

Simply typed β -convertibility is TOWER-complete even for safe λ -terms

Lê Thành Dũng (Tito) Nguyễn — nltd@nguyentito.eu – École normale supérieure de Lyon

4 March 2024, Swansea University

Dynamics of the untyped λ -calculus

A naive syntactic theory of functions:

$$\begin{aligned} f x &\approx f(x) \\ \lambda x. t &\approx x \mapsto t \\ (\lambda x. t) u \longrightarrow_{\beta} t\{x := u\} &\approx (x \mapsto x^2 + 1)(42) = 42^2 + 1 \end{aligned}$$

Dynamics of the untyped λ -calculus

A naive syntactic theory of functions:

$$\begin{aligned} f x &\approx f(x) \\ \lambda x. t &\approx x \mapsto t \\ (\lambda x. t) u \longrightarrow_{\beta} t\{x := u\} &\approx (x \mapsto x^2 + 1)(42) = 42^2 + 1 \end{aligned}$$

- Non-termination possible – actually, the untyped λ -calculus is Turing-complete

$$(\lambda x. x x) (\lambda x. x x) \longrightarrow_{\beta} (x x)\{x := (\lambda x. x x)\} = (\lambda x. x x) (\lambda x. x x) \longrightarrow_{\beta} \dots$$

Dynamics of the untyped λ -calculus

A naive syntactic theory of functions:

$$\begin{aligned} f x &\approx f(x) \\ \lambda x. t &\approx x \mapsto t \\ (\lambda x. t) u \longrightarrow_{\beta} t\{x := u\} &\approx (x \mapsto x^2 + 1)(42) = 42^2 + 1 \end{aligned}$$

- Non-termination possible – actually, the untyped λ -calculus is Turing-complete

$$(\lambda x. x x) (\lambda x. x x) \longrightarrow_{\beta} (x x)\{x := (\lambda x. x x)\} = (\lambda x. x x) (\lambda x. x x) \longrightarrow_{\beta} \dots$$

- *Convertibility* $t =_{\beta} u$ (reflexive transitive closure of \rightarrow_{β}) is undecidable

Dynamics of the untyped λ -calculus

A naive syntactic theory of functions:

$$\begin{aligned} f x &\approx f(x) \\ \lambda x. t &\approx x \mapsto t \\ (\lambda x. t) u \longrightarrow_{\beta} t\{x := u\} &\approx (x \mapsto x^2 + 1)(42) = 42^2 + 1 \end{aligned}$$

- Non-termination possible – actually, the untyped λ -calculus is Turing-complete

$$(\lambda x. x x) (\lambda x. x x) \longrightarrow_{\beta} (x x)\{x := (\lambda x. x x)\} = (\lambda x. x x) (\lambda x. x x) \longrightarrow_{\beta} \dots$$

- *Convertibility* $t =_{\beta} u$ (reflexive transitive closure of \rightarrow_{β}) is undecidable

This talk:

- Recall/clarify the complexity of $t =_{\beta} u$ for *simply typed* λ -terms
- Then extend the result to *safe* λ -terms

The simply typed λ -calculus

We now consider a *type system*: labeling λ -terms with specifications

$$t : A \rightarrow B \quad \approx \quad "t \text{ is a function from } A \text{ to } B"$$

Simple types: built using " \rightarrow " from a base type o

The simply typed λ -calculus

We now consider a *type system*: labeling λ -terms with specifications

$$t : A \rightarrow B \quad \approx \quad "t \text{ is a function from } A \text{ to } B"$$

Simple types: built using " \rightarrow " from a base type o

$$\frac{f : o \rightarrow o \quad \frac{f : o \rightarrow o \quad x : o}{f x : o}}{f (f x) : o}$$

The simply typed λ -calculus

We now consider a *type system*: labeling λ -terms with specifications

$$t : A \rightarrow B \approx \text{“}t \text{ is a function from } A \text{ to } B\text{”}$$

Simple types: built using “ \rightarrow ” from a base type o

$$\frac{f : o \rightarrow o}{f(fx) : o} \qquad \frac{f : o \rightarrow o \quad x : o}{fx : o} \qquad \underbrace{\lambda f. \lambda x. f(fx)}_{\text{encodes the number 2}} : \overbrace{(o \rightarrow o) \rightarrow o \rightarrow o}^{\text{“type of natural numbers”}}$$

