\\
\mathsf{Exp}_\Delta(M\ N) &= \{M'(i\ N') \mid M' \in \mathsf{Exp}_\Delta(M),\ N' \in \mathsf{Exp}_\Delta(N) \text{ and } (i, A \to B) \in \Delta^\bullet\}\\
\mathsf{Exp}_\Delta(\langle a, b\rangle) &= \{\langle a', b'\rangle \mid a' \in \mathsf{Exp}(a) \ \wedge \ b' \in \mathsf{Exp}_\Delta(b)\}\\
\mathsf{Exp}_\Delta(\mathsf{fst}\ M) &= \{\mathsf{fst}\ M' \mid M' \in \mathsf{Exp}_\Delta(M)\}\\
\mathsf{Exp}_\Delta(\mathsf{snd}\ M) &= \{\mathsf{snd}\ M' \mid M' \in \mathsf{Exp}_\Delta(M)\}
\end{aligned}$$

where it is assumed that x is a variable and s is a sort. Note that $\mathsf{Exp}_\Delta(M)$ is finite and contains all the possible legal reducts of M.

Lemma 13 *$\Delta|\Gamma \vdash M : A$ is derivable iff Δ is a coherent set of coercions and $|\Gamma' \vdash M' : A'$ is derivable for some $\Gamma' \in \mathsf{Exp}_\Delta(\Gamma)$, $M' \in \mathsf{Exp}(M)$ and $A' \in \mathsf{Exp}_\Delta(A)$.*

We have a procedure to check whether a judgement with implicit coercions is derivable in provided that:

- STC holds;
- it is decidable whether a set Δ of coercions is coherent;
- the closure $\Delta^\bullet$ of a coherent set of coercions Δ can be computed effectively.

Note that the last two requirements are automatically fulfilled when the domains and codomains of the coercions are normalising.

Definition 14 *A pure type system with implicit coercions is standard strongly normalising (SSN) if for every derivable judgement $|\Gamma \vdash M : A$, the term M is strongly normalising w.r.t. $\to_\beta$.*

We have:

Theorem 15 *A pure type system with implicit coercions has decidable type-checking (and type-synthesis) if it has STC and SSN.*

In [17], L.S. van Benthem Jutting has proved that type-checking and type-synthesis are decidable for a normalising pure type system with finitely many sorts. It follows:

Corollary 16 *The systems of the λ-cube with implicit coercions have decidable type-checking and type-synthesis.*

6 Possible extensions and related work

In this section, we put our work into a more general perspective by looking at some related work. We also discuss the possibility of using implicit coercions to define subtyping.

6.1 Implicit coercions with principal typing

P. Aczel and A. Bailey have recently suggested an alternative approach to implicit coercions. Their approach is based on type systems with type-casting and principal types such as the type system of Lego. Principal types are crucially used in the method rule: if $i : A' \to A$ is a coercion, $f : A \to B$ and $a :: A'$ (where $::$ denotes principal typing), then $f\ a : B$. If $a : A$ but not $a :: A'$, then $f\ a$ will not be legal. However, one will be able to type-cast a and apply it to f. In other words, $f\ (a : A) : B$. In our view, their approach is extremely syntactic and does not fully reflect the mathematical intuition behind the use of implicit coercions. However their approach has the considerable advantage to yield a simple and efficient type-checking algorithm.

6.2 Records

G. Betarte and A. Tasistro have recently provided an alternative solution to the problem of implicit syntax based on dependent records ([5, 6]). Roughly speaking, record types correspond to Σ-types and coercions correspond to projections. The specific structure of the coercions has the pleasant effect to simplify the coherence problem and to allow for coercions between records with free variables. Moreover, the problem of conversion seems to disappear. Because of the obvious advantages of their approach, it would be interesting whether their results can be carried over to the framework of pure type systems.

6.3 Classes

The original motivation for our work was to enhance proof-checkers with a notion similar to that of type class as it is used in Gofer ([11, 12]) or Haskell ([10]). Although our work shares many motivations with type classes as developed in these languages, the actual formalisms of type classes and implicit coercions are quite distinct. It makes it difficult to compare formalisms.

6.4 Subtyping

The type system for implicit coercions remains strongly typed in the sense that every term has at most one type (provided the pure type system is functional). One might consider replacing the Method rule with a subsumption rule

$$\frac{\Delta|\Gamma \vdash M : A}{\Delta|\Gamma \vdash M : A'} \quad \text{if } (t : A \to A') \in \Delta$$

and redefine $\epsilon(\Delta)$-reduction as the compatible closure of $t \rightarrow_{\epsilon(\Delta)} it$ if $i \in \Delta^{+}$. In this way, one would obtain type systems with subtyping. It would be interesting to see whether the coherence and conservativity conditions hold for this new syntax.

