Complexity of normalization for subsystems of untyped linear lambda calculus

Noam Zeilberger

Structure meets Power Workshop 2021

27-28 June 2021

based on joint work with

Anupam Das, Damiano Mazza, and Tito Nguyễn

Background

lambda calculus, types, linearity, and Mairson's proof of PTIME-completeness of linear lambda calculus

Lambda calculus: a very brief history*

Invented by Alonzo Church in late 20s, published in 1932

Original goal: foundation for logic without free variables

Minor defect: *inconsistent!*

Resolution: separate into an **untyped calculus** for computation, and a **typed calculus** for logic.

(Both have since found many uses.)

*Source: Cardone & Hindley's "History of Lambda-calculus and Combinatory Logic"

Untyped lambda calculus: syntax

Suppose given some collection of variables.
Terms are built up from variables using two fundamental operations.

terms	$t, u \; ::=$	x	$\mid$	$t(u)$	$\mid$	$\lambda x.t$
		variable		*application*		*abstraction*

Given two terms t and u and a free variable x of t, we can define the *capture-avoiding substitution* of u for x in t, written t[u/x].

Untyped lambda calculus: normalization

Computation through ***the rule of β-reduction:***

$$(\lambda x.t)(u) \ \to^{\beta} \ t[u/x]$$

can apply to any matching subterm
(confluent and weakly normalizing)

Sometimes paired with ***the rule of η-expansion:***

$$t \to^{\eta} \lambda x.t(x)$$

Example:

$$\begin{aligned} &(\lambda x.\lambda y.\lambda z.x(yz))(\lambda a.a)(t) \\ &\to^{\beta} (\lambda y.\lambda z.(\lambda a.a)(yz))(t) \\ &\to^{\beta} (\lambda y.\lambda z.yz)(t) \\ &\to^{\beta} \lambda z.t(z) \ {}^{\eta}\!\!\leftarrow t \end{aligned}$$

Simply-typed lambda calculus

Terms are refined by annotating them with simple types:

types	$A, B ::=$	α	|	$A \to B$	
		atomic type		*function type*	
typed terms	$t^A, u^B ::=$	x^A	| $(t^{A \to B}\, u^A)^B$	|	$(\lambda x^A . t^B)^{A \to B}$

Rules of β-reduction and η-expansion preserve typing, so STLC is a well-behaved subsystem of untyped LC.

An aside on combinatory logic

(an ancestor to λ-calculus)

Moses Schönfinkel

Consider the following set of closed terms:

B := $\lambda x.\lambda y.\lambda z.x(yz)$ **K** := $\lambda x.\lambda y.x$

C := $\lambda x.\lambda y.\lambda z.(xz)y$ **W** := $\lambda x.\lambda y.(xy)y$

This set forms a *basis* for untyped LC, meaning any closed term is obtainable from them via application and β-reduction.

Observe how C, K, W respectively reorder, erase, and duplicate variables...

Haskell Curry

Photos of logicians obtained from the Open Logic Project (see credits on page)

Untyped lambda calculus: normalization

Theorem [Church 1936]: *there is no effective procedure for deciding whether two untyped terms are β-equivalent, or whether a given term has a β-normal form.*

The original proof relied on "Church encoding" of natural numbers

$$
\begin{aligned}
1 &= \lambda f.\lambda x.f(x) \\
2 &= \lambda f.\lambda x.f(f(x)) \\
3 &= \lambda f.\lambda x.f(f(f(x))) \\
&\vdots
\end{aligned}
$$

as well as on a minimization operator due to Kleene, whose definition was soon simplified by Turing...