The simply typed λ -calculus

We now consider a *type system*: labeling λ -terms with specifications

$$t : A \rightarrow B \quad \approx \quad \text{“}t \text{ is a function from } A \text{ to } B\text{”}$$

Simple types: built using “ \rightarrow ” from a base type o

$$\frac{f : o \rightarrow o}{f(fx) : o} \qquad \frac{f : o \rightarrow o \quad x : o}{fx : o} \qquad \underbrace{\lambda f. \lambda x. f(fx)}_{\text{encodes the number 2}} : \overbrace{(o \rightarrow o) \rightarrow o \rightarrow o}^{\text{“type of natural numbers”}}$$

Theorem

\rightarrow_β is strongly normalizing (terminating) and confluent on simply typed λ -terms.

Corollary

$=_\beta$ is decidable on simply typed λ -terms.

Just compute the normal forms and compare them!

Two related questions

Theorem

\rightarrow_β is strongly normalizing (terminating) and confluent on simply typed λ -terms.

Corollary

$=_\beta$ is decidable on simply typed λ -terms (compute & compare normal forms).

- Combinatorics: what's the length of \rightarrow_β sequences?
(for *linear* λ -terms: see Alexandros Singh's PhD thesis)
- Computational complexity: how hard is it to decide $=_\beta$?

What's the length of \rightarrow_β sequences?

- Schwichtenberg, *Complexity of Normalization in the Pure Typed Lambda-Calculus*, 1982
- Beckmann, *Exact bounds for lengths of reductions in typed λ -calculus*, 2001

What's the length of \rightarrow_β sequences?

- Schwichtenberg, *Complexity of Normalization in the Pure Typed Lambda-Calculus*, 1982
- Beckmann, *Exact bounds for lengths of reductions in typed λ -calculus*, 2001

Let the *order* of a type be the maximum nesting of \rightarrow to the left:

$$\text{ord}(o) = 0 \quad \text{ord}(A \rightarrow B) = \max(\text{ord}(A) + 1, \text{ord}(B))$$

Theorem

Let t be a simply typed λ -term. If all subterms of t have types of order $\leq k$, then the maximum reduction length for t is at most $2_k(O(\text{size}(t)))$ where $2_0(x) = x$ and $2_{n+1}(x) = 2^{2_n(x)}$.
 \rightsquigarrow tower of exponentials of height k

What's the length of \rightarrow_β sequences?

- Schwichtenberg, *Complexity of Normalization in the Pure Typed Lambda-Calculus*, 1982
- Beckmann, *Exact bounds for lengths of reductions in typed λ -calculus*, 2001

Let the *order* of a type be the maximum nesting of \rightarrow to the left:

$$\text{ord}(o) = 0 \quad \text{ord}(A \rightarrow B) = \max(\text{ord}(A) + 1, \text{ord}(B))$$

Theorem

Let t be a simply typed λ -term. If all subterms of t have types of order $\leq k$, then the maximum reduction length for t is at most $2_k(O(\text{size}(t)))$ where $2_0(x) = x$ and $2_{n+1}(x) = 2^{2_n(x)}$.

\rightsquigarrow tower of exponentials of height k

$$\underbrace{\overbrace{(\lambda f. \lambda x. f(f x))}^{\text{type of order } k+1} \dots \overbrace{(\lambda f. \lambda x. f(f x))}^{\text{type } (o \rightarrow o) \rightarrow (o \rightarrow o) \text{ of order } 2}}_{k \text{ times}} \longrightarrow_\beta^* \text{something of size } \Theta(2_k(1))$$

What's the length of \rightarrow_β sequences?

- Schwichtenberg, *Complexity of Normalization in the Pure Typed Lambda-Calculus*, 1982
- Beckmann, *Exact bounds for lengths of reductions in typed λ -calculus*, 2001

Let the *order* of a type be the maximum nesting of \rightarrow to the left:

$$\text{ord}(o) = 0 \quad \text{ord}(A \rightarrow B) = \max(\text{ord}(A) + 1, \text{ord}(B))$$

Theorem

Let t be a simply typed λ -term. If all subterms of t have types of order $\leq k$, then the maximum reduction length for t is at most $2_k(O(\text{size}(t)))$ where $2_0(x) = x$ and $2_{n+1}(x) = 2^{2_n(x)}$.

