A puzzle to ponder on α-conversion

- give an upperbound on the #α-renamings needed to β-reduce $((\underline{2}\,\underline{8})\,(\underline{4}\,\underline{9}))\,(\underline{5}\,\underline{7})\,(\underline{4}\,\underline{2})$ to normal form?
- note 1: $\underline{n} := \lambda sz.s^n\,z$ is Church-numeral n
- note 2: application of Church-numerals is exponentiation; $\underline{k}\,\underline{n} \twoheadrightarrow_\beta \underline{n^k}$
- note 3: whether α-conversion is needed in a β-reduction is undecidable

Thoughts on naïvely implementing the $\lambda\beta$-calculus

Vincent van Oostrom

λ-calculus naïvely

$\underline{2} := \lambda sz.s\,(s\,z)$

Church numeral 2

running example, reduces to four

$(\lambda sz.s\,(s\,z))\,\lambda sz.s\,(s\,z)$

λ-calculus naïvely

$\underline{2} := \lambda sz.s\,(s\,z)$

$\underline{2}\,\underline{2}$

λ-calculus naïvely

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

β-reduction with naïve substitution

(not in λx; indiscriminantly in λy)

$\underline{2}\,\underline{2}$

Substitution naïvely (no α)

$$
\begin{array}{rcll}
x[x{:=}N] & := & N & \\
y[x{:=}N] & := & y & (\text{for } x \neq y) \\
(\lambda x.M)[x{:=}N] & := & \lambda x.M & \\
(\lambda y.M)[x{:=}N] & := & \lambda y.M[x{:=}N] & (\text{for } x \neq y) \\
(M_1\, M_2)[x{:=}N] & := & M_1[x{:=}N]\, M_2[x{:=}N] &
\end{array}
$$

```
data Lam = Lam Head [Lam] deriving (Show)
data Head = Var String | Abs String Lam deriving (Show)
subst x s (Lam h l) = let
  (Lam h' l') = case h of
     (Var y)   | x == y -> s
     (Abs y u) | x /= y -> Lam (Abs y (subst x s u)) []
     _                  -> Lam h [] in (Lam h' (l'++(map (subst x s) l)))
```

λ-calculus naïvely

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$

$$\underline{2}\,\underline{2}$$

combinator system

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

$\underline{2}\;\underline{2}$

lifting $\lambda z.s\,(s\,z)$

combinator system

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \rightarrow_\beta M[x{:=}N]$

lifting $\lambda z.s\,(s\,z)$

skeleton $\lambda z.[\,]\,([\,]\,z) \mapsto$ f-symbol Z

maximal free subexpressions s, s

$\underline{2}\,\underline{2}$

combinator system

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

$Z[x,y]\,z \to_\kappa x\,(y\,z)$

$Z[x,y]$ represents $\lambda z.x\,(y\,z)$

$\underline{2}\,\underline{2}$

combinator system

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

$Z[x,y]\,z \to_\kappa x\,(y\,z)$

$\underline{2}\,\underline{2}$

combinator system

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

$\underline{2}\,\underline{2}$

$Z[x,y]\,z \to_\kappa x\,(y\,z)$

lifting $\lambda s.Z[s,s]$

its own skeleton $\mapsto$ f-symbol S

no maximal free subexpressions

combinator system

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
S represents $\underline{2} := \lambda sz.s\,(s\,z)$

$$\underline{2}\,\underline{2}$$

combinator system

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

$Z[x,y]\,z \to_\kappa x\,(y\,z)$

$S\,z \to_\kappa Z[z,z]$

running example, reduces to four

$\underline{2}\,\underline{2}$

$S\,S$

combinator system

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$

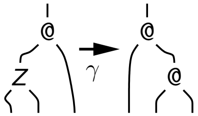
$$\underline{2}\,\underline{2}$$

$$S\,S$$

TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$



$$\underline{2}\,\underline{2}$$

$$S\,S$$

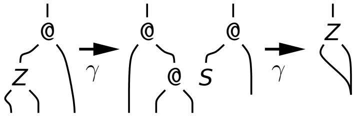
TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$

