

Υ 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

$$F M =_{\beta} M (F M).$$



Why fixed-point combinators?

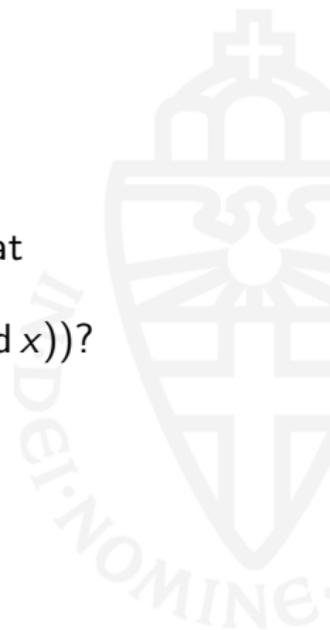
In untyped λ -calculus, a **fixed-point combinator** F gives you a fixed point of every term M

$$F M =_{\beta} M (F M).$$

Why is this useful?

Solve recursive equations! E.g. is there an M such that

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



Why fixed-point combinators?

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

$$F M =_{\beta} M (F M).$$

Why is this useful?

Solve recursive equations! E.g. is there an M such that

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

Yes: take

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

Your favorite fixed point combinator?

$Y := \lambda f.(\lambda x.f(x x))(\lambda x.f(x x))$

$\Theta := (\lambda x f.f(x x f))(\lambda x f.f(x x f))$



Your favorite fixed point combinator?

$$Y := \lambda f.(\lambda x.f(x x))(\lambda x.f(x x))$$

$$\Theta := (\lambda x f.f(x x f))(\lambda x f.f(x x f))$$

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



Your favorite fixed point combinator?

$$Y := \lambda f.(\lambda x.f(x x))(\lambda x.f(x x))$$

$$\Theta := (\lambda x f.f(x x f))(\lambda x f.f(x x f))$$

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

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

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



Your favorite fixed point combinator?

$$Y := \lambda f.(\lambda x.f(x x))(\lambda x.f(x x))$$

$$\Theta := (\lambda x f.f(x x f))(\lambda x f.f(x x f))$$

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

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

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

THEOREM

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

What is λU ?

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

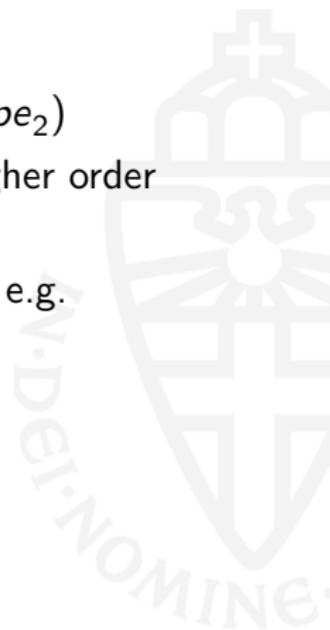


What is λU ?

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

More precisely:

- $\star : \square, \square : \triangle$ (In Rocq: $Prop : Type_1, Type_1 : Type_2$)
- \star is the **impredicative** type of formulas, giving higher order predicate logic
- \square is the **impredicative** type of data types, giving, e.g. $\text{nat} := \prod k : \square. k \rightarrow (k \rightarrow k) \rightarrow k$ of type \square .



What is λU ?

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

More precisely:

- $\star : \square, \square : \triangle$ (In Rocq: $Prop : Type_1, Type_1 : Type_2$)
- \star is the **impredicative** type of formulas, giving higher order predicate logic
- \square is the **impredicative** type of data types, giving, e.g. $\text{nat} := \prod k : \square. k \rightarrow (k \rightarrow k) \rightarrow k$ of type \square .
- λU also allows quantification over \square :
we have $\prod k : \square. \varphi : \star$ (for $\varphi : \star$).
- λU^- is λU without quantification over \square .

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 $\perp := \prod \alpha : \star.\alpha$:

$$\vdash M : \perp.$$

- NB. a term $M : \perp$ 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 $\perp := \prod \alpha : \star.\alpha$:

$$\vdash M : \perp.$$

- NB. a term $M : \perp$ 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 $M : \perp$ into a term M_f with

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

- 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 $M : \perp$ into a term M_f with

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

- 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.

λU	S	$\star, \square, \triangle$
	A	$\star : \square, \square : \triangle$
	R	$(\star, \star), (\square, \star), (\triangle, \star), (\square, \square), (\triangle, \square)$

If you know Rocq:

λU	S	$Prop, Type_1, Type_2$
	A	$Prop : Type_1, Type_1 : Type_2$
	R	$(Prop, Prop), (Type_1, Prop), (Type_2, Prop), (Type_1, Type_1), (Type_2, Type_1)$

- We don't give the full typing rules.
- We divide the set of variables \mathcal{V} into three disjoint sets var^* , var^\square and var^Δ .

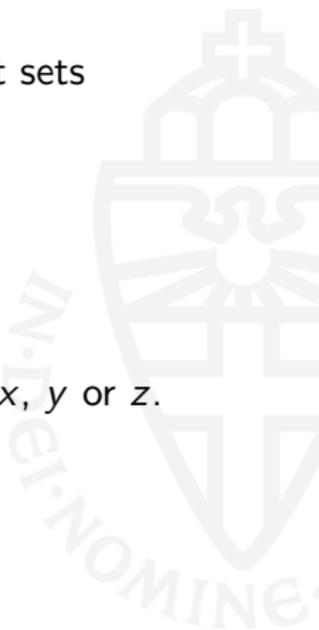
- We use standard characters:

$$\text{var}^* = \{x, y, z, \dots\},$$

$$\text{var}^\square = \{\alpha, \beta, \gamma, \dots\},$$

$$\text{var}^\Delta = \{k_1, k_2, k_3, \dots\}.$$

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 $\text{Types} \subset \text{Constructors}$).

