Y is not typable in λU

Herman Geuvers[1,2,*] and Joep Verkoelen

[1] iCIS, Radboud University Nijmegen, The Netherlands
[2] Technical University Eindhoven, The Netherlands

Abstract

The type theories λU and λU^- are known to be logically inconsistent. For λU, this is known as Girard's paradox [7]; for λU^- the inconsistency was proved by Coquand [3]. It is also known that the inconsistency gives rise to a so called *looping combinator*: a family of terms L_n such that $L_n f$ is convertible with $f(L_{n+1} f)$. It is un-known whether a fixed point combinator exists in these systems. Hurkens [9] has given a simpler version of the paradox in λU^-, giving rise to an actual proof term that can be analyzed, and which is proven to be a looping combinator and not a fixed point combinator in [2]. However, the underlying untyped term is a real fixed point combinator.

Here we analyze the possibility of typing a fixed point combinator in λU and we prove that the Curry and Turing fixed point combinators Y and Θ cannot be typed in λU, and the same holds for Ω.

Although systems like $\lambda\star$ and λU are logically inconsistent, computationally they are still interesting, because not all terms are β-convertible. The first to study the computational power of these inconsistent systems was Howe [8], going back to earlier (unpublished) work of [10]. Howe coined the terminology *looping combinator* for a family of terms $\{L_n\}_{n\in\mathbb{N}}$ such that $L_n\, f =_\beta f(L_{n+1}\, f)$, and he showed that a looping combinator can be defined in $\lambda\star$. Using a looping combinator, it can be shown that the equational theory (the theory of β-conversion) is undecidable and that the theory is Turing complete.

When Girard [7] proved the paradox in 1972, he did that for λU, an extension of higher order logic with polymorphic domains and quantification over all domains. This system allows less type constructions than $\lambda\star$, but that has the advantage that it is somewhat easier to see what is going on. By that time, it was unclear whether λU^-: higher order logic with polymorphic domains (but no quantification over all domains) was inconsistent.

In 1994, Coquand [3] proved that λU^- is inconsistent as well, by encoding Reynolds' result [11], stating that no set-theoretic model of polymorphic lambda calculus exists, into λU^-. Later, Hurkens gave a considerably shorter proof [9], which is based on interpreting Russell's paradox in λU^-. Recently, Coquand [4] has given an adapted presentation of Hurkens' proof, emphasizing the relation with Reynolds' result.

Here we analyze the paradox in λU syntactically. (For a semantic analysis, relating the paradox to models of higher order logic, see [6].) The main question we are interested in is whether there exists a fixed-point combinator in λU. We give a partial answer by showing that the well-known Turing and Curry fixed-point combinators (Θ and Y) cannot be typed in λU.

We assume λU to be known (see [1, 5]), so we don't give the typing rules but we just emphasize that we divide the set of variables $\mathcal{V}$ into three disjoint sets $\mathrm{var}^\triangle$, $\mathrm{var}^\square$ and $\mathrm{var}^\star$ for which we use standard characters: $\mathrm{var}^\triangle = \{k_1, k_2, k_3, \ldots\}$, $\mathrm{var}^\square = \{\alpha, \beta, \gamma, \ldots\}$, $\mathrm{var}^\star = \{x, y, z, \ldots\}$. So a variable that lives in a type $A : \star$ is typically x, y or z etcetera. We also define the syntactical

*herman@cs.ru.nl

categories *Kinds* (K_1, K_2, K_3), *Constructors* (P, Q, R) and *Proof terms* (t, p, q) as follows.

Kinds	K	$::=$	$k \mid \star \mid K \to K \mid \Pi k : \Box . K$
Constructors	P	$::=$	$\alpha \mid \lambda\alpha : K.P \mid PP \mid P \to P \mid \lambda k : \Box . P \mid PK \mid \Pi\alpha : K.P$
Proof terms	t	$::=$	$x \mid \lambda x : P.t \mid tt \mid \lambda\alpha : K.t \mid tP \mid \lambda k : \Box . p \mid pK$

An important property of λU (which is not the case in $\lambda\star$) is that

Lemma 1. *All kinds and constructors of λU are strongly normalizing.*

Therefore, type checking is decidable in λU. For t a proof term of λU, we define the *erasure* of t, denoted by $|t|$, as follows, by induction on the construction of proof terms.