$$\underline{2}\,\underline{2}$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$

$$S\,S$$

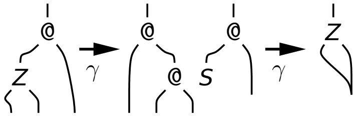


duplication by sharing in rhs

TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$



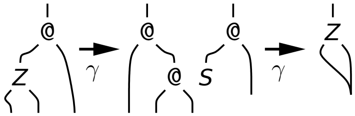
$$\underline{2}\,\underline{2}$$

$$S\,S$$

TGRS

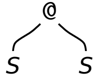
$\underline{2} := \lambda sz.s\,(s\,z)$
$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

$Z[x,y]\,z \to_\kappa x\,(y\,z)$
$S\,z \to_\kappa Z[z,z]$



$\underline{2}\,\underline{2}$

$S\,S$

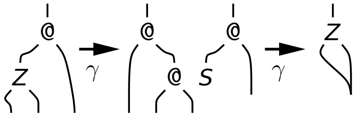


running example, reduces to four

λ-calculus $\iff$ combinator system $\iff$ TGRS

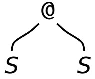
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$



$\underline{2}\,\underline{2}$

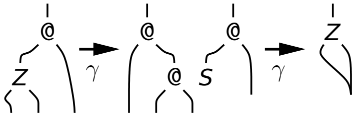
$S\,S$



λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

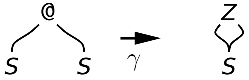
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$



$$\begin{tikzcd} \underline{2}\,\underline{2} \arrow[d, "\beta"] \\ \lambda z.\underline{2}\,(\underline{2}\,z) \end{tikzcd}$$

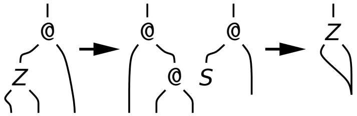
$$\begin{tikzcd} S\,S \arrow[d, "\kappa"] \\ Z[S,S] \end{tikzcd}$$



λ-calculus $\iff$ combinator system $\iff$ TGRS

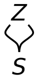
$\underline{2} := \lambda sz.s\,(s\,z)$
$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

$Z[x,y]\,z \to_\kappa x\,(y\,z)$
$S\,z \to_\kappa Z[z,z]$



$\lambda z.\underline{2}\,(\underline{2}\,z)$

$Z[S,S]$



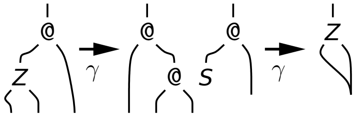
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

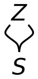
$Z[x,y]\,z \to_\kappa x\,(y\,z)$

$S\,z \to_\kappa Z[z,z]$



$\lambda z.\underline{2}\,(\underline{2}\,z)$

$Z[S,S]$



weak head normal form (under λ)

normal form (Z is stuck)

normal form (Z is stuck)

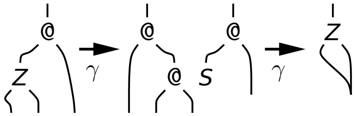
λ-calculus $\iff$ combinator system $\iff$ TGRS

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

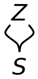
$Z[x,y]\,z \to_\kappa x\,(y\,z)$

$S\,z \to_\kappa Z[z,z]$



$\lambda z.\underline{2}\,(\underline{2}\,z)$

$Z[S,S]$



root-introduce fresh constant

root-introduce fresh constant

root-introduce fresh constant

λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

$\underline{2} := \lambda sz.s\,(s\,z)$

$(\lambda x.M)\,N \to_\beta M[x{:=}N]$

$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$

(z' fresh; think of as constant)

$\lambda z.\underline{2}\,(\underline{2}\,z)$

factor α through β (at root)