Kinds $K ::= k \mid \star \mid K \rightarrow K \mid \Pi k:\square.K$

Constructors $P ::= \alpha \mid \lambda\alpha:K.P \mid PP$
 $\mid \lambda k:\square.P \mid PK$
 $\mid P \rightarrow P \mid \Pi\alpha:K.P$

Types $\sigma ::= \sigma \rightarrow \sigma \mid \Pi\alpha:K.\sigma$

Proof terms $q ::= x \mid \lambda x:\sigma.q \mid qq$
 $\mid \lambda\alpha:K.q \mid qP$
 $\mid \lambda k:\square.q \mid qK$



λU schematically:

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

λU schematically:

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

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.

$$\begin{array}{ll} |x| & = x \\ |\lambda x:\sigma.p| & = \lambda x.|p| & |pq| & = |p||q| \\ |\lambda \alpha:K.p| & = |p| & |pQ| & = |p| \\ |\lambda k:\square.p| & = |p| & |pK| & = |p| \end{array}$$

DEFINITION

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

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

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:

$$|q| = (\lambda x.\dots x x \dots)(\lambda y.\dots y y \dots).$$

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).

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



Parse trees of types

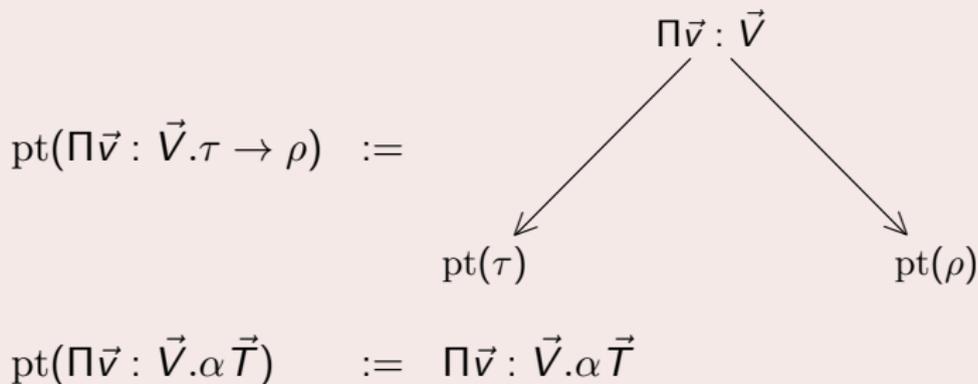
A type σ in normal form is of one of the following two forms (\vec{v} and \vec{V} or \vec{T} may be empty).

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

We extend the notion of **parse tree** of a type σ , known from Urzyczyn 1997 for system $F\omega$.

DEFINITION

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



DEFINITION

- The **left-terminal path** of $\text{pt}(\sigma)$, $\text{ltp}(\text{pt}(\sigma))$ is the left-most path in $\text{pt}(\sigma)$ 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 , $\text{hv}(\sigma)$.



DEFINITION

- The **left-terminal path** of $\text{pt}(\sigma)$, $\text{ltp}(\text{pt}(\sigma))$ is the left-most path in $\text{pt}(\sigma)$ 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 , $\text{hv}(\sigma)$.

DEFINITION

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

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

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 $\text{pt}(\sigma)$, $\text{ltp}(\text{pt}(\sigma))$ is the left-most path in $\text{pt}(\sigma)$ 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 , $\text{hv}(\sigma)$.

DEFINITION

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

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

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 $\text{length}(\text{ltp}(\sigma)) \leq \text{length}(\text{ltp}(\tau))$.

LEMMA

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



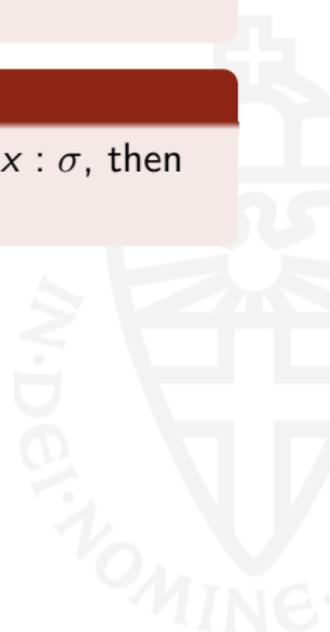
Analysing self-application

LEMMA

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

PROPOSITION

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



LEMMA

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

PROPOSITION

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

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

LEMMA

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

PROPOSITION

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

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

Also $\text{length}(\text{ltp}(\rho_1 \rightarrow \rho_2)) = \text{length}(\text{ltp}(\rho_1)) + 1$, so $\text{length}(\text{ltp}(\sigma)) \leq \text{length}(\text{ltp}(\rho_1)) < \text{length}(\text{ltp}(\rho_1 \rightarrow \rho_2))$, so $\text{hv}(\sigma)$ is bound at the root of $\text{pt}(\sigma)$. □

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

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

PROOF

We can assume that t has the following shape

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

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

No Ω -like terms are typable in λU

THEOREM

In λU there is no typable term t such that

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

PROOF

We can assume that t has the following shape

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

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

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

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

$$\Omega = (\lambda x. x x) (\lambda x. x x),$$

$$Y = \lambda f. (\lambda x. f(x x)) (\lambda x. f(x x)),$$

$$\Theta = (\lambda x f. f(x x f)) (\lambda x f. f(x x f)).$$



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

$$\Omega = (\lambda x. x x) (\lambda x. x x),$$

$$Y = \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x)),$$

$$\Theta = (\lambda x f. f (x x f)) (\lambda x f. f (x x f)).$$

- 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.

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



Questions?