Fixpoints, typing, and (non-)linearity

Turing proved the equivalence of TMs and LC, and also gave the first *fixed-point combinator*.*

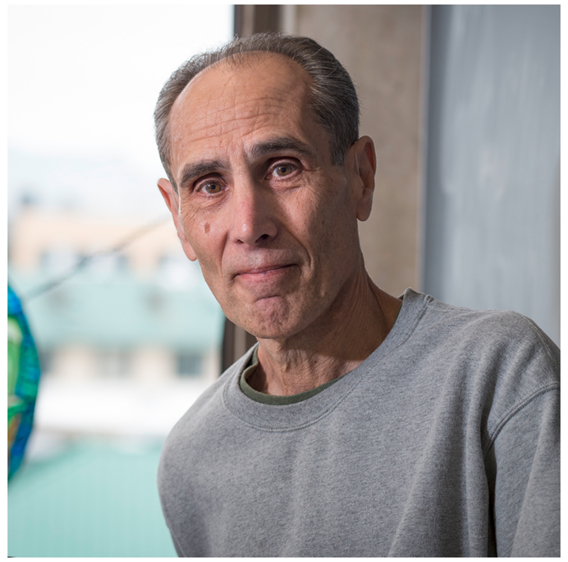
*see "Computability and λ-definability" and "The 𝔭-function in λ-K-conversion" in JSL 2:4, Dec 1937

$$\Theta = \boxed{(\lambda x.\lambda y.y(xxy))(\lambda x.\lambda y.y(xxy))}$$

The fixed-point combinator $\Theta t =_\beta t(\Theta t)$ *cannot be assigned a simple type,* since it would require a paradoxical type $\tau = \tau \to \tau$.

But also pay attention to the doubled uses of variables x and y!
A term where every variable is used exactly once is said to be *linear.*
(So B and C are linear, but K and W are non-linear, like Θ.)

Simply-typed vs linear normalization



Theorem [Statman 1979]: *deciding β-equivalence of simply-typed terms is not elementary recursive.*

Theorem [Mairson 2004]: *deciding β-equivalence of untyped* linear terms is complete for polynomial time.*

Mairson's proof

Linear lambda calculus and PTIME-completeness, JFP 14(6), Nov 2004

PTIME easy, since each β-reduction decreases the size of a linear term.

PTIME-hardness by reduction from the circuit value problem (CVP): any boolean circuit C can be encoded as a linear term ⌜C⌝, so that

$$C(v_1,\ldots,v_n) \Downarrow \text{true} \quad \text{iff} \quad \ulcorner C \urcorner\, \ulcorner v_1 \urcorner \ldots \ulcorner v_n \urcorner =_\beta \ulcorner \text{true} \urcorner.$$

Mairson's proof

Linear lambda calculus and PTIME-completeness, JFP 14(6), Nov 2004

Mairson replaces the usual "Church booleans" by the following:*

*up to inessential reordering of variables

$$\text{true} = \lambda x.\lambda y.\lambda z.(xy)z$$
$$\text{false} = \lambda x.\lambda y.\lambda z.(xz)y$$

The encoding is untyped, but w/2nd order quantifiers one can define

$$\text{bool} = \forall\alpha\beta.(\alpha \multimap \alpha \multimap \beta) \multimap (\alpha \multimap \alpha \multimap \beta)$$

and then assign any open circuit $C(x_1,\ldots,x_n)$ a uniform type

$$\ulcorner C \urcorner : \underbrace{\text{bool} \multimap \ldots \multimap}_{\text{n times}} \text{bool} \multimap \text{bool}$$

Some structural perspectives

Species and operads

A (plain) species S is a family of sets of elements $(S_n)_{n \in \mathbb{N}}$

An operad P is a species supporting composition and identity

$$\frac{f \in P_n \qquad g \in P_m}{f \circ_i g \in P_{n+m-1}} \qquad\qquad \frac{}{id \in P_1}$$

that satisfy appropriate associativity and neutrality laws.

A species/operad may be symmetric, cartesian, etc., if it comes equipped with additional "structural" rules...