$Z[x,y]\,z \to_\kappa x\,(y\,z)$

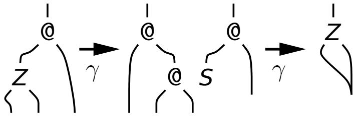
$S\,z \to_\kappa Z[z,z]$

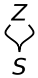
$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$

$S \to_\alpha \lambda z'.S\,z'$

$Z[S,S]$

unstuck combinator (at root)





Z, S-rules as expected (at root)

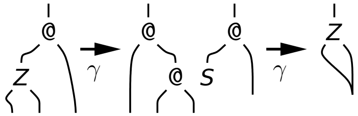
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

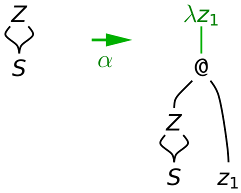
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z.\underline{2}\,(\underline{2}\,z)$$
$$\Big\downarrow \alpha$$
$$\lambda z_1.(\lambda z.\underline{2}\,(\underline{2}\,z))\,z_1$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$Z[S,S]$$
$$\Big\downarrow \alpha$$
$$\lambda z_1.Z[S,S]\,z_1$$



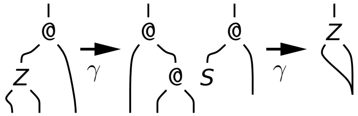


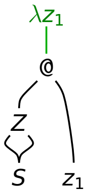
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z_1.(\lambda z.\underline{2}\,(\underline{2}\,z))\,z_1$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$
$$\lambda z_1.Z[S,S]\,z_1$$





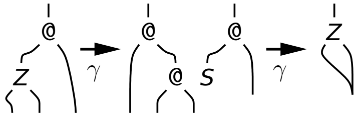
λ-calculus $\iff$ combinator system $\iff$ TGRS

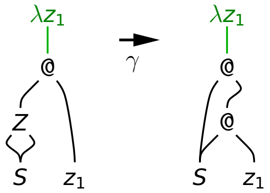
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z_1.(\lambda z.\underline{2}\,(\underline{2}\,z))\,z_1$$
$$\Big\downarrow \beta$$
$$\lambda z_1.\underline{2}\,(\underline{2}\,z_1)$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda z_1.Z[S,S]\,z_1$$
$$\Big\downarrow \kappa$$
$$\lambda z_1.S\,(S\,z_1)$$





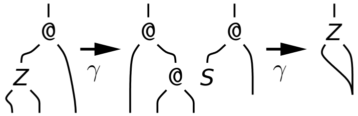
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

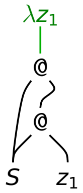
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z_1.\underline{2}\,(\underline{2}\,z_1)$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda z_1.S\,(S\,z_1)$$



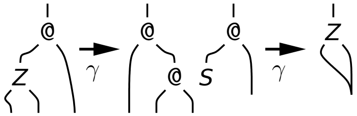


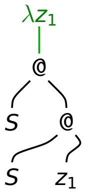
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z_1.\underline{2}\,(\underline{2}\,z_1)$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$
$$\lambda z_1.S\,(S\,z_1)$$





unshare constructor of redex

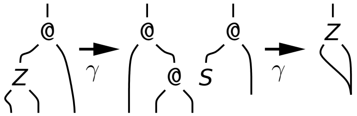
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

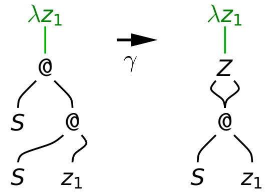
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z_1.\underline{2}\,(\underline{2}\,z_1)$$
$$\Big\downarrow \beta$$
$$\lambda z_1.\lambda z.\underline{2}\,z_1\,(\underline{2}\,z_1\,z)$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda z_1.S\,(S\,z_1)$$
$$\Big\downarrow \kappa$$
$$\lambda z_1.Z[S\,z_1, S\,z_1]$$