$\lvert x \rvert$	$=$	x					
$\lvert \lambda x : P.p \rvert$	$=$	$\lambda x. \lvert p \rvert$	if $P \in$ Constructors	$\lvert pq \rvert$	$=$	$\lvert p \rvert \lvert q \rvert$	if $p, q \in$ Proof terms
$\lvert \lambda\alpha : K.p \rvert$	$=$	$\lvert p \rvert$	if $K \in$ Kinds	$\lvert pP \rvert$	$=$	$\lvert p \rvert$	if $P \in$ Constructors
$\lvert \lambda k : \Box . p \rvert$	$=$	$\lvert p \rvert$		$\lvert pK \rvert$	$=$	$\lvert p \rvert$	if $K \in$ Kinds

We say that an untyped lambda term M is *typable in λU* iff there exist Γ, t, A such that $\Gamma \vdash t : A : \star$ and $|t| = M$. We prove the following result

Proposition 1. *The terms Ω, Y and Θ are not typable in λU.*

This result comes as a corollary of a more general result:

Theorem 2. Double self-application *is not possible in λU.*
Here we mean with "double self-application" a term $t : A : \star$ such that $|t| = (\lambda x.N)(\lambda y.P)$ and N contains a sub-term $x\,x$ and P contains a sub-term $y\,y$.

The Theorem is proving by analyzing the so called *parse tree* of a type, following ideas from [12]. The argument basically consists of two parts:

1. if t contains a self-application, so $|t|$ contains a sub-term $x\,x$, then the type of x in t is of the form $\Pi\vec{v} : \vec{V}.\alpha\vec{T} \to \ldots$ with $\alpha \in \vec{v}$;

2. if $|q| = \lambda y.N$ where N contains $y\,y$, then the type of q is not of the $\Pi\vec{v} : \vec{V}.\alpha\vec{T} \to \ldots$ with $\alpha \in \vec{v}$.

From this the Theorem follows.

If we now look back at the looping combinator L_0 that can be derived from the inconsistency proof of Hurkens [9], and we erase all type information, we obtain the following term.

$$|L_i| = L = \lambda f.(\lambda x.x(\lambda pq.f(qpq))x)(\lambda y.yy)$$

In the untyped λ-calculus, this is a fixed-point combinator and an interesting one, because it contains no *double self-application*, as Ω, Y and Θ do.

References

[1] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992.

[2] G. Barthe and Th. Coquand. Remarks on the equational theory of non-normalizing pure type systems. *Journal of Functional Programming*, 16(2):137–155, 2006.

[3] Th. Coquand. A new paradox in type theory. In *Logic, Methodology and Philosophy of Science IX: Proc. Ninth Int. Congress of Logic, Methodology, and Philosophy of Science*, pages 7–14. Elsevier, 1994.

[4] Thierry Coquand. A variation of Reynolds-Hurkens paradox. In Venanzio Capretta, Robbert Krebbers, and Freek Wiedijk, editors, *Logics and Type Systems in Theory and Practice - Essays Dedicated to Herman Geuvers on The Occasion of His 60th Birthday*, volume 14560 of *Lecture Notes in Computer Science*, pages 111–117. Springer, 2024.

[5] H. Geuvers. *Logics and Type Systems.* PhD thesis, Radboud University, Nijmegen, 1993.

[6] H. Geuvers. (In)consistency of extensions of higher order logic and type theory. In Th. Altenkirch and C. McBride, editors, *Types for Proofs and Programs Int. Workshop, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, volume 4502 of *LNCS*, pages 140–159. Springer, 2007.

[7] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

[8] D. J. Howe. The computational behaviour of Girard's paradox. In *Proceedings of the 2nd Symposium on Logic in Computer Science*, pages 205–214. IEEE, 1987.

[9] A.J.C. Hurkens. A simplification of Girard's paradox. In *TLCA '95: Proceedings of the 2nd Int. Conf. on Typed Lambda Calculi and Applications*, pages 266–278, London, UK, 1995. Springer.

[10] M.B. Reinhold. Typechecking is undecidable when 'type' is a type, 1986.

[11] J.C. Reynolds. Polymorphism is not set-theoretic. In G. Kahn, D.B. MacQueen, and G.D. Plotkin, editors, *Semantics of Data Types, International Symposium, Sophia-Antipolis, France, June 27-29, 1984, Proceedings*, volume 173 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 1984.

[12] P. Urzyczyn. Type reconstruction in Fω. *Mathematical Structures in Comp. Sci.*, 7(4):329–358, 1997.