Y is not typable in λU and neither are Θ, Ω

Herman Geuvers

Radboud University Nijmegen & TUE
jww Joep Verkoelen

June 12, 2025
TYPES
Glasgow

Why fixed-point combinators?

In untyped λ-calculus, a fixed-point combinator F gives you a fixed point of every term M

$$F\,M \quad =_\beta M\,(F\,M).$$

Why fixed-point combinators?

In untyped λ-calculus, a fixed-point combinator F gives you a fixed point of every term M

$$F\,M \quad =_\beta M\,(F\,M).$$

Why is this useful?
Solve recursive equations! E.g. is there an M such that

$$M\,x =_\beta \textbf{if}\,(\mathsf{Zero?}\,x)\,\textbf{then}\,1\,\textbf{else}\,\mathsf{Mult}\,x\,(M\,(\mathsf{Pred}\,x))?$$

Why fixed-point combinators?

In untyped λ-calculus, a fixed-point combinator F gives you a fixed point of every term M

$$F\,M \quad =_\beta M\,(F\,M).$$

Why is this useful?
Solve recursive equations! E.g. is there an M such that

$$M\,x =_\beta \textbf{if}\,(\text{Zero?}\,x)\,\textbf{then}\,1\,\textbf{else}\,\text{Mult}\,x\,(M\,(\text{Pred}\,x))?$$

Yes: take
$M := F\,(\lambda m.\lambda x.\textbf{if}\,(\text{Zero?}\,x)\,\textbf{then}\,1\,\textbf{else}\,\text{Mult}\,x\,(m\,(\text{Pred}\,x)),$
where F is your favourite fixed-point combinator.

Your favorite fixed point combinator?

$$
\begin{aligned}
Y &:= \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x)) \\
\Theta &:= (\lambda x\,f.f\,(x\,x\,f))\,(\lambda x\,f.f\,(x\,x\,f))
\end{aligned}
$$

Your favorite fixed point combinator?

$$
\begin{array}{rcl}
Y & := & \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x)) \\
\Theta & := & (\lambda x\,f.f\,(x\,x\,f))\,(\lambda x\,f.f\,(x\,x\,f))
\end{array}
$$

$$
L \;:=\; \lambda f.(\lambda x.x\,(\lambda p\,q.f\,(q\,p\,q))\,x)\,(\lambda y.y\,y)
$$

Your favorite fixed point combinator?

$$
\begin{array}{lcl}
Y & := & \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x)) \\
\Theta & := & (\lambda x\,f.f\,(x\,x\,f))\,(\lambda x\,f.f\,(x\,x\,f))
\end{array}
$$

$$
L \quad := \quad \lambda f.(\lambda x.x\,(\lambda p\,q.f\,(q\,p\,q))\,x)\,(\lambda y.y\,y)
$$

Writing $M_f := \lambda p\,q.f\,(q\,p\,q)$, $\omega := \lambda y.y\,y$, we have

$$
\begin{array}{lcl}
L\,f & =_\beta & \omega\,M_f\,\omega \\
 & =_\beta & M_f\,M_f\,\omega \\
 & =_\beta & f\,(\omega\,M_f\,\omega)
\end{array}
$$

Your favorite fixed point combinator?

$$
\begin{array}{rcl}
Y & := & \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x)) \\
\Theta & := & (\lambda x\,f.f\,(x\,x\,f))\,(\lambda x\,f.f\,(x\,x\,f))
\end{array}
$$

$$
L \quad := \quad \lambda f.(\lambda x.x\,(\lambda p\,q.f\,(q\,p\,q))\,x)\,(\lambda y.y\,y)
$$

Writing $M_f := \lambda p\,q.f\,(q\,p\,q)$, $\omega := \lambda y.y\,y$, we have

$$
\begin{array}{rcl}
L\,f & =_\beta & \omega\,M_f\,\omega \\
 & =_\beta & M_f\,M_f\,\omega \\
 & =_\beta & f\,(\omega\,M_f\,\omega)
\end{array}
$$

Theorem

- L is typable in λU.
- Y and Θ and Ω $(= \omega\,\omega)$ are not typable in λU.

