A note on an implicit calculus with annotated terms: introducing universal dependent types.

Lionel Vaux

21st June 2004

Abstract

This note presents a decidable variant of the IC calculus described in [Vau04]. It was first designed by Thierry Coquand for the purpose of giving a theoretical grounding to a possible implementation of IC, but no formal result had been published yet. This note tries to overcome this lack.

Introduction

One of the possible purposes in designing type systems is to obtain logical frameworks, *i.e.* type systems in which one can encode formulas and proofs of some logical systems as canonical terms in well chosen types (contrary to the proofs-as-programs paradigm, in which formulas are interpreted by types and proofs by terms); see *e.g.* [HHP87], [NL98] and [CPT]. Most logical frameworks are close to original (Edinburgh) LF defined in [HHP87]. In particular, they use types built up with a dependent product constructor $\Pi x{:}A.B$ and rules similar to:

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x M : \Pi x{:}A.B}\ (\textsc{Abs}) \qquad \frac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B(N/x)}\ (\textsc{App})$$

Notice however that in rule (App), term N is very often used for typing purpose only, and not as a "true" argument of M: typically, polymorphic identity id has type $\Pi A{:}\mathsf{Set}.(A \to A)$, and when applying it to a term M in type A, one writes $((idA)M)$.

One could think of a type system closer to system F or the ML family, in the sense that id should have type $A \to A$ for every type A. One such system is Miquel's Implicit Calculus of Constructions (ICC) (see [Miq01b] or [Miq01a]). Miquel introduces a new dependent product $\cap x{:}A.B$ called implicit product, that can roughly be thought of as an intersection, with rules:

$$\frac{\Gamma, x : A \vdash M : B \quad x \notin FV(M)}{\Gamma \vdash M : \cap x{:}A.B}\ (\textsc{Gen}) \qquad \frac{\Gamma \vdash M : \cap x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M : B(N/x)}\ (\textsc{Inst})$$

In this system, λxx has type $\cap A{:}\mathsf{Set}.(A \to A)$, hence by rule (Inst), it has type $A \to A$ for every type A in Set. This type system however, is deeply related to Coquand's Calculus of Constructions, and as such, not so close to LF. Moreover, it is not decidable.

A “simplification” of ICC, closer to a logical framework and Martin-Löf’s type theory, has been presented in [Vau04]. This Implicit Calculus (IC) has very simple models and it’s typing rules are implementation-oriented: sub-typing is explicitly part of the typing rules and a type-checking algorithm can be implemented: given term M and type A, first infer a “minimal” type for M, then check if it is a sub-type of A. However, IC is undecidable and this implementation can only be partial.

In this note, a new decidable calculus is presented, that is closely related to IC. The idea is to annotate terms of IC (which are pure λ-terms) with that information which is necessary to make typing decidable. We add two new constructions on terms: generalization ($\nu x M$) and specialization ($M\langle N\rangle$). One should consider that $\nu x M$ is another version of M that has been generalized so as to be typed in an implicit product, and that $M\langle N\rangle$ is a version of M whose (implicit) type has been been intanciated by N. More precisely, we give new versions of (Gen) and (Inst):

$$\frac{\Gamma, x : A \vdash M : B \quad x \notin FV(|M|)}{\Gamma \vdash \nu x M : \cap x{:}A.B}\ (\textsc{Gen}) \quad \frac{\Gamma \vdash M : \cap x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M\langle N\rangle : B(N/x)}\ (\textsc{Spe})$$

Hence product $\cap x{:}A.B$ is no more really implicit, but should rather be thought of as universal: if $N : A$, one can annotate any term of type $\cap x{:}A.B$ by N so as to give it type $B(N/x)$; conversely if M can be annotated to have type B when $x : A$, but M itself does not depends on x, then M can be universally typed $\cap x{:}A.B$. Notice that the variable which is used to generalize M in $\nu x M$ must only appear in annotations of M, not in M itself: it is a quantification on possible annotations. Hence the $|M|$ notation: $|\cdot|$ is a forget map that just deletes annotations.

The description of this Universal Calculus (UC) follows.

1 Syntax

1.1 Terms

Besides the usual variable (x), abstraction ($\lambda x M$) and application (MN) constructions on terms, we introduce generalization of term M by variable x ($\nu x M$) and specialization of term M by term N ($M\langle N\rangle$). Generalization and specialization can be thinked of as annotations of pure λ-terms for typing purposes, with generalization $\nu x M$ binding variables only in specializations appearing in M. Hence, for term $\nu x M$ to be acceptable, x must not appear in M, except in specialization subterms. A more precise definition follows.