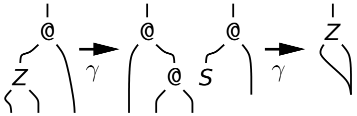


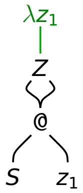
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z_1.\lambda z.\underline{2}\,z_1\,(\underline{2}\,z_1\,z)$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$
$$\lambda z_1.Z[S\,z_1, S\,z_1]$$





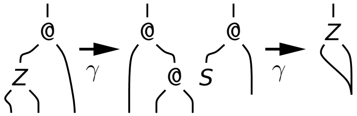
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

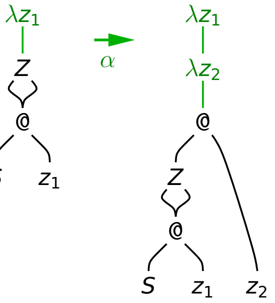
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z_1.\lambda z.\underline{2}\,z_1\,(\underline{2}\,z_1\,z)$$
$$\big\downarrow \alpha$$
$$\lambda z_1 z_2.(\lambda z.\underline{2}\,z_1\,(\underline{2}\,z_1\,z))\,z_2$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda z_1.Z[S\,z_1, S\,z_1]$$
$$\big\downarrow \alpha$$
$$\lambda z_1 z_2.Z[S\,z_1, S\,z_1]\,z_2$$





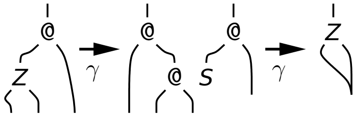
λ-calculus $\iff$ combinator system $\iff$ TGRS

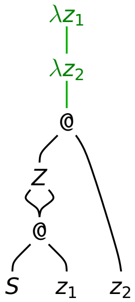
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z_1 z_2.(\lambda z.\underline{2}\,z_1\,(\underline{2}\,z_1\,z))\,z_2$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda z_1 z_2.Z[S\,z_1, S\,z_1]\,z_2$$





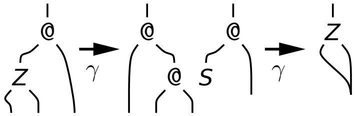
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

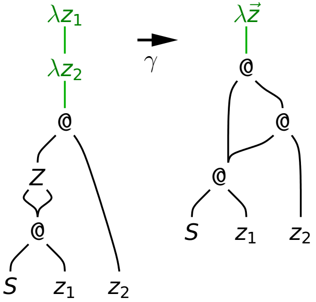
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda z_1 z_2.(\lambda z.\underline{2}\,z_1\,(\underline{2}\,z_1\,z))\,z_2$$
$$\Big\downarrow \beta$$
$$\lambda\vec{z}.\underline{2}\,z_1\,(\underline{2}\,z_1\,z_2)$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda z_1 z_2.Z[S\,z_1, S\,z_1]\,z_2$$
$$\Big\downarrow \kappa$$
$$\lambda\vec{z}.S\,z_1\,(S\,z_1\,z_2)$$



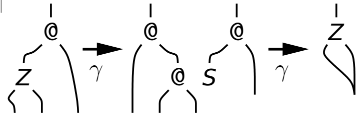


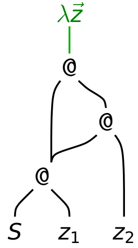
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda\vec{z}.\underline{2}\,z_1\,(\underline{2}\,z_1\,z_2)$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$
$$\lambda\vec{z}.S\,z_1\,(S\,z_1\,z_2)$$





λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

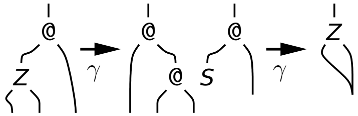
$$\lambda\vec{z}.\underline{2}\,z_1\,(\underline{2}\,z_1\,z_2)$$
$$\big\downarrow \mathsf{f}\beta$$
$$\lambda\vec{z}.(\lambda z.z_1\,(z_1\,z))\,((\lambda z.z_1\,(z_1\,z))\,z_2)$$

