Combinatorics of explicit substitutions

Maciej Bendkowski[1] Pierre Lescanne[2]

[1]Jagiellonian University in Kraków
[2]École normale supérieure de Lyon

Computational Logic and Applications
Paris, May 25th 2018

Outline

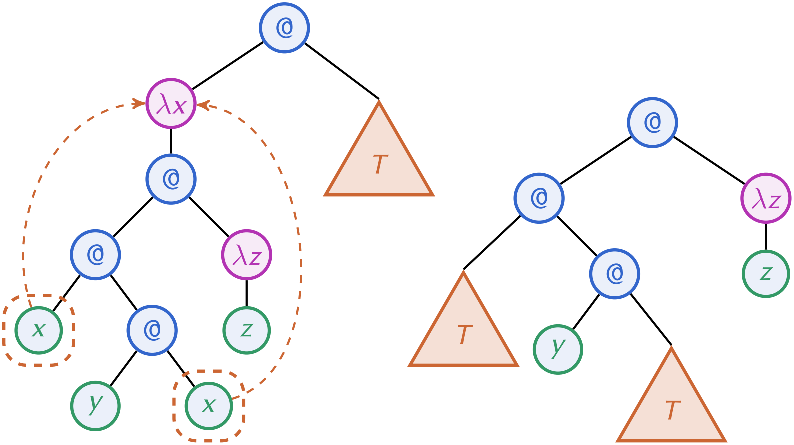
λ-calculus and substitution resolution

$\lambda\upsilon$-calculus and explicit substitutions

Our contribution

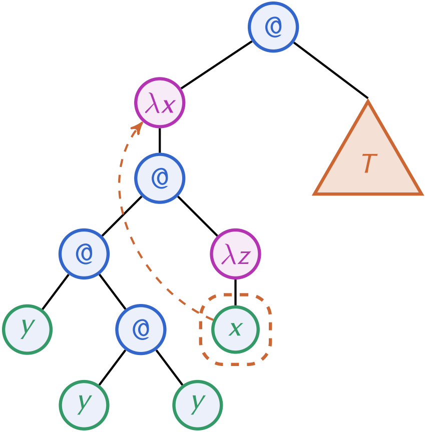
β-reduction in λ-calculus

Substitution of terms for variables forms the essence of β-reduction in λ-calculus. For instance, $(\lambda x.x(yx)(\lambda z.z))T \to_\beta T(yT)(\lambda z.z)$:



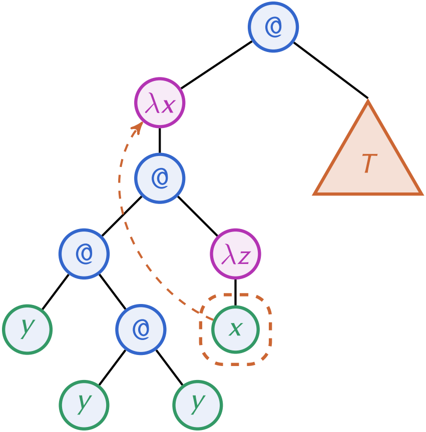
β-reduction in λ-calculus (ii)

However, what if we substitute T with *free variables* under an abstraction? For instance, what if z occurs freely in T, as in the following example:



β-reduction in λ-calculus (ii)

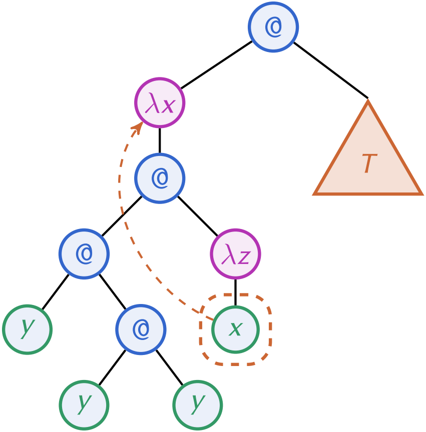
However, what if we substitute T with *free variables* under an abstraction? For instance, what if z occurs freely in T, as in the following example:



- We want to avoid the *capture* of variable z in T;

β-reduction in λ-calculus (ii)