Define pre-terms by:

$$M ::= x \mid \lambda x M \mid MM \mid \nu x M \mid M\langle M\rangle$$

Define free variables in pre-terms as usual, with binders λ and ν. Denote $FV(M)$ the set of free variables in M. Define the forget map on pre-terms $|\cdot|$ by:

$$\begin{aligned} |x| &= x \\ |\lambda x M| &= \lambda x |M| \\ |MN| &= |M||N| \\ |\nu x M| &= |M| \\ |M\langle N\rangle| &= |M| \end{aligned}$$

Define the set Λ_U of terms by:

$$\begin{array}{rl} x \in \Lambda_U & \\ \lambda x M \in \Lambda_U & \text{if } M \in \Lambda_U \\ MN \in \Lambda_U & \text{if } M, N \in \Lambda_U \\ \nu x M \in \Lambda_U & \text{if } M \in \Lambda_U \wedge x \notin FV(|M|) \\ M\langle N \rangle \in \Lambda_U & \text{if } M, N \in \Lambda_U \end{array}$$

If M is a term, $|M|$ is the underlying pure λ-term of M.

1.2 Types

Build types using a fixed sort-type (Set), a type constructor on terms ($\mathsf{El}M$), the usual dependent product ($\Pi x{:}A.B$) — here called explicit product — and another dependent product ($\cap x{:}A.B$) — called universal product. Define types by:

$$A ::= \mathsf{Set} \mid \mathsf{El}M \mid \Pi x{:}A.A \mid \cap x{:}A.A$$

Define free variables in pre-types as usual, with binders Π and $\cap$, free variables in $\mathsf{El}M$ being those in M. Define the forget map on pre-types $|\cdot|$ by:

$$\begin{array}{rcl} |\mathsf{Set}| & = & \mathsf{Set} \\ |\mathsf{El}M| & = & \mathsf{El}|M| \\ |\Pi x{:}A.B| & = & \Pi x{:}|A|.|B| \\ |\cap x{:}A.B| & = & \cap x{:}|A|.|B| \end{array}$$

Define the set $\mathcal{T}_U$ of types by:

$$\begin{array}{rl} \mathsf{Set} \in \mathcal{T}_U & \\ \mathsf{El}M \in \mathcal{T}_U & \text{if } M \in \Lambda_U \\ \Pi x{:}A.B \in \mathcal{T}_U & \text{if } A, B \in \mathcal{T}_U \\ \cap x{:}A.B \in \mathcal{T}_U & \text{if } A, B \in \mathcal{T}_U \end{array}$$

Remark Types are syntactically distinct from terms (hence the El construction), and will not be typable themselves (see typing rules in figure 1).

1.3 Judgements

Define pre-contexts by:

$$\Gamma ::= \diamond \mid \Gamma, x : A$$

Define free and declared variables in pre-contexts as follows. Denoting $FV(\Gamma)$ the set of free variables in Γ and $DV(\Gamma)$ that of the variables declared in Γ:

$$\begin{array}{rcl} FV(\diamond) = DV(\diamond) & = & \emptyset \\ FV(\Gamma, x : A) & = & FV(\Gamma) \cup (FV(A) \setminus DV(\Gamma)) \\ DV(\Gamma, x : A) & = & DV(\Gamma) \cup \{x\} \end{array}$$

Define the set $\mathcal{C}_U$ of contexts by:

$$\begin{array}{rl} \diamond \in \mathcal{C}_U & \\ \Gamma, x : A \in \mathcal{C}_U & \text{if } A \in \mathcal{T}_U \wedge x \notin DV(\Gamma) \end{array}$$

There are three forms of judgements: validity of context ($\Gamma \vdash \cdot$), validity of type ($\Gamma \vdash A$) and typing ($\Gamma \vdash M : A$). Define pre-assertions by:

$$\alpha ::= \cdot \mid A \mid M : A$$

and pre-judgements by:

$$J ::= \Gamma \vdash \alpha.$$

Give the obvious definition to free variables in pre-assertions, and pre-judgements — contexts bind the variables they declare. Define the set of assertions $\mathcal{A}_U$ by:

$$\begin{array}{rl} \cdot \in \mathcal{A}_U & \\ A \in \mathcal{A}_U & \text{if } A \in \mathcal{T}_U \\ M : A \in \mathcal{A}_U & \text{if } M \in \Lambda_U \wedge A \in \mathcal{T}_U \end{array}$$

and that of judgements ($\mathcal{J}_U$) by:

$$\Gamma \vdash \alpha \in \mathcal{J}_U \text{ if } \Gamma \in \mathcal{C}_U \wedge \alpha \in \mathcal{A}_U$$

1.4 Equalities

Equality on (pre-)terms, (pre-)types and (pre-)judgements is α-convertibility, with binders λ, ν, Π, $\cap$ and $\vdash$ (*i.e.* α-conversion can involve renaming of defined variables).