The cartesian operad of untyped lambda terms

cf. Hyland's "Classical lambda calculus in modern dress"

basic judgment $x_1, \ldots, x_n \vdash t$

t is an untyped term with free variables $x_1, \ldots, x_n$

inductive definition of untyped terms in context

$$\frac{}{x \vdash x}\ \textit{var} \qquad \frac{\Gamma \vdash t \qquad \Delta \vdash u}{\Gamma, \Delta \vdash t(u)}\ \textit{app} \qquad \frac{\Gamma, x \vdash t}{\Gamma \vdash \lambda x.t}\ \textit{abs}$$

$$\frac{\Gamma \vdash t}{\Gamma, x \vdash t}\ \textit{wea} \qquad \frac{\Gamma, x, y \vdash t}{\Gamma, x \vdash t[x/y]}\ \textit{con} \qquad \frac{\Gamma, x, y, \Delta \vdash t}{\Gamma, y, x, \Delta \vdash t}\ \textit{exc}$$

operadic composition = substitution of a term for a free variable.
(don't quotient by β, but could define a 2-operad...)

The cartesian operad of untyped lambda terms
...and its non-cartesian suboperads

$$\frac{\Gamma \vdash t}{\Gamma , x \vdash t}\, wea \qquad \frac{\Gamma , x , y \vdash t}{\Gamma , x \vdash t[x/y]}\, con \qquad \frac{\Gamma , x, y , \Delta \vdash t}{\Gamma , y, x , \Delta \vdash t}\, exc$$

by dropping some of the structural rules, we can capture different families of untyped terms:

strict = -wea $\ni$ W

affine = -con $\ni$ K

linear = -wea,-con $\ni$ C

symmetric operad

these define suboperads and *subsystems* of LC, in the sense that all three suboperads are closed under β-reduction (and η-equivalence).

Colored operads of typed lambda terms

Typed lambda terms may be organized into *colored operads,* equipped with forgetful functors to operads of untyped terms. Typing an untyped term may be seen as a "lifting" problem.

$$\begin{tikzcd} D \arrow[d] \\ T \end{tikzcd}$$

cf. Melliès & Zeilberger POPL 2015, Mazza et al POPL 2018

The operad of simply-typed general terms/$=_{\beta\eta}$ is equivalent to the

free closed cartesian operad

on a set of atoms, the operad of simply-typed linear terms/=βη to the

free closed symmetric operad

on a set of atoms.

Planar and bridgeless lambda calculus

We consider the following subsystems* of linear lambda calculus:

*Note the graph-theoretic terminology is justified! (cf. Alex Singh's talk)

planar = -wea,-con,-exc
$\ni$ B
non-symmetric operad

bridgeless = -wea,-con,-(Γ=$\cdot$)
non-unitary operad

planar bridgeless = -wea,-con,exc,-(Γ=$\cdot$)

All define operads closed under β-reduction...and these restrictions are natural from an operadic perspective, among others!

Notes:

- Planar lambda calculus was briefly discussed by Abramsky (2008).
- Bridgeless planar lambda terms were [implicitly!] enumerated by Tutte (1962).
- The original Lambek calculus (1958) was both non-symmetric + non-unitary.

Questions

Structure meets power?

Imposing the planarity and/or bridgeless restrictions makes it harder to program – does it make it easier to decide β-equivalence?

- Positive answers would be exciting (e.g., complexity hierarchy by genus?)
- Negative answers would be counterintuitive (hence interesting)

We consider β-equivalence of untyped planar and/or bridgeless terms, but we are interested in whether they can be typed *uniformly.**

*Note all linear terms admit a principal simple type!

Questions of term representation (string vs graph) are potentially important when considering subpolynomial complexity classes.

partial

Answers

Our current state of knowledge [planar terms]

Deciding $=_\beta$ of untyped planar terms is PTIME-complete!

- Proof is an adaptation of Mairson's reduction from CVP (to be discussed...)
- We found two different planar encodings of boolean circuits, though only one admits uniform typing.
- Both encodings are "inherently non-bridgeless"

Our current state of knowledge [bridgeless terms]

There is a polynomial time Cook reduction from β-normalization of linear terms to β-normalization of bridgeless terms.

- This alas does not directly imply PTIME-hardness of deciding $=_\beta$
- Our reduction (inserting "handlers" around vars) is inherently non-planar

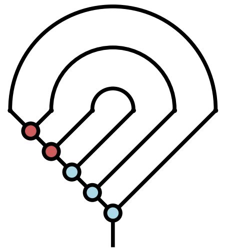
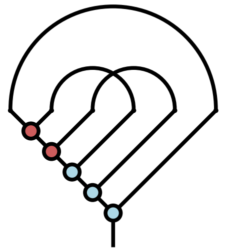
Our current state of knowledge [planar bridgeless terms]

Deciding $=_\beta$ of untyped planar bridgeless terms is L-hard on the graph representation, TC0-hard on the string representation.