What is λU?

λU is higher order predicate logic over polymorphic domains: two impredicative sorts on top of eachother.

What is λU?

λU is higher order predicate logic over polymorphic domains: two impredicative sorts on top of eachother.

More precisely:

- $\star : \Box$, $\Box : \triangle$ (In Rocq: $\mathit{Prop} : \mathit{Type}_1$, $\mathit{Type}_1 : \mathit{Type}_2$)
- $\star$ is the impredicative type of formulas, giving higher order predicate logic
- $\Box$ is the impredicative type of data types, giving, e.g. $\mathsf{nat} := \Pi k{:}\Box. k \to (k \to k) \to k$ of type $\Box$.

What is λU?

λU is higher order predicate logic over polymorphic domains: two impredicative sorts on top of eachother.
More precisely:

- $\star : \Box$, $\Box : \triangle$ (In Rocq: *Prop* : *Type*$_1$, *Type*$_1$: *Type*$_2$)
- $\star$ is the impredicative type of formulas, giving higher order predicate logic
- $\Box$ is the impredicative type of data types, giving, e.g. nat $:= \Pi k{:}\Box.k \to (k \to k) \to k$ of type $\Box$.
- λU also allows quantification over $\Box$: we have $\Pi k{:}\Box.\varphi \ : \ \star$ (for $\varphi : \star$).
- λU^- is λU without quantification over $\Box$.

Inconsistency of λU

- Girard 1972: λU is inconsistent (and therefore $\lambda\star$, with $\star : \star$) is inconsistent.
  That is: there is a closed term M of type $\bot := \Pi\alpha : \star.\alpha$:
  $$\vdash M : \bot.$$
- NB. a term $M : \bot$ does not have a normal form.

Inconsistency of λU

- Girard 1972: λU is inconsistent (and therefore $\lambda\star$, with $\star : \star$) is inconsistent.
  That is: there is a closed term M of type $\bot := \Pi\alpha : \star.\alpha$:
  $$\vdash M : \bot.$$
- NB. a term $M : \bot$ does not have a normal form.
- Question (Girard): is λU^- also inconsistent? Answer (Coquand 1994): yes, λU^- is also inconsistent.
- Hurkens (1995): a short proof of inconsistency of λU^-, i.e.: one can actually observe the term M and play with it.

What can we compute in λU?

- Howe 1987 (based on Coquand's 1986 analysis of Girard's proof) transformed $M : \bot$ into a term M_f with

$$\alpha : \star, f : \alpha \to \alpha \vdash M_f : \alpha.$$

- Howe showed (in $\lambda\star$) that from M_f a looping combinator can be defined: a family of terms $\{L_n\}_{n \in \mathbb{N}}$ such that

$$L_n\, f =_\beta f(L_{n+1}\, f).$$

NB. This is enough to define all partial recursive functions.

What can we compute in λU?

- Howe 1987 (based on Coquand's 1986 analysis of Girard's proof) transformed $M : \bot$ into a term M_f with
$$\alpha : \star, f : \alpha{\to}\alpha \vdash M_f : \alpha.$$
- Howe showed (in $\lambda\star$) that from M_f a looping combinator can be defined: a family of terms $\{L_n\}_{n\in\mathbb{N}}$ such that
$$L_n\, f =_\beta f(L_{n+1}\, f).$$

  NB. This is enough to define all partial recursive functions.
- A similar construction can be carried out in λU and λU^- (Coquand and Herbelin 1994).
- G. and Pollack (in 1995) showed that the inconsistency proof of Hurkens yields a looping combinator $\{L_n\}_{n\in\mathbb{N}}$ in λU^- (see Barthe and Coquand 2006).

So we can do everything in λU^-?

What can we not compute in λU?

- Are all untyped λ-terms typable in λU?

What can we not compute in λU?

- Are all untyped λ-terms typable in λU? No

What can we not compute in λU?

- Are all untyped λ-terms typable in λU? No
- Is there a fixed point combinator in λU?

What can we not compute in λU?