\rightsquigarrow tower of exponentials of height k

$$\underbrace{\overbrace{(\lambda f. \lambda x. f(f x))}^{\text{type of order } k+1} \dots \overbrace{(\lambda f. \lambda x. f(f x))}^{\text{type } (o \rightarrow o) \rightarrow (o \rightarrow o) \text{ of order } 2}}_{k \text{ times}} \longrightarrow_\beta^* \text{something of size } \Theta(2_k(1))$$

+ Asada et al., *Almost Every Simply Typed Lambda-Term Has a Long Beta-Reduction Sequence*, 2017

Reduction length vs complexity of convertibility

$\text{ord}(o) = 0$ and $\text{ord}(A \rightarrow B) = \max(\text{ord}(A) + 1, \text{ord}(B))$

Theorem

Let t be a simply typed λ -term. If all subterms of t have types of order $\leq k$, then the maximum reduction length for t is at most $2_k(O(\text{size}(t)))$ where $2_0(x) = x$ and $2_{n+1}(x) = 2^{2^n(x)}$.

Compute normal forms and compare \rightsquigarrow check $=_\beta$ in time $2_k(\text{poly}(\text{size}(t)))$ i.e. $k\text{-ExpTime}$

Reduction length vs complexity of convertibility

$\text{ord}(o) = 0$ and $\text{ord}(A \rightarrow B) = \max(\text{ord}(A) + 1, \text{ord}(B))$

Theorem

Let t be a simply typed λ -term. If all subterms of t have types of order $\leq k$, then the maximum reduction length for t is at most $2_k(O(\text{size}(t)))$ where $2_0(x) = x$ and $2_{n+1}(x) = 2^{2^n(x)}$.

Compute normal forms and compare \rightsquigarrow check $=_\beta$ in time $2_k(\text{poly}(\text{size}(t)))$ i.e. $k\text{-ExpTime}$

But one can do better! Let $\text{Bool} = o \rightarrow o \rightarrow o$, $\text{true} = \lambda x. \lambda y. x$ and $\text{false} = \lambda x. \lambda y. y$

Theorem (Terui 2012)

Fix $k \in \mathbb{N}$. Given a simply typed λ -term $t : \text{Bool}$ whose subterms have types of order $\leq 2k + 2$ (resp. $\leq 2k + 3$), checking whether $t =_\beta \text{true}$ is complete for $k\text{-ExpTime}$ (resp. $k\text{-ExpSpace}$).

Reduction length vs complexity of convertibility

$\text{ord}(o) = 0$ and $\text{ord}(A \rightarrow B) = \max(\text{ord}(A) + 1, \text{ord}(B))$

Theorem

Let t be a simply typed λ -term. If all subterms of t have types of order $\leq k$, then the maximum reduction length for t is at most $2_k(O(\text{size}(t)))$ where $2_0(x) = x$ and $2_{n+1}(x) = 2^{2^n(x)}$.

Compute normal forms and compare \rightsquigarrow check $=_\beta$ in time $2_k(\text{poly}(\text{size}(t)))$ i.e. $k\text{-ExpTime}$

But one can do better! Let $\text{Bool} = o \rightarrow o \rightarrow o$, $\text{true} = \lambda x. \lambda y. x$ and $\text{false} = \lambda x. \lambda y. y$

Theorem (Terui 2012)

Fix $k \in \mathbb{N}$. Given a simply typed λ -term $t : \text{Bool}$ whose subterms have types of order $\leq 2k + 2$ (resp. $\leq 2k + 3$), checking whether $t =_\beta \text{true}$ is complete for $k\text{-ExpTime}$ (resp. $k\text{-ExpSpace}$).

Corollary (of the hardness part; originally from Statman 198X)

β -convertibility of arbitrary simply typed λ -terms is non-elementary, i.e. $\notin \bigcup_{k \in \mathbb{N}} k\text{-ExpTime}$.

How hard is it to decide β -convertibility?

Theorem (Statman 198X)

β -convertibility of arbitrary simply typed λ -terms is non-elementary, i.e. $\notin \bigcup_{k \in \mathbb{N}} k\text{-ExpTime}$.

At the same time, we can check it naively in $2_{\text{order}(\text{poly}(\text{size}))}$:

tower of exponentials of “reasonably” increasing height, “just beyond” elementary

So can we be more precise?