- Reduction from directed forest reachability / counting.
- In both cases, our best upper bound is still P!

Revisiting Mairson's encoding of CVP

Non-planarity seems to be used in an essential way*...

true = $\lambda x.\lambda y.\lambda z.(xy)z$  *vs.*  $\lambda x.\lambda y.\lambda z.(xz)y$ = false

$$\text{bool} = \forall\alpha\beta.(\alpha \multimap \alpha \multimap \beta) \multimap (\alpha \multimap \alpha \multimap \beta)$$

*again we have reordered the arguments to true and false, which were both non-planar in Mairson's original bool = $\forall\alpha\beta.\alpha \multimap \alpha \multimap (\alpha \multimap \alpha \multimap \beta) \multimap \beta$

Can we give a planar encoding of booleans & boolean circuits?

Specification of the reduction from CVP

We will provide the following:

1. two distinct closed planar terms **true** + **false**

2. closed planar terms implementing **and**-, **or**-, **not**-gates...

and true true = true | and false true = false | ...
and true false = false | and false false = false

3. a closed planar term implementing a **fan-out** gate

copy B = λx.x B B for B ∈ {true,false}

4. a closed planar term implementing a **swap** gate

swap B_1 B_2 = λx.x B_2 B_1 for B_1,B_2 ∈ {true,false}

(Note that 123 ⇒ 4, by known reduction CVP → planar CVP!)

Solution #1

Take true and false to be as simple as possible...

```
false = \a.a
true  = \a.a(\b.b)
```

Then look for the other circuit gates by brute force proof search!

```
and = not = \a.a(\b.b(\c.c))
or = \a.\b.(a(b(\e.e(\f.f))))(\c.c(\d.d))
copy = \a.\b.((a(\e.\f.(e(\i.i))(f(\g.g(\h.h)))))(b))(\c.c(\d.d))
swap = \a.\b.\c.(a(\f.b(c(f))))(\d.d(\e.e))
```

We can find these pretty quickly, since there are "only" 2112357 β-normal planar lambda terms of size ≤ 26. (see https://oeis.org/A000168)

Solution #2

Consider the following polymorphic type (where ι is an atom)*:

$$\text{bool} = \forall\alpha.((\iota \multimap \iota) \multimap (\iota \multimap \iota) \multimap \alpha) \multimap (\iota \multimap \iota) \multimap \alpha$$

A boolean is "something which takes a continuation expecting a pair of endofunctions, as well as an endofunction, and returns an answer".

There are exactly two β-normal planar terms of type bool:

false = \k.\f.k(f)(\x.x)
true = \k.\f.k(\x.x)(f)

More generally, there are $\binom{n+m-1}{m}$ planar terms of type $\forall\alpha.((\iota \multimap \iota)^n \multimap \alpha) \multimap (\iota \multimap \iota)^m \multimap \alpha$

All other gates can be built and typed (replace brute force by "Tito force").

*it turns out that up to CPS translation, the same type appeared in Satoshi Matsuoka (2015), although Matsuoka's encoding of circuits was non-planar.

Conclusion

Studying the planar/bridgeless subsystems of λ-calculus is well-motivated by (varying types of) structural considerations.

Does the *decision problem* for $=_\beta$ become any easier? So far we have **a mix of negative results and inconclusive results:**

- Deciding $=_\beta$ of untyped planar terms is PTIME-complete.
- There is a polynomial time Cook reduction from β-normalization of linear terms to β-normalization of bridgeless terms.
- Deciding $=_\beta$ of untyped planar bridgeless terms is L-hard on the graph representation, TC0-hard on the string representation.

Worth comparing with recent positive results by Nguyễn and Pradic, on the decreased *expressive power* of planar lambda calculi…