- Are all untyped λ-terms typable in λU? No
- Is there a fixed point combinator in λU? Don't know...

What can we not compute in λU?

- Are all untyped λ-terms typable in λU? No
- Is there a fixed point combinator in λU? Don't know...

To be precise: λU is the following Pure Type System.

$$\lambda U \quad \boxed{\begin{array}{ll} \mathcal{S} & \star, \Box, \triangle \\ \mathcal{A} & \star : \Box, \Box : \triangle \\ \mathcal{R} & (\star,\star), (\Box,\star), (\triangle,\star), (\Box,\Box), (\triangle,\Box) \end{array}}$$

If you know Rocq:

$$\lambda U \quad \boxed{\begin{array}{ll} \mathcal{S} & \mathit{Prop}, \mathit{Type}_1, \mathit{Type}_2 \\ \mathcal{A} & \mathit{Prop} : \mathit{Type}_1, \mathit{Type}_1 : \mathit{Type}_2 \\ \mathcal{R} & (\mathit{Prop}, \mathit{Prop}), (\mathit{Type}_1, \mathit{Prop}), (\mathit{Type}_2, \mathit{Prop}), \\ & (\mathit{Type}_1, \mathit{Type}_1), (\mathit{Type}_2, \mathit{Type}_1) \end{array}}$$

- We don't give the full typing rules.
- We divide the set of variables $\mathcal{V}$ into three disjoint sets $\mathrm{var}^{\star}$, $\mathrm{var}^{\Box}$ and $\mathrm{var}^{\triangle}$.
- We use standard characters:
  $\mathrm{var}^{\star} = \{x, y, z, \ldots\}$,
  $\mathrm{var}^{\Box} = \{\alpha, \beta, \gamma, \ldots\}$,
  $\mathrm{var}^{\triangle} = \{k_1, k_2, k_3, \ldots\}$.
  So a variable that lives in a type $\sigma : \star$ is typically x, y or z.

We define the syntactic categories Kinds (K_1, K_2), Constructors (P, Q) and Proof terms (p, q). We also introduce Types (σ, τ) (where $\text{Types} \subset \text{Constructors}$).

We define the syntactic categories Kinds (K_1, K_2), Constructors (P, Q) and Proof terms (p, q). We also introduce Types (σ, τ) (where Types $\subset$ Constructors).

$$
\begin{array}{llll}
\text{Kinds} & K & ::= & k \mid \star \mid K \to K \mid \Pi k{:}\Box.K \\[2ex]
\text{Constructors} & P & ::= & \alpha \mid \lambda\alpha{:}K.P \mid P\,P \\
 & & & \mid \lambda k{:}\Box.P \mid P\,K \\
 & & & \mid P \to P \mid \Pi\alpha{:}K.P \\[2ex]
\text{Types} & \sigma & ::= & \sigma \to \sigma \mid \Pi\alpha{:}K.\sigma \\[2ex]
\text{Proof terms} & q & ::= & x \mid \lambda x{:}\sigma.q \mid q\,q \\
 & & & \mid \lambda\alpha{:}K.q \mid q\,P \\
 & & & \mid \lambda k{:}\Box.q \mid q\,K
\end{array}
$$

λU schematically:

Proof terms	Constructors Types	Kinds	
	P, Q	$: K$	$: \Box(\mathit{Type}_1)$
p, q	$: \sigma, \tau$	$: \star(\mathit{Prop})$	
x, y, z	α, β, γ	k_1, k_2, k_3	
$\lambda x{:}\sigma.q \;,\; q\,p$			
$\lambda\alpha{:}K.q \;,\; q\,P$	$\lambda\alpha{:}K.Q \;,\; Q\,P$		
$\lambda k{:}\Box.q \;,\; q\,K$	$\lambda k{:}\Box.P \;,\; P\,K$		
	$\sigma \to \tau \;,\; \Pi\alpha{:}K.\sigma$	$K \to K \;,\; \Pi k{:}\Box.K$	

λU schematically:

Proof terms	Constructors / Types	Kinds	
	P, Q	$: K$	$: \Box(\mathit{Type}_1)$
p, q	$: \sigma, \tau$	$: \star(\mathit{Prop})$	
x, y, z	α, β, γ	k_1, k_2, k_3	
$\lambda x{:}\sigma.q \ , \ q\, p$			
$\lambda \alpha{:}K.q \ , \ q\, P$	$\lambda \alpha{:}K.Q \ , \ Q\, P$		
$\lambda k{:}\Box.q \ , \ q\, K$	$\lambda k{:}\Box.P \ , \ P\, K$		
	$\sigma \to \tau \ , \ \Pi \alpha{:}K.\sigma$	$K \to K \ , \ \Pi k{:}\Box.K$	

Lemma

- Everything to the right of Proof terms is normalizing.
- Type checking is decidable in λU.

Erasure from λU to untyped λ-calculus

For q a proof term of λU, we define the erasure of q, denoted by $|t|$ as follows.

$$
\begin{array}{lcllcl}
|x| & = & x & & & \\
|\lambda x{:}\sigma.p| & = & \lambda x.|p| & |p\,q| & = & |p||q| \\
|\lambda \alpha{:}K.p| & = & |p| & |p\,Q| & = & |p| \\
|\lambda k{:}\Box.p| & = & |p| & |p\,K| & = & |p|
\end{array}
$$

Definition

The untyped lambda term M is typable in λU if there exist Γ, q, σ such that

$$\Gamma \vdash q : \sigma : \star \qquad \text{and } |q| = M.$$

Main result

Proposition

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

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

Theorem

Double self-application is not possible in λU.
Here we mean with "double self-application" a term $q : \sigma : \star$ such that

$$|q| = (\lambda x.M)(\lambda y.N)$$

and M contains a sub-term $x\,x$ and N contains a sub-term $y\,y$.

So the erasure of a double self-application looks like this:

$$|q| = (\lambda x.\ldots x\,x\ldots)(\lambda y.\ldots y\,y\ldots).$$

Parse trees of types

A type σ in normal form is of one of the following two forms ($\vec{v}$ and $\vec{V}$ or $\vec{T}$ may be empty).

- $\Pi \vec{v} : \vec{V}.\tau \to \rho$
- $\Pi \vec{v} : \vec{V}.\alpha \vec{T}$

Parse trees of types

A type σ in normal form is of one of the following two forms ($\vec{v}$ and $\vec{V}$ or $\vec{T}$ may be empty).

- $\Pi\vec{v} : \vec{V}.\tau \to \rho$
- $\Pi\vec{v} : \vec{V}.\alpha\,\vec{T}$

We extend the notion of parse tree of a type σ, known from Urzyczyn 1997 for system Fω.

Definition

We define the parse tree of a type σ (written $\mathrm{pt}(\sigma)$) as follows.

$$
\mathrm{pt}(\Pi\vec{v} : \vec{V}.\tau \to \rho) \quad := \quad
\begin{tikzcd}
 & \Pi\vec{v} : \vec{V} \arrow[dl] \arrow[dr] & \\
\mathrm{pt}(\tau) & & \mathrm{pt}(\rho)
\end{tikzcd}
$$

$$
\mathrm{pt}(\Pi\vec{v} : \vec{V}.\alpha\,\vec{T}) \quad := \quad \Pi\vec{v} : \vec{V}.\alpha\,\vec{T}
$$

Analysing the parse trees of a type

Definition

- The left-terminal path of $\mathrm{pt}(\sigma)$, $\mathrm{ltp}(\mathrm{pt}(\sigma))$ is the left-most path in $\mathrm{pt}(\sigma)$ that ends in a node labelled $\Pi\vec{v} : \vec{V}.\alpha\ \vec{T}$.
- The variable α we arrive at is called the head variable of the type σ, $\mathrm{hv}(\sigma)$.

Definition

- The left-terminal path of $\mathrm{pt}(\sigma)$, $\mathrm{ltp}(\mathrm{pt}(\sigma))$ is the left-most path in $\mathrm{pt}(\sigma)$ that ends in a node labelled $\Pi\vec{v} : \vec{V}.\alpha\, \vec{T}$.
- The variable α we arrive at is called the head variable of the type σ, $\mathrm{hv}(\sigma)$.