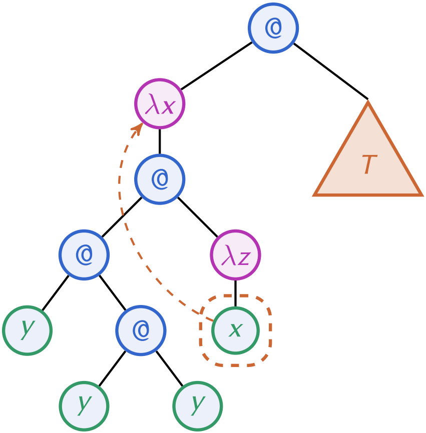
However, what if we substitute T with *free variables* under an abstraction? For instance, what if z occurs freely in T, as in the following example:



- We want to avoid the *capture* of variable z in T;
- Requires keeping track of variable names and (sometimes) their *renaming*;

β-reduction in λ-calculus (ii)

However, what if we substitute T with *free variables* under an abstraction? For instance, what if z occurs freely in T, as in the following example:



- We want to avoid the *capture* of variable z in T;
- Requires keeping track of variable names and (sometimes) their *renaming*;
- In principle feasible, but *in practice* it is not a trivial operation to execute.

β-reduction and computational effectiveness

Postulate (essentially due to de Bruijn '78)

The number of β-reductions required to normalise a given term (in other words, evaluate the encoded computation) *does not* quite reflect the computational effort required to *carry out* the computation.

β-reduction and computational effectiveness

Postulate (essentially due to de Bruijn '78)

The number of β-reductions required to normalise a given term (in other words, evaluate the encoded computation) *does not* quite reflect the computational effort required to *carry out* the computation.

- It hides the non-trivial *details* of capture-avoiding substitution;

β-reduction and computational effectiveness

Postulate (essentially due to de Bruijn '78)

The number of β-reductions required to normalise a given term (in other words, evaluate the encoded computation) *does not* quite reflect the computational effort required to *carry out* the computation.

- It hides the non-trivial *details* of capture-avoiding substitution;
- It does not reflect the cost of *substitution resolution*;

β-reduction and computational effectiveness

Postulate (essentially due to de Bruijn '78)

The number of β-reductions required to normalise a given term (in other words, evaluate the encoded computation) *does not* quite reflect the computational effort required to *carry out* the computation.

- It hides the non-trivial *details* of capture-avoiding substitution;
- It does not reflect the cost of *substitution resolution*;
- and finally, it hides the details of *evaluation strategies*.

β-reduction and computational effectiveness

Postulate (essentially due to de Bruijn '78)

The number of β-reductions required to normalise a given term (in other words, evaluate the encoded computation) *does not* quite reflect the computational effort required to *carry out* the computation.

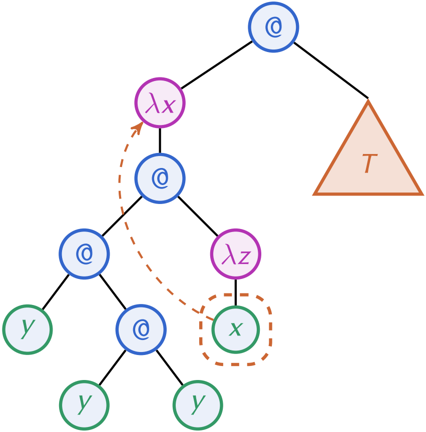
- It hides the non-trivial *details* of capture-avoiding substitution;
- It does not reflect the cost of *substitution resolution*;
- and finally, it hides the details of *evaluation strategies*.

. . . and that is due to the <u>epitheoretic substitution</u> operation.

β-reduction and computational effectiveness (ii)

Issue I

β-reduction hides the non-trivial *details* of capture-avoiding substitution. Variable renaming can take time linear in the term size.

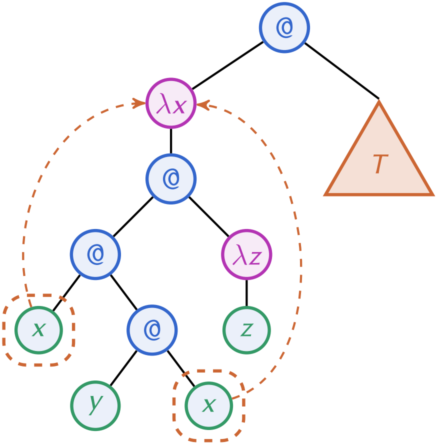


β-reduction and computational effectiveness (iii)

Issue II

β-reduction does not reflect the cost of *substitution resolution*.