Write $M(N/x)$ (resp. $A(N/x)$, $\Gamma(N/x)$, $\alpha(N/x)$ and $J(N/x)$) for capture avoiding substitution of u for x in pre-term M (resp. pre-type A, pre-context Γ, pre-assertion α and pre-judgement J).

$\beta\eta$-convertibility ($\equiv_{\beta\eta}$) is defined on pure λ-terms as usual. It is not extended to general pre-terms or terms. Convertibility is easily extended to those types, contexts, assertions and judgements that involve only pure λ-terms.

2 Typing

Typing rules are given in figure 1. Derivations are trees built on these rules. Pre-judgement J "holds" if it is the conclusion of a derivation.

2.1 Typing terms in a universal product

The main two typing rules for universal product are (Gen) and (Spe) which are similar to (Abs) and (App) for explicit product. We can interpret both as defining universal product as type intersection: a term is in an intersection iff it is in every type of the intersection. Here generalization ($\nu x M$) and specialization ($M\langle N\rangle$) are used as typing annotations: notice that the underlying λ-term is preserved by (Gen) and (Spe).

2.2 Well-formedness

Lemma 1 *If pre-judgement $\Gamma \vdash \alpha$ holds, then it is a judgement, and $FV(\alpha) \subset DV(\Gamma)$.*

Proof. Easy proof by induction on derivations. □

Contexts:

$$\frac{}{\diamond \vdash}\ (\text{WF-E}) \quad \frac{\Gamma \vdash A \quad x \notin DV(\Gamma)}{\Gamma, x : A \vdash}\ (\text{WF-C})$$

Types:

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Set}}\ (\textsc{Set}) \quad \frac{\Gamma \vdash M : \mathsf{Set}}{\Gamma \vdash \mathsf{El}M}\ (\textsc{El})$$

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash \Pi x{:}A.B}\ (\textsc{Exp}) \quad \frac{\Gamma, x : A \vdash B}{\Gamma \vdash \cap x{:}A.B}\ (\textsc{Uni})$$

Terms:

$$\frac{\Gamma \vdash \quad (x : A) \in \Gamma}{\Gamma \vdash x : A}\ (\textsc{Var})$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x M : \Pi x{:}A.B}\ (\textsc{Abs}) \quad \frac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B(N/x)}\ (\textsc{App})$$

$$\frac{\Gamma, x : A \vdash M : B \quad x \notin FV(|M|)}{\Gamma \vdash \nu x M : \cap x{:}A.B}\ (\textsc{Gen}) \quad \frac{\Gamma \vdash M : \cap x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M\langle N\rangle : B(N/x)}\ (\textsc{Spe})$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B \quad |A| \equiv_{\beta\eta} |B|}{\Gamma \vdash M : B}\ (\textsc{Conv})$$

Figure 1: Typing rules

3 General properties of the calculus

The following lemma is easily proved by induction on derivations. It is useful in subsequent proofs by induction because it allows to apply induction hypothesis to other sub-trees than the immediate sub-derivations.

Lemma 2 *Every derivation of* $\Gamma \vdash \alpha$ *has a node* Γ. *Every derivation of* $\Gamma, x : A \vdash \alpha$ *has a node* $\Gamma \vdash A$.

The following three lemmas are proved by induction on derivations, using lemma 2.

Lemma 3 *If* $\Gamma \subset \Gamma'$, *and* $\Gamma \vdash \alpha$ *and* $\Gamma' \vdash$ *hold, then* $\Gamma' \vdash \alpha$ *holds.*

Lemma 4 *If* $\Gamma_1, x : A, \Gamma_2 \vdash \alpha$ *and* $\Gamma_1 \vdash N : A$ *hold, then* $\Gamma_1, \Gamma_2(N/x) \vdash \alpha(N/x)$ *holds.*

Lemma 5 *If* $\Gamma \vdash M : A$ *holds, then* $\Gamma \vdash A$ *holds.*

The following trivial lemma allows proofs by induction on couples of derivations.

Lemma 6 *Denote* $h(\mathfrak{T})$ *the height of derivation* $\mathfrak{T}$. *Define* $\sqsubset$ *on couples of derivations by:* $(\mathfrak{T}_1, \mathfrak{T}_2) \sqsubset (\mathfrak{T}'_1, \mathfrak{T}'_2)$ *iff either* $h(\mathfrak{T}_1) < h(\mathfrak{T}'_1)$ *and* $h(\mathfrak{T}_2) \leq h(\mathfrak{T}'_2)$ *or* $h(\mathfrak{T}_1) \leq h(\mathfrak{T}'_1)$ *and* $h(\mathfrak{T}_2) < h(\mathfrak{T}'_2)$. *Then the set of couples of derivations is well-founded for* $\subset$.