How hard is it to decide β -convertibility?

Theorem (Statman 198X)

β -convertibility of arbitrary simply typed λ -terms is non-elementary, i.e. $\notin \bigcup_{k \in \mathbb{N}} k\text{-ExpTime}$.

At the same time, we can check it naively in $2_{\text{order}}(\text{poly}(\text{size}))$:

tower of exponentials of “reasonably” increasing height, “just beyond” elementary

So can we be more precise?

Definition (Schmitz 2016 – motivated by several problems in logic and verification)

$\text{TOWER} = \bigcup_{f \text{ elementary}} \text{DTIME}(2_{f(n)}(1))$ i.e. tower of exp of elementary height

TOWER-completeness is defined w.r.t. elementary reductions.

Theorem

β -convertibility of arbitrary simply typed λ -terms is TOWER-complete.

Simply typed β -convertibility is TOWER-complete

Definition (Schmitz 2016)

$\text{TOWER} = \bigcup_{f \text{ elementary}} \text{DTIME}(2_{f(n)}(1))$ i.e. tower of exponentials of elementary height

Theorem (folklore, refining Statman 198X)

β -convertibility of arbitrary simply typed λ -terms is TOWER-complete.

- Membership in TOWER: naive algorithm (compare normal forms) is OK
- TOWER-hard: reduction from “higher-order quantified boolean formulas” (Statman)
e.g. $\forall f : \text{Bool} \rightarrow \text{Bool}. (\exists x : \text{Bool}. f(x)) \Rightarrow (\exists y : \text{Bool}. f(\text{not}(y)))$ is true

Simply typed β -convertibility is TOWER-complete

Definition (Schmitz 2016)

$\text{TOWER} = \bigcup_{f \text{ elementary}} \text{DTIME}(2_{f(n)}(1))$ i.e. tower of exponentials of elementary height

Theorem (folklore, refining Statman 198X)

β -convertibility of arbitrary simply typed λ -terms is TOWER-complete.

- Membership in TOWER: naive algorithm (compare normal forms) is OK
- TOWER-hard: reduction from “higher-order quantified boolean formulas” (Statman)
 - e.g. $\forall f : \text{Bool} \rightarrow \text{Bool}. (\exists x : \text{Bool}. f(x)) \Rightarrow (\exists y : \text{Bool}. f(\text{not}(y)))$ is true
 - First supposed to appear in a never-existing paper [Fisher & Meyer 1975]
 - [Mairson 1992] gave a proof that it’s non-elementary by simulating a Turing machine
 - Explicit TOWER-hardness: Chistikov, Haase, Hadizadeh & Mansutti,

Higher-Order Quantified Boolean Satisfiability, 2022

Simply typed β -convertibility is TOWER-complete

Definition (Schmitz 2016)

$TOWER = \bigcup_{f \text{ elementary}} DTIME(2_{f(n)}(1))$ i.e. tower of exponentials of elementary height

Theorem (folklore, refining Statman 198X)

β -convertibility of arbitrary simply typed λ -terms is TOWER-complete.

- Membership in TOWER: naive algorithm (compare normal forms) is OK
- TOWER-hard: reduction from “higher-order quantified boolean formulas” (Statman)
 - e.g. $\forall f : Bool \rightarrow Bool. (\exists x : Bool. f(x)) \Rightarrow (\exists y : Bool. f(\text{not}(y)))$ is true
 - First supposed to appear in a never-existing paper [Fisher & Meyer 1975]
 - [Mairson 1992] gave a proof that it's non-elementary by simulating a Turing machine
 - Explicit TOWER-hardness: Chistikov, Haase, Hadizadeh & Mansutti,

Higher-Order Quantified Boolean Satisfiability, 2022

<https://cstheory.stackexchange.com/questions/34883/>

reference-request-deciding-validity-of-higher-order-quantified-boolean-formulas

Motivating the safety condition

The *safety* restriction on simply typed λ -terms comes from the theory of *higher-order recursion schemes* which generate infinite trees

(also the motivation of Asada et al.'s work on average-case reduction length)

simply typed λ -terms + let rec \equiv collapsible pushdown automata (late 2000s tech)

safe λ -terms + let rec \equiv higher-order pushdown automata (from 1980s)