Definition

For σ, τ types $\sigma \preceq \tau$ (σ is contained in τ), is defined by

$$\Pi\vec{v}{:}\vec{V}.\rho \quad \preceq \quad \Pi\vec{w}{:}\vec{W}.\rho[\vec{T}/\vec{v}],$$

where the variables in $\vec{w}$ do not occur free in σ.

The containment relation is reflexive and transitive.

Definition

- The left-terminal path of $\mathrm{pt}(\sigma)$, $\mathrm{ltp}(\mathrm{pt}(\sigma))$ is the left-most path in $\mathrm{pt}(\sigma)$ that ends in a node labelled $\Pi\vec{v} : \vec{V}.\alpha\ \vec{T}$.
- The variable α we arrive at is called the head variable of the type σ, $\mathrm{hv}(\sigma)$.

Definition

For σ, τ types $\sigma \preceq \tau$ (σ is contained in τ), is defined by

$$\Pi\vec{v}{:}\vec{V}.\rho \quad \preceq \quad \Pi\vec{w}{:}\vec{W}.\rho[\vec{T}/\vec{v}],$$

where the variables in $\vec{w}$ do not occur free in σ.

The containment relation is reflexive and transitive.

Lemma

If $\sigma \preceq \tau$, then $\mathrm{length}(\mathrm{ltp}(\sigma)) \leq \mathrm{length}(\mathrm{ltp}(\tau))$.

Analysing self-application

Lemma

If $\sigma \preceq \tau$ and $\mathrm{length}(\mathrm{ltp}(\sigma)) < \mathrm{length}(\mathrm{ltp}(\tau))$, then $\mathrm{hv}(\sigma)$ is bound at the root of $\mathrm{pt}(\sigma)$.

Analysing self-application

Lemma

If $\sigma \preceq \tau$ and $\mathrm{length}(\mathrm{ltp}(\sigma)) < \mathrm{length}(\mathrm{ltp}(\tau))$, then $\mathrm{hv}(\sigma)$ is bound at the root of $\mathrm{pt}(\sigma)$.

Proposition

If $t : \sigma : \star$ and t contains a self application of x, with $x : \sigma$, then $\mathrm{hv}(\sigma)$ is bound at the root of $\mathrm{pt}(\sigma)$.

Lemma

If $\sigma \preceq \tau$ and $\mathrm{length}(\mathrm{ltp}(\sigma)) < \mathrm{length}(\mathrm{ltp}(\tau))$, then $\mathrm{hv}(\sigma)$ is bound at the root of $\mathrm{pt}(\sigma)$.

Proposition

If $t : \sigma : \star$ and t contains a self application of x, with $x : \sigma$, then $\mathrm{hv}(\sigma)$ is bound at the root of $\mathrm{pt}(\sigma)$.

Proof

The general form of the self-application of $x : \sigma$ in t is

$$x\,\vec{T}\,(\lambda\vec{w} : \vec{W}.x\,\vec{R}).$$

We have $x\vec{T} : \rho_1 \to \rho_2$ and $\lambda\vec{w} : \vec{W}.x\,\vec{R} : \rho_1$ for some ρ_1, ρ_2, where $\sigma \preceq \rho_1 \to \rho_2$ and $\sigma \preceq \rho_1$.

Analysing self-application

Lemma

If $\sigma \preceq \tau$ and $\mathrm{length}(\mathrm{ltp}(\sigma)) < \mathrm{length}(\mathrm{ltp}(\tau))$, then $\mathrm{hv}(\sigma)$ is bound at the root of $\mathrm{pt}(\sigma)$.

Proposition

If $t : \sigma : \star$ and t contains a self application of x, with $x : \sigma$, then $\mathrm{hv}(\sigma)$ is bound at the root of $\mathrm{pt}(\sigma)$.