Lemma 7 *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash M : A'$ *hold, then* $|A| \equiv_{\beta\eta} |A'|$.

This is obviously proved by well founded induction on couples of derivations, in accordance with lemma 6.

Lemma 8 *If $\Gamma \vdash \alpha$ has a derivation $\mathfrak{T}$, $\Gamma' \vdash$ holds, and $|\Gamma| \equiv_{\beta\eta} |\Gamma'|$, then $\Gamma' \vdash \alpha$ holds. Moreover there is a derivation of $\Gamma' \vdash \alpha$ with height $h(\mathfrak{T})$.*

This is proved by induction on derivations of $\Gamma \vdash \alpha$.

Lemma 9 *If $\Gamma \vdash \lambda x M : \Pi x{:}A.B$ or $\Gamma \vdash \nu x M : \cap x{:}A.B$ holds, then $\Gamma, x : A \vdash M : B$ holds.*

This is proved by induction on derivations, using lemma 8 for rule (Conv).

4 UC and IC

Recall in figure 2 the rules of the implicit calculus (here called IC) defined in [Vau04]. The types of IC are the same as those of UC, but IC terms are pure λ-terms. Informally, a term in UC is a term in IC with annotations for typing implicit/universal products. This is more precisely stated by the following theorem, which was the main purpose of designing UC.

Theorem 10 *If judgement J holds in UC then $|J|$ holds in IC.*

Proof. In accordance with lemma 6, prove by induction on couples of UC derivations that:

- if the conclusions are J and J' then $|J|$ and $|J'|$ hold in IC;
- if $J = \Gamma \vdash A$, $J' = \Gamma \vdash A'$ and $|A| \equiv_{\beta\eta} |A'|$ then $|\Gamma| \vdash |A| \leq |A'|$ and $|\Gamma| \vdash |A'| \leq |A|$.

Here we only detail the case ((Uni),(Uni)). The root of the first derivation is of the form

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash \cap x{:}A.B} \text{ (Uni)}$$

By induction hypothesis, $|\Gamma|, x : |A| \vdash |B|$ holds in IC, hence by IC rule (Imp), $|\Gamma| \vdash \cap x{:}|A|.|B|$ holds too. The same applies to the second derivation. Assume moreover that the second derivation root is of the form:

$$\frac{\Gamma, x : A' \vdash B'}{\Gamma \vdash \cap x{:}A'.B'} \text{ (Uni)}$$

with $A \equiv_{\beta\eta} A'$ and $B \equiv_{\beta\eta} B'$. By lemmas 2 and 8, we can apply the induction hypothesis to derivations of $\Gamma, x : A \vdash B$ and $\Gamma, x : A \vdash B'$; hence

$$|\Gamma|, x : |A| \vdash |B| \leq |B'|$$

holds in IC. Moreover $|\Gamma|, x : |A| \vdash x : |A|$ and

$$|\Gamma|, x : |A|, y : |A'| \vdash |B'|(y/x)$$

Contexts:

$$\frac{}{\diamond \vdash}\ (\text{WF-E}) \quad \frac{\Gamma \vdash A \quad x \notin DV(\Gamma)}{\Gamma, x : A \vdash}\ (\text{WF-C})$$

Types:

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Set}}\ (\textsc{Set}) \quad \frac{\Gamma \vdash M : \mathsf{Set}}{\Gamma \vdash \mathsf{El} M}\ (\textsc{El})$$

$$\frac{\Gamma, x : A \vdash B}{\Gamma \vdash \Pi x{:}A.B}\ (\textsc{Exp}) \quad \frac{\Gamma, x : A \vdash B}{\Gamma \vdash \cap x{:}A.B}\ (\textsc{Imp})$$

Terms:

$$\frac{\Gamma \vdash \quad (x : A) \in \Gamma}{\Gamma \vdash x : A}\ (\textsc{Var})$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x M : \Pi x{:}A.B}\ (\textsc{Abs}) \quad \frac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B(N/x)}\ (\textsc{App})$$

$$\frac{\Gamma, x : A \vdash M : B \quad x \notin FV(M)}{\Gamma \vdash M : \cap x{:}A.B}\ (\textsc{Gen}) \quad \frac{\Gamma \vdash M : \cap x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M : B(N/x)}\ (\textsc{Inst})$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A \leq B}{\Gamma \vdash M : B}\ (\textsc{Cum})$$