Variable search can take time linear in the term size.



β-reduction and computational effectiveness (iv)

Issue III

β-reduction hides the details of *evaluation strategies*. What if the substitution is carried out non-strictly, e.g. it is suspended and evaluated *on demand* (perhaps even never)?

Real-life[1] example of an infinite lists of Fibonacci numbers:

```
fibs = 0 : 1 : next fibs
  where
    next (a : b : xs) = (a + b) : next (b : xs)
```

[1] Is this the *real life*? Is this just fantasy? (. . .) [Mercury et al. '75]

Effectuation of substitution

In order to effectuate substitution, its resolution should be *internalised* into the calculus itself. Possible solutions:

- investigate combinatory logic, or

Effectuation of substitution

In order to effectuate substitution, its resolution should be *internalised* into the calculus itself. Possible solutions:

- investigate combinatory logic, or
- investigate *calculi of explicit substitution*;

Effectuation of substitution

In order to effectuate substitution, its resolution should be *internalised* into the calculus itself. Possible solutions:

- investigate combinatory logic, or
- investigate *calculi of explicit substitution*;

Quest

Investigate quantitative aspects of substitution. For instance,

Effectuation of substitution

In order to effectuate substitution, its resolution should be *internalised* into the calculus itself. Possible solutions:

- investigate combinatory logic, or
- investigate *calculi of explicit substitution*;

Quest

Investigate quantitative aspects of substitution. For instance,

- What is the average-case cost of resolving substitutions in computations (terms) of size n? How does it change depending on the assumed evaluation strategy? What contributes to its execution time?

Effectuation of substitution

In order to effectuate substitution, its resolution should be *internalised* into the calculus itself. Possible solutions:

- investigate combinatory logic, or
- investigate *calculi of explicit substitution*;

Quest

Investigate quantitative aspects of substitution. For instance,

- What is the average-case cost of resolving substitutions in computations (terms) of size n? How does it change depending on the assumed evaluation strategy? What contributes to its execution time?
- What is the *average-case complexity* of abstract machines executing (terminating) computations of considered calculi? Is it possible to optimise them based on the structure of typical computations?

Reflections on combinatory logic

Normal-order reduction

If a term (combinator) T is normalisable, then the iterative contraction of the leftmost-outermost redex leads to the (unique) normal form of T.

$$
\begin{array}{lrcl}
Sxyz \to xz(yz) & R_0 & = & S \oplus K \oplus KR_0 \oplus SR_0 \oplus SR_0R_0 \\
Kxy \to x & R_1 & = & SR_1 \oplus KR_1 \oplus SR_0R_1 \oplus SR_1R_0 \oplus KR_0C \oplus \\
 & & & KSCR_0 \oplus KKCR_0 \oplus KSCR_0R_0 \oplus K(SR_0)CR_0 \\
 & & & \oplus SSSR_0 \oplus SSKR_0 \oplus SS(SR_0)R_0 \\
 & R_2 & = & \ldots
\end{array}
$$

Figure: Rewriting rules of SK-combinators and the corresponding normal-order reduction grammars.

Reflections on combinatory logic (ii)

Theorem (B., Grygiel and Zaionc '17)

For each $k \geq 1$, the asymptotic density $\mu(R_k/C)$ of combinators reducing in k normal-order reduction steps in the set of all combinators is *positive*.

In particular, we have:

k	$\mu(R_k/C)$
0	0.
1	0.08961
2	0.06417
3	0.05010
4	0.04131
5	0.03570
6	0.03119
7	0.02798

Problem

How to *port* these results to the realm of λ-calculus having an *external* substitution?

... use *explicit substitutions*!