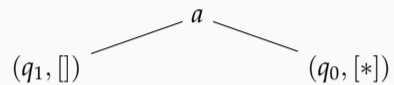
[Damm '82; Knapkik, Niwiński & Urzyczyn '02; Salvati & Walukiewicz '12]

Example on next slide, but it's not important for the rest of the talk, just for motivation

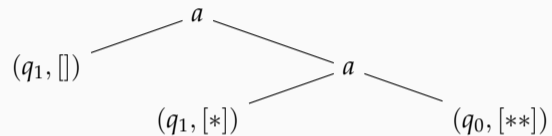
Generating infinite trees: automata vs λ -calculus

(q_0, \square)

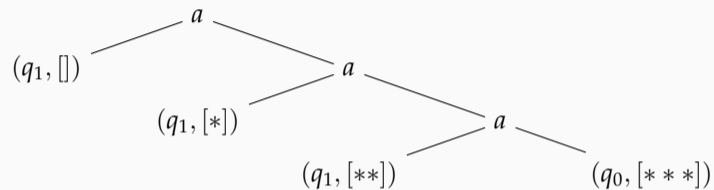
Generating infinite trees: automata vs λ -calculus



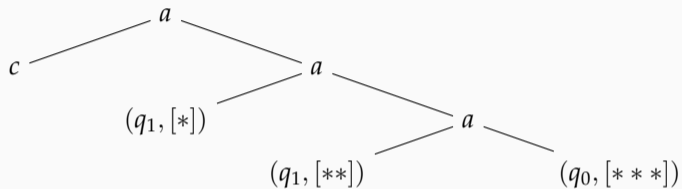
Generating infinite trees: automata vs λ -calculus



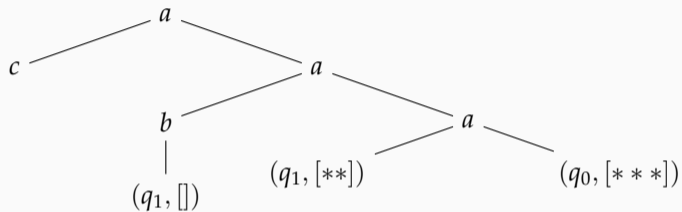
Generating infinite trees: automata vs λ -calculus



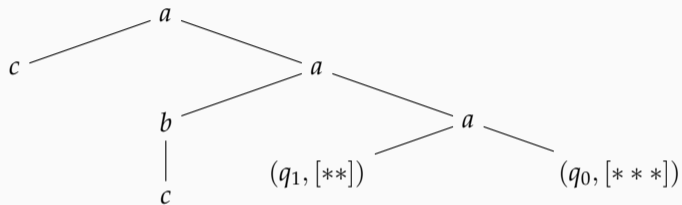
Generating infinite trees: automata vs λ -calculus



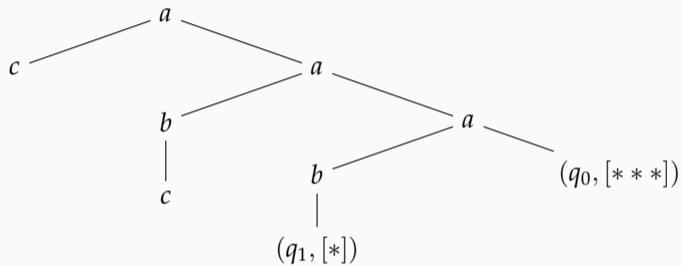
Generating infinite trees: automata vs λ -calculus



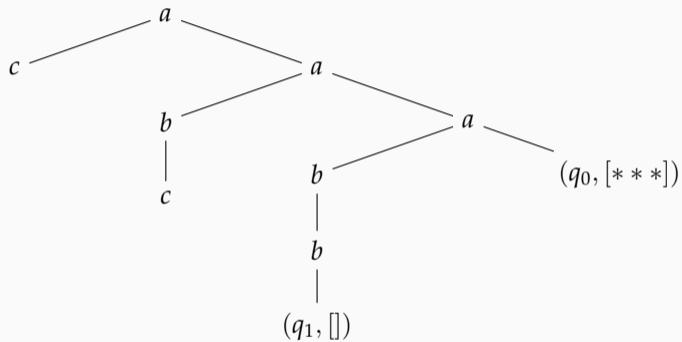
Generating infinite trees: automata vs λ -calculus