6.5 Implementation

This work originates from previous work with Peter Aczel on formalising Galois theory in Lego. In absence of a mechanism to handle multiple inheritance, we realised that the syntax was becoming too heavy and the number of identifiers was becoming disproportionate very rapidly. This led us to consider the possibility of implementing implicit coercions in Lego; this was done by September 1993. However, this implementation only supports single inheritance. It would be nice to have an implementation of the syntax proposed in this paper.

7 Conclusion

In this paper, we have presented a modified syntax for pure type systems which allows for a uniform treatment of implicit coercions. The syntax enjoys some important properties and has proved useful in the formalisation of mathematics in Lego ([4]). However, the syntax also suffers from some severe limitations. Future research should concentrate on the possility of overcoming some of these limitations, especially the one to simple coercions.

In the longer term, it seems important to understand the interaction between inheritance, subtyping and argument synthesis in order to be able to bring the flexibility of expression in formal systems close to the one of informal mathematics. Such a program, if completed, would constitute a definite step towards feasible formal mathematics.

References

1. P. Aczel. A notion of class for type theory. Note, 1995.
2. D. Aspinall and A. Compagnoni. Subtyping dependent types. In *Proceedings of LICS'96.* IEEE Computer Society Press, 1996. To appear.
3. A. Bailey. Lego with classes. Note, 1995.
4. G. Barthe. Formalising algebra in type theory: fundamentals and applications to group theory. Manuscript. An earlier version appeared as technical report CSI-R9508, University of Nijmegen, under the title 'Formalising mathematics in type theory: fundamentals and case studies', 1995.
5. G. Betarte and A. Tasistro. Extension of Martin-Löf's theory of types with record types and subtyping: motivation, rules and type checking. Manuscript, 1995.
6. G. Betarte and A. Tasistro. Formalisation of systems of algebras using dependent record types and subtyping: an example. Manuscript, 1995.
7. R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, J.T. Sasaki, and S.F. Smith. *Implementing Mathematics with the NuPrl Development System.* Prentice-Hall, inc., Englewood Cliffs, New Jersey, first edition, 1986.

8. N.G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In R. Nederpelt, H. Geuvers, and R. de Vrijer, editors, *Selected papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 865–935. North-Holland, Amsterdam, 1994.
9. C.A. Gunter and J.C. Mitchell. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics and Language Design*. The MIT Press, 1994.
10. P. Hudak, S.L. Peyton Jones, P.L. Wadler, Arvind, B. Boutel, J. Fairbairn, J. Fasel, K. Guzman, K. Hammond, J. Hughes, T. Johnsson, R. Kieburtz, R.S. Nikhil, W. Partain, and J. Peterson. Report on the functional programming language Haskell, version 1.2. *Special Issue of SIGPLAN Notices*, 27, 1992.
11. M. Jones. Introduction to Gofer. Included as part of the Gofer distribution. Available by anonymous `ftp` from `nebula.cs.yale.edu` in the directory `pub/haskell/gofer`, 1991.
12. M. Jones. A system of constructor classes: overloading and implicit higher-order polymorphism. *Journal of Functional Programming*, pages 1–25, January 1995.
13. Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*. Number 11 in International Series of Monographs on Computer Science. Oxford University Press, 1994.
14. Z. Luo. Coercive subtyping. Draft, 1995.
15. F. Pfenning. Refinement types for logical frameworks. In H. Geuvers, editor, *Informal Proceedings of TYPES'93*, pages 285–299, 1993. Available from http://www.dcs.ed.ac.uk/lfcsinfo/research/types-bra/proc/index.html.
16. R. Pollack. Implicit syntax. In G. Huet and G. Plotkin, editors, *Informal Proceedings of First Workshop on Logical Frameworks, Antibes*, May 1990.
17. L. S. van Benthem Jutting. Typing in pure type systems. *Information and Computation*, 105(1):30–41, July 1993.
18. P. Wadler and S. Blott. How to make ad hoc polymorphism less ad hoc. In *Proceedings of POPL'89*, pages 60–76. ACM Press, 1989.
19. A. Wikström. *Functional Progrmmaming using Standard ML*. Interntional Series in Computer Science. Prenctice Hall, 1987.