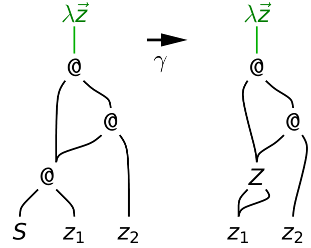
parallel β (weak family)

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda\vec{z}.S\,z_1\,(S\,z_1\,z_2)$$
$$\big\downarrow \mathsf{f}\kappa$$
$$\lambda\vec{z}.Z[z_1,z_1]\,(Z[z_1,z_1]\,z_2)$$

parallel κ (family)





contract shared S-redex

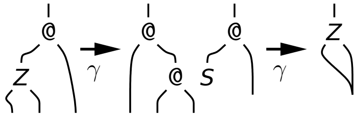
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

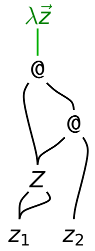
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda\vec{z}.(\lambda z.z_1\,(z_1\,z))\,((\lambda z.z_1\,(z_1\,z))\,z_2)$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda\vec{z}.Z[z_1,z_1]\,(Z[z_1,z_1]\,z_2)$$





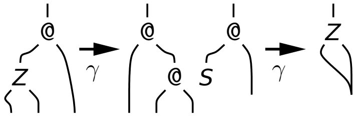
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

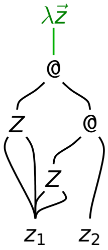
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda\vec{z}.(\lambda z.z_1\,(z_1\,z))\,((\lambda z.z_1\,(z_1\,z))\,z_2)$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda\vec{z}.Z[z_1,z_1]\,(Z[z_1,z_1]\,z_2)$$





unshare constructor of redex

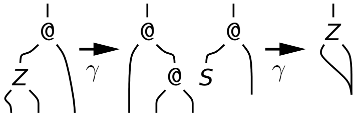
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

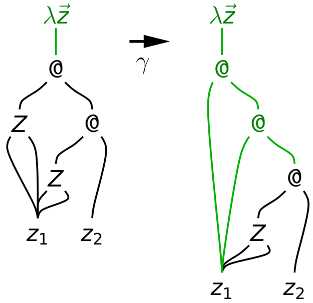
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda\vec{z}.(\lambda z.z_1\,(z_1\,z))\,((\lambda z.z_1\,(z_1\,z))\,z_2)$$
$$\Big\downarrow \beta$$
$$\lambda\vec{z}.z_1\,(z_1\,((\lambda z.z_1\,(z_1\,z))\,z_2))$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda\vec{z}.Z[z_1,z_1]\,(Z[z_1,z_1]\,z_2)$$
$$\Big\downarrow \kappa$$
$$\lambda\vec{z}.z_1\,(z_1\,(Z[z_1,z_1]\,z_2))$$





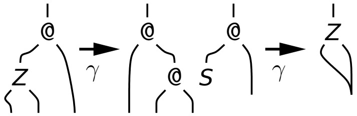
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

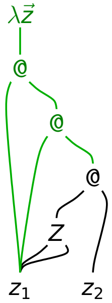
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \rightarrow_\beta M[x{:=}N]$$
$$\lambda z.M \rightarrow_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda\vec{z}.z_1\,(z_1\,((\lambda z.z_1\,(z_1\,z))\,z_2))$$

$$Z[x,y]\,z \rightarrow_\kappa x\,(y\,z)$$
$$S\,z \rightarrow_\kappa Z[z,z]$$
$$Z[x,y] \rightarrow_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \rightarrow_\alpha \lambda z'.S\,z'$$

$$\lambda\vec{z}.z_1\,(z_1\,(Z[z_1,z_1]\,z_2))$$





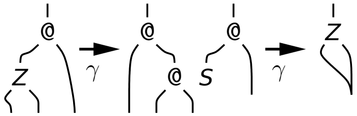
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