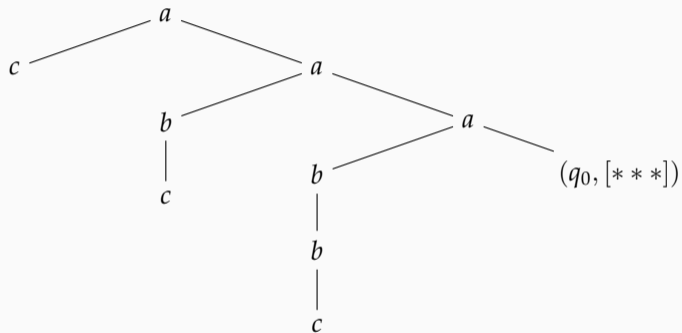
Generating infinite trees: automata vs λ -calculus



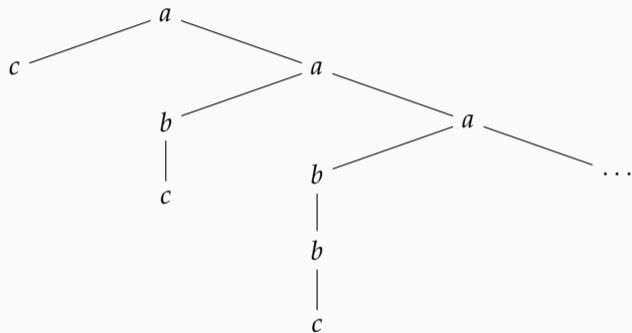
Generating infinite trees: automata vs λ -calculus



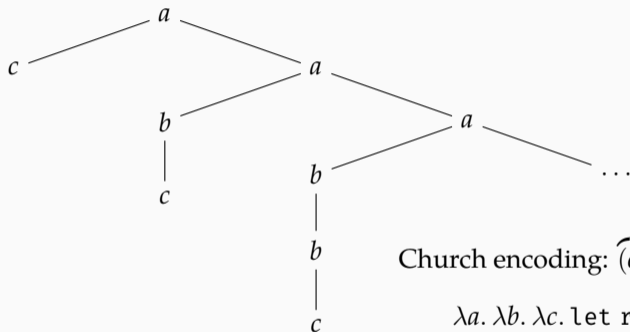
Generating infinite trees: automata vs λ -calculus



Generating infinite trees: automata vs λ -calculus



Generating infinite trees: automata vs λ -calculus



Church encoding: $\overbrace{(o \rightarrow o \rightarrow o)}^a \rightarrow \overbrace{(o \rightarrow o)}^b \rightarrow \overset{c}{\downarrow} o \rightarrow o$

$\lambda a. \lambda b. \lambda c. \text{let rec } f = \lambda x. a x (f(b x)) \text{ in } f c$

The safe λ -calculus (without let rec)

Definition (Blum & Ong 2009)

A simply typed λ -term t is *unsafe* if it contains a subterm t' such that

- t' contains some x as a free variable with $\text{ord}(x) < \text{ord}(t')$
- t' is in “unsafe position” in u – e.g. t' is an argument to some application

$\lambda f^{(o \rightarrow o) \rightarrow o}. f(\lambda x^o. f(\lambda y^o. y))$ is safe

$\lambda f^{(o \rightarrow o) \rightarrow o}. f(\lambda x^o. f(\lambda y^o. \overset{\text{type } o \rightsquigarrow \text{order } 0}{\downarrow} x)))$ is unsafe!
type $(o \rightarrow o) \rightsquigarrow$ order 1

The safe λ -calculus (without let rec)

Definition (Blum & Ong 2009)

A simply typed λ -term t is *unsafe* if it contains a subterm t' such that

- t' contains some x as a free variable with $\text{ord}(x) < \text{ord}(t')$
- t' is in “unsafe position” in u – e.g. t' is an argument to some application

$\lambda f^{(o \rightarrow o) \rightarrow o}. f(\lambda x^o. f(\lambda y^o. y))$ is safe

$\lambda f^{(o \rightarrow o) \rightarrow o}. f(\lambda x^o. f(\lambda y^o. \overset{\text{type } o \rightsquigarrow \text{order } 0}{\downarrow} x)))$ is unsafe!
type $(o \rightarrow o) \rightsquigarrow$ order 1

Theorem (Blum & Ong 2009)

β -convertibility is PSPACE-hard on safe λ -terms.