Sub-typing:

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Set} \leq \mathsf{Set}}\ (\text{S-}\textsc{Set})$$

$$\frac{\Gamma \vdash M : \mathsf{Set} \quad \Gamma \vdash N : \mathsf{Set} \quad M \equiv_{\beta\eta} N}{\Gamma \vdash \mathsf{El} M \leq \mathsf{El} N}\ (\text{S-}\textsc{El})$$

$$\frac{\Gamma \vdash A_2 \leq A_1 \quad \Gamma, x : A_2 \vdash B_1 \leq B_2 \quad \Gamma, x : A_1 \vdash B_1}{\Gamma \vdash \Pi x{:}A_1.B_1 \leq \Pi x{:}A_2.B_2}\ (\text{S-}\textsc{Exp})$$

$$\frac{\Gamma, x : A \vdash C \leq B \quad x \notin FV(C)}{\Gamma \vdash C \leq \cap x{:}A.B}\ (\text{S-}\textsc{Imp})$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B(M/x) \leq C \quad \Gamma, x : A \vdash B}{\Gamma \vdash \cap x{:}A.B \leq C}\ (\textsc{Imp}\text{-S})$$

Strengthening:

$$\frac{\Gamma, x : A \vdash B \quad x \notin FV(B)}{\Gamma \vdash B}\ (\textsc{Str-Type})$$

$$\frac{\Gamma, x : A \vdash M : B \quad x \notin FV(M) \cup FV(B)}{\Gamma \vdash M : B}\ (\textsc{Str-Term})$$

$$\frac{\Gamma, x : A \vdash B \leq C \quad x \notin FV(B) \cup FV(C)}{\Gamma \vdash B \leq C}\ (\textsc{Str-Sub})$$

Figure 2: IC typing rules

with $y \notin DV(\Gamma) \cup \{x\}$ both hold in IC too (by induction hypothesis and (Var) rule and weakening property in IC). Hence by IC rule (Imp-S), and up to α-conversion,

$$|\Gamma|, x : |A| \vdash \cap x{:}|A'|.|B'| \leq |B|$$

holds in IC. Since $x \notin FV(\cap x{:}|A'|.|B'|)$, rule (S-Imp) gives

$$|\Gamma| \vdash \cap x{:}|A'|.|B'| \leq \cap x{:}|A|.|B|$$

□

Conclusions

It is clear that type checking in UC is decidable. Hence we have a decidable type system that translates easily to IC and this translation preserves derivability.

By designing UC, we had two main purposes in mind: build a logical framework close to IC and give some theoretic grounding to an implementation of IC that could override undecidablity. Truth is that those goals remain to be achieved but we've gone a step further.

First, syntactic properties of UC are easy to establish, and close enough to those used by the authors of [HHP87] that it shouldn't prove hard to adapt their results to UC.

Second, one could easily adapt a partial implementation of IC to UC so that annotations would be built on the fly by the type checker (in the spirit of [NL98]) and only those that could not be guessed by the algorithm would be asked to the user; then UC terms would be saved with user-provided annotations unchanged and place-holders in place of those that could be infered. Then it is only a matter of deciding which presentation is provided to the user: M or $|M|$.

References

[CPT] Thierry Coquand, Randy Pollack, and Makoto Takeyama. A logical framework with dependently typed records. `http://www.cs.chalmers.se/~coquand/MODULE/main.html`.

[HHP87] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *Proceedings 2nd Annual IEEE Symp. on Logic in Computer Science, LICS'87, Ithaca, NY, USA, 22–25 June 1987*, pages 194–204. IEEE Computer Society Press, New York, 1987.

[Miq01a] Alexandre Miquel. The implicit calculus of constructions: Extending pure type systems with an intersection type binder and subtyping. In S. Abramsky, editor, *Proc. of 5th Int. Conf. on Typed Lambda Calculi and Applications, TLCA'01, Krakow, Poland, 2–5 May 2001*, volume 2044, pages 344–359. Springer-Verlag, Berlin, 2001.

[Miq01b] Alexandre Miquel. *Le Calcul des Constructions Implicite : syntaxe et sémantique*. PhD thesis, Université Paris 7, dec. 2001. `http://pauillac.inria.fr/~miquel/index.html`.

[NL98] George C. Necula and Peter Lee. Efficient representation and validation of proofs. In *Logic in Computer Science*, pages 93–104, 1998.

[Vau04] Lionel Vaux. A type system with implicit types. `http://perso.ens-lyon.fr/lionel.vaux/`, 2004.