Proof

The general form of the self-application of $x : \sigma$ in t is

$$x\,\vec{T}\,(\lambda \vec{w} : \vec{W}.x\,\vec{R}).$$

We have $x\,\vec{T} : \rho_1 \to \rho_2$ and $\lambda \vec{w} : \vec{W}.x\,\vec{R} : \rho_1$ for some ρ_1, ρ_2, where $\sigma \preceq \rho_1 \to \rho_2$ and $\sigma \preceq \rho_1$.
Also $\mathrm{length}(\mathrm{ltp}(\rho_1 \to \rho_2)) = \mathrm{length}(\mathrm{ltp}(\rho_1)) + 1$, so
$\mathrm{length}(\mathrm{ltp}(\sigma)) \leq \mathrm{length}(\mathrm{ltp}(\rho_1)) < \mathrm{length}(\mathrm{ltp}(\rho_1 \to \rho_2))$,
so $\mathrm{hv}(\sigma)$ is bound at the root of $\mathrm{pt}(\sigma)$. □

No Ω-like terms are typable in λU

Theorem

In λU there is no typable term t such that

$$|t| = (\lambda x. \dots x\, x \dots)(\lambda y. \dots y\, y \dots).$$

No Ω-like terms are typable in λU

Theorem

In λU there is no typable term t such that

$$|t| = (\lambda x. \ldots x\, x \ldots)(\lambda y. \ldots y\, y \ldots).$$

Proof

We can assume that t has the following shape

$$(\overbrace{\lambda x : \sigma. \ldots}^{q})\,(\overbrace{\lambda \vec{w} : \vec{W}.\lambda y : \rho. \ldots}^{p}),$$

with $q : \sigma \to \tau$ and $p : \sigma$.

No Ω-like terms are typable in λU

Theorem

In λU there is no typable term t such that

$$|t| = (\lambda x. \ldots x\, x \ldots)(\lambda y. \ldots y\, y \ldots).$$

Proof

We can assume that t has the following shape

$$(\overbrace{\lambda x : \sigma. \ldots}^{q})\, (\overbrace{\lambda \vec{w} : \vec{W}. \lambda y : \rho. \ldots}^{p}),$$

with $q : \sigma \to \tau$ and $p : \sigma$.

1. The $\mathrm{hv}(\sigma)$ is bound at the root of $\mathrm{pt}(\sigma)$.
2. The $\mathrm{hv}(\rho)$ is bound at the root of $\mathrm{pt}(\rho)$.
3. $\sigma =_\beta \Pi \vec{w} : \vec{W}. \rho \to \mu$ for some μ.
4. Contradiction, so the term t cannot be well-typed. $\square$

Conclusion

The following well-known untyped λ-terms are not typable in λU:

$$\begin{array}{rcl} \Omega & = & (\lambda x.x\,x)\,(\lambda x.x\,x), \\ Y & = & \lambda f.(\lambda x.f(x\,x))\,(\lambda x.f(x\,x)), \\ \Theta & = & (\lambda x\,f.f(x\,x\,f)\,(\lambda x\,f.f(x\,x\,f)). \end{array}$$

Conclusion

The following well-known untyped λ-terms are not typable in λU:

$$\begin{array}{rcl} \Omega & = & (\lambda x.x\,x)\,(\lambda x.x\,x), \\ Y & = & \lambda f.(\lambda x.f(x\,x))\,(\lambda x.f(x\,x)), \\ \Theta & = & (\lambda x\,f.f(x\,x\,f)\,(\lambda x\,f.f(x\,x\,f)). \end{array}$$

- NB. the typable fixed-point combinator

$$L := \lambda f.(\lambda x.x\,(\lambda p\,q.f\,(q\,p\,q))\,x)\,(\lambda y.y\,y)$$

does not have double self-application.

- That the typable version of L is not a fixed-point combinator (but merely a looping-combinator) is due to the type annotations in the λ-abstractions.

Further work / Open questions

- Is there a fixed-point combinator typable in λU?
- Is Ω (Y, Θ, ...) typable in $\lambda\star$?
- Do other paradoxes give significant other looping combinators?

Questions?