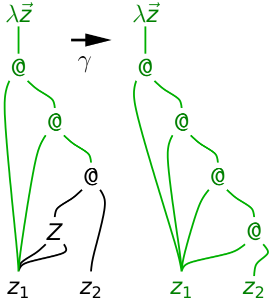
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda\vec{z}.z_1\,(z_1\,((\lambda z.z_1\,(z_1\,z))\,z_2))$$
$$\Big\downarrow \beta$$
$$\lambda\vec{z}.z_1\,(z_1\,(z_1\,(z_1\,z_2)))$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda\vec{z}.z_1\,(z_1\,(Z[z_1,z_1]\,z_2))$$
$$\Big\downarrow \kappa$$
$$\lambda\vec{z}.z_1\,(z_1\,(z_1\,(z_1\,z_2)))$$





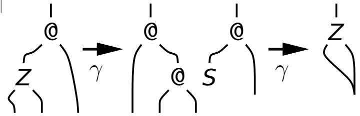
λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

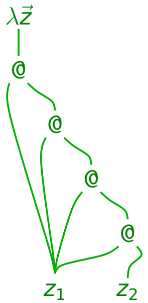
$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

$$\lambda\vec{z}.z_1\,(z_1\,(z_1\,(z_1\,z_2)))$$

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda\vec{z}.z_1\,(z_1\,(z_1\,(z_1\,z_2)))$$





λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

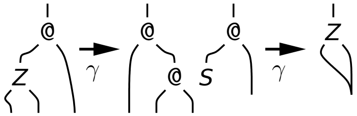
$$\lambda\vec{z}.z_1\,(z_1\,(z_1\,(z_1\,z_2)))$$

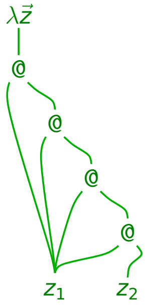
normal form

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$$\lambda\vec{z}.z_1\,(z_1\,(z_1\,(z_1\,z_2)))$$

normal form





normal form

λ-calculus $\iff$ combinator system $\iff$ TGRS

$$\underline{2} := \lambda sz.s\,(s\,z)$$
$$(\lambda x.M)\,N \to_\beta M[x{:=}N]$$
$$\lambda z.M \to_\alpha \lambda z'.(\lambda z.M)\,z'$$

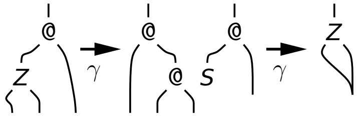
$\underline{4}$

normal form

$$Z[x,y]\,z \to_\kappa x\,(y\,z)$$
$$S\,z \to_\kappa Z[z,z]$$
$$Z[x,y] \to_\alpha \lambda z'.Z[x,y]\,z'$$
$$S \to_\alpha \lambda z'.S\,z'$$

$\underline{4}$

normal form



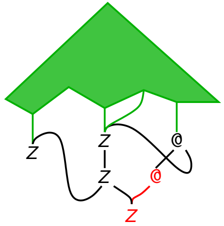
$\dagger_4$

normal form

Spine strategy

Definition (spine prefix)

λ-term-nodes (@,λx,x) of whnf (recursively; in tree; reachable from root)

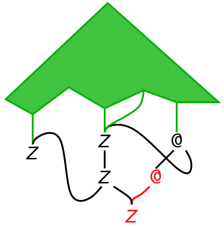


1. leftmost Z is non-green-covered

Spine strategy

Definition (spine prefix)

λ-term-nodes of whnf (recursively; in tree; reachable from root)

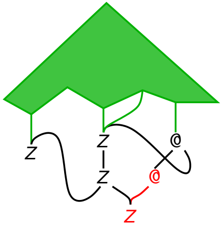


1. leftmost Z is non-green-covered
2. top–middle Z is again non-green-covered

Spine strategy

Definition (spine prefix)

λ-term-nodes of whnf (recursively; in tree; reachable from root)

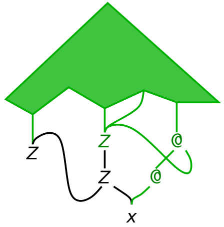


1. leftmost Z is non-green-covered
2. top–middle Z is again non-green