By reduction from Quantified Boolean Formulas...

but they don't manage to encode *higher-order* QBF without violating safety!

Complexity of β -convertibility in the safe fragment

Theorem (Blum & Ong 2009)

β -convertibility is PSPACE-hard on safe λ -terms.

They also remark:

Because the safety condition restricts expressivity in a non-trivial way, one can reasonably expect the beta-eta equivalence problem to have a lower complexity in the safe case than in the normal case; this intuition is strengthened by our failed attempt to encode [higher-order QBF] in the safe lambda calculus. No upper bounds is known at present. On the other hand our PSPACE-hardness result is probably a coarse lower bound; it would be interesting to know whether we also have EXPTIME-hardness.

Complexity of β -convertibility in the safe fragment

Theorem (Blum & Ong 2009)

β -convertibility is PSPACE-hard on safe λ -terms.

They also remark:

Because the safety condition restricts expressivity in a non-trivial way, one can reasonably expect the beta-eta equivalence problem to have a lower complexity in the safe case than in the normal case; this intuition is strengthened by our failed attempt to encode [higher-order QBF] in the safe lambda calculus. No upper bounds is known at present. On the other hand our PSPACE-hardness result is probably a coarse lower bound; it would be interesting to know whether we also have EXPTIME-hardness.

Theorem (new!)

β -convertibility is TOWER-complete on safe λ -terms.

So it's not any easier in the safe case than in the normal case! (at this level of precision)

Our new TOWER-hardness proof

Theorem (new!)

β -convertibility is TOWER-complete on safe λ -terms.

Blum and Ong didn't manage to reduce from higher-order QBF

\rightsquigarrow we take another approach: a reduction from *star-free expression* emptiness

$$E, E' ::= \emptyset \mid \overbrace{\varepsilon}^{\text{empty string}} \mid \underbrace{a}_{\text{letter in a finite alphabet } \Sigma} \mid E \cup E' \mid \overbrace{E \cdot E'}^{\text{concatenation}} \mid \underbrace{\neg E}_{\text{complement}} \quad \rightsquigarrow \quad \llbracket E \rrbracket \subseteq \Sigma^*$$

(Remark: *star-free languages* = a subclass of regular languages also characterized by aperiodic monoids, first-order logic over finite models, linear temporal logic, ...)

Our new TOWER-hardness proof

Theorem (new!)

β -convertibility is TOWER-complete on safe λ -terms.

Blum and Ong didn't manage to reduce from higher-order QBF

\rightsquigarrow we take another approach: a reduction from *star-free expression* emptiness

$$E, E' ::= \emptyset \mid \overbrace{\varepsilon}^{\text{empty string}} \mid \underbrace{a}_{\text{letter in a finite alphabet } \Sigma} \mid E \cup E' \mid \overbrace{E \cdot E'}^{\text{concatenation}} \mid \underbrace{\neg E}_{\text{complement}} \quad \rightsquigarrow \quad \llbracket E \rrbracket \subseteq \Sigma^*$$

(Remark: *star-free languages* = a subclass of regular languages also characterized by aperiodic monoids, first-order logic over finite models, linear temporal logic, ...)

Theorem (Stockmeyer & Meyer 1973, revisited by Schmitz 2016)

Given a star-free expression E , it is TOWER-complete to decide whether $\llbracket E \rrbracket = \emptyset$.

source of complexity: alternation \cdot vs \neg (cf. dot-depth / Straubing–Thérien hierarchy)

Turning star-free expressions into safe λ -terms

$$E, E' ::= \emptyset \mid \varepsilon \mid a \mid E \cup E' \mid E \cdot E' \mid \neg E \quad \rightsquigarrow \quad \llbracket E \rrbracket \subseteq \Sigma^*$$

Lemma

Any expression E can be turned in P_{TIME} into an equivalent safe term $t_E : \text{Str}_\Sigma[A] \rightarrow \text{Bool}$.

$$\begin{aligned} \text{Str}_\Sigma[A] &= \overbrace{(A \rightarrow A) \rightarrow \cdots \rightarrow (A \rightarrow A)}^{|\Sigma| \text{ times}} \rightarrow A \rightarrow A \\ abb \in \Sigma^* = \{a, b\}^* &\rightsquigarrow \overline{abb} = \lambda f_a. \lambda f_b. \lambda x. f_a (f_b (f_b x)) : \text{Str}_\Sigma[A] \quad \text{for any } A \end{aligned}$$