Outline

λ-calculus and substitution resolution

$\lambda\upsilon$-calculus and explicit substitutions

Our contribution

$\lambda\upsilon$-calculus (lambda upsilon calculus)

A simplistic calculus of explicit substitutions due to Lescanne '94.

$$\mathcal{T} ::= \mathcal{N} \mid \lambda\mathcal{T} \mid \mathcal{T}\mathcal{T} \mid \mathcal{T}[\mathcal{S}]$$
$$\mathcal{S} ::= \mathcal{T}/ \mid \Uparrow(\mathcal{S}) \mid \uparrow$$
$$\mathcal{N} ::= \underline{0} \mid \mathbf{S}\mathcal{N}.$$

Figure: Terms of $\lambda\upsilon$-calculus.

$$\begin{aligned}
(\lambda a)b &\to a[b/] && \text{(Beta)}\\
(ab)[s] &\to a[s](b[s]) && \text{(App)}\\
(\lambda a)[s] &\to \lambda(a[\Uparrow(s)]) && \text{(Lambda)}\\
\underline{0}[a/] &\to a && \text{(FVar)}\\
(\mathbf{S}\underline{n})[a/] &\to \underline{n} && \text{(RVar)}\\
\underline{0}[\Uparrow(s)] &\to \underline{0} && \text{(FVarLift)}\\
(\mathbf{S}\underline{n})[\Uparrow(s)] &\to \underline{n}[s][\uparrow] && \text{(RVarLift)}\\
\underline{n}[\uparrow] &\to \mathbf{S}\underline{n}. && \text{(VarShift)}
\end{aligned}$$

Figure: Rewriting rules.

$\lambda\upsilon$-calculus (ii)

Consider the term $K = \lambda x.\lambda y.x$; denoted as $\lambda\lambda\underline{1}$. Certainly, $Kab \to_\beta a$ for each term a (in one step). Note however, that with explicit substitutions we have

$$\begin{aligned}(\lambda\lambda\underline{1})a &\to (\lambda\underline{1})[a/] \\ &\to \lambda(\underline{1}[\Uparrow(a/)]) \\ &\to \lambda\,(\underline{0}[a/][\uparrow]) \\ &\to \lambda\,(a[\uparrow])\,.\end{aligned}$$

The final shift operator guarantees that (potential) free indices are aptly incremented so to avoid potential variable captures. If a is closed, then $a[\uparrow]$ resolves simply to a, as intended.

Outline

λ-calculus and substitution resolution

$\lambda\upsilon$-calculus and explicit substitutions

Our contribution

Enumerative results

Proposition

Let $T(z)$ and $S(z)$ denote the generating functions corresponding to $\lambda\upsilon$-terms and substitutions, respectively. Then,

$$T(z) = \frac{1-\sqrt{1-4z}}{2z} - 1 \quad \text{whereas} \quad S(z) = \frac{1-\sqrt{1-4z}}{2z}\left(\frac{z}{1-z}\right).$$

In consequence

$$[z^n]T(z) = \begin{cases} 0, & \text{for } n = 0 \\ \frac{1}{n+1}\binom{2n}{n}, & \text{otherwise} \end{cases}$$

$$[z^n]S(z) = \begin{cases} 0, & \text{for } n = 0 \\ \sum_{k=0}^{n-1} \frac{1}{k+1}\binom{2k}{k} & \text{otherwise.} \end{cases}$$

Enumerative results – explicit bijection

$$\varphi\left(\begin{array}{ccc} \bullet & & \\ & \diagdown & \\ & & R \end{array}\right) = \lambda\varphi(R)$$

$$\varphi\left(\begin{array}{ccc} & & \bullet \\ & \diagup & \\ L & & \end{array}\right) = \mathbf{S}\underline{\mathrm{n}}$$

$$\textbf{when} \quad \varphi(L) = \underline{\mathrm{n}}$$

$$\varphi\left(\begin{array}{ccc} & & \bullet \\ & \diagup & \\ L & & \end{array}\right) = a[\Uparrow^{n+1}(s)]$$

$$\textbf{when} \quad \varphi(L) = a[\Uparrow^{n}(s)]$$

$$\varphi(\bullet) = \underline{0}$$

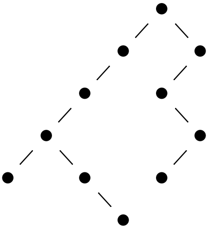
$$\varphi\left(\begin{array}{ccc} & & \bullet \\ & \diagup & \\ \bullet & & \\ & \diagdown & \\ & & R \end{array}\right) = \varphi(R)[\uparrow]$$

$$\varphi\left(\begin{array}{ccccc} & & & & \bullet \\ & & & \diagup & \\ & & \bullet & & \\ & \diagup & & \diagdown & \\ L & & & & R \end{array}\right) = \varphi(L)[\varphi(R)/]$$

$$\varphi\left(\begin{array}{ccccc} & & \bullet & & \\ & \diagup & & \diagdown & \\ L & & & & R \end{array}\right) = \varphi(L)\,\varphi(R)$$