Turning star-free expressions into safe λ -terms

$$E, E' ::= \emptyset \mid \varepsilon \mid a \mid E \cup E' \mid E \cdot E' \mid \neg E \quad \rightsquigarrow \quad \llbracket E \rrbracket \subseteq \Sigma^*$$

Lemma

Any expression E can be turned in P_{TIME} into an equivalent safe term $t_E : \text{Str}_{\Sigma}[A] \rightarrow \text{Bool}$.

$$\begin{aligned} \text{Str}_{\Sigma}[A] &= \overbrace{(A \rightarrow A) \rightarrow \cdots \rightarrow (A \rightarrow A)}^{|\Sigma| \text{ times}} \rightarrow A \rightarrow A \\ abb \in \Sigma^* = \{a, b\}^* &\rightsquigarrow \overline{abb} = \lambda f_a. \lambda f_b. \lambda x. f_a (f_b (f_b x)) : \text{Str}_{\Sigma}[A] \quad \text{for any } A \end{aligned}$$

Theorem (Hillebrand & Kanellakis 1996 – main inspiration for the above)

A language $L \subseteq \Sigma^*$ can be defined by some simply typed λ -term $t : \text{Str}_{\Sigma}[A] \rightarrow \text{Bool}$ (where A may be chosen depending on L) if and only if it is regular.

Ex. (not star-free!): $t = \lambda s. s \text{ id not true} : \text{Str}_{\{a,b\}}[\text{Bool}] \rightarrow \text{Bool}$ (even number of bs)

$$t \overline{abb} \longrightarrow_{\beta} \overline{abb} \text{ id not true} \longrightarrow_{\beta} \text{id (not (not true))} \longrightarrow_{\beta} \text{true}$$

Turning star-free expressions into safe λ -terms

Lemma

Any expression E can be turned in P_{TIME} into an equivalent safe term $t_E : \text{Str}_\Sigma[A] \rightarrow \text{Bool}$.

The proof is inspired by my research with Pradic on “Implicit automata in typed λ -calculi”

- Especially our results on *transducers* (automata computing string-to-string functions)
- e.g. the inductive case for translating $E \cdot E'$ uses a safe λ -term computing

$$123 \mapsto \square 123 \# 1 \square 23 \# 12 \square 3 \# 123 \square$$

a typical *polyregular function* (cf. Bojańczyk’s LICS’22 invited paper)

Then we need a bit more work to apply t_E to all “short enough” words and take the disjunction. Finally we get a term of type Bool which is true when $\llbracket E \rrbracket \neq \emptyset$, hence:

Theorem

Given a safe λ -term $t : \text{Bool}$, it is $TOWER$ -complete to decide whether $t =_\beta \text{true}$.

Theorem

Given a safe λ -term $t : \text{Bool}$, it is TOWER-complete to decide whether $t =_{\beta} \text{true}$.

- The same holds for the simply typed λ -calculus (of which the safe λ -calculus is a fragment), “traditionally” proved via higher-order quantified boolean formulas
- This does not work in the safe case; instead we leverage connections between automata theory and λ -calculus

Theorem

Given a safe λ -term $t : \text{Bool}$, it is TOWER-complete to decide whether $t =_{\beta} \text{true}$.

- The same holds for the simply typed λ -calculus (of which the safe λ -calculus is a fragment), “traditionally” proved via higher-order quantified boolean formulas
- This does not work in the safe case; instead we leverage connections between automata theory and λ -calculus

Final remark: Pradic and I have characterized star-free languages using *planar* λ -terms (ICALP 2020). For this result, translating star-free expressions didn't work (instead we used the Krohn–Rhodes decomposition theorem).

Theorem

Given a safe λ -term $t : \text{Bool}$, it is TOWER-complete to decide whether $t =_{\beta} \text{true}$.

- The same holds for the simply typed λ -calculus (of which the safe λ -calculus is a fragment), “traditionally” proved via higher-order quantified boolean formulas
- This does not work in the safe case; instead we leverage connections between automata theory and λ -calculus

Final remark: Pradic and I have characterized star-free languages using *planar* λ -terms (ICALP 2020). For this result, translating star-free expressions didn't work (instead we used the Krohn–Rhodes decomposition theorem).