Enumerative results – correspondence example

For instance, $\underline{0}[\Uparrow (\lambda \underline{0}/)]\ \underline{1}[\uparrow]$ corresponds to



Statistical properties

Strict substitution forms

A $\lambda\upsilon$-term t is in *strict substitution form* if there exist two pure (i.e. without explicit substitutions) terms a, b and a sequence $t_1, \ldots, t_n$ of $\lambda\upsilon$-terms such that

$$a[b/] \to t_1 \to \cdots \to t_n = t$$

and none of the above reductions is (Beta).
Otherwise, t is said to be in *lazy substitution form*.

Theorem

Asymptotically almost all $\lambda\upsilon$-terms are in *lazy substitution form*.

Statistical properties

Strict substitution forms

A $\lambda\upsilon$-term t is in *strict substitution form* if there exist two pure (i.e. without explicit substitutions) terms a, b and a sequence $t_1, \ldots, t_n$ of $\lambda\upsilon$-terms such that

$$a[b/] \to t_1 \to \cdots \to t_n = t$$

and none of the above reductions is (Beta).
Otherwise, t is said to be in *lazy substitution form*.

Theorem

Asymptotically almost all $\lambda\upsilon$-terms are in *lazy substitution form*.

Proof.

Idea: Consider the class of terms without *nested closures*. $\square$

Statistical properties (ii)

Substitution suspension

Let s be an substitution and t be a $\lambda\upsilon$-term. Then, s, all its subterms, and all the constructors it contains are said to be *suspended in t* if t contains a subterm in form of $[s]$; in other words, when s occurs under a closure in t.

Theorem

Let X_n be a random variable denoting the number of constructors not suspended under a closure in a random $\lambda\upsilon$-term of size n. Then, the expectation $\mathbb{E}(X_n)$ satisfies

$$\mathbb{E}(X_n) \xrightarrow[n\to\infty]{} \frac{316}{3} \approx 105.33.$$

Statistical properties (iii)

Proposition

Let R be a redex in the $\lambda\upsilon$-calculus and X_n be the corresponding random variable denoting the number of occurrences of R in a random term of size n. Then, after standardisation, X_n admits a limiting Gaussian law.

In particular, we obtain:

Redex	(limit) mean	(limit) variance
(Beta)	$0.046875n$	$0.037354n$
(App)	$0.031250n$	$0.021973n$
(Lambda)	$0.031250n$	$0.025879n$
(VarShift)	$0.015625n$	$0.013916n$
(FVar)	$0.011719n$	$0.011124n$
(FVarLift)	$0.007812n$	$0.007355n$
(RVar)	$0.003906n$	$0.003799n$
(RVarLift)	$0.002604n$	$0.002557n$

Conclusions

- Standard analytic methods provide insight into the structure of random $\lambda\upsilon$-terms, in particular *substitution resolution*;

Conclusions

- Standard analytic methods provide insight into the structure of random $\lambda\upsilon$-terms, in particular *substitution resolution*;
- Typical computations are in a strong sense *lazy*. Substitutions are evaluated non-strictly. In fact, almost all of the computation content is *suspended*;

Conclusions

- Standard analytic methods provide insight into the structure of random $\lambda\upsilon$-terms, in particular *substitution resolution*;
- Typical computations are in a strong sense *lazy*. Substitutions are evaluated non-strictly. In fact, almost all of the computation content is *suspended*;
- Substitution primitives have, on average, an uneven contribution in typical terms. We can exploit that in *micro optimisations* of abstract machines.

Future work

- What is the cost distribution of substitution resolution, given a random term of size n? What primitives are its main contributors?

Future work

- What is the cost distribution of substitution resolution, given a random term of size *n*? What primitives are its main contributors?
- Does normal-order reduction (think of non-strict functional programming languages) has a similar, average-case "distribution shape" as the one for combinatory logic?

Thank you

Thank you for your attention!
(Strict questions and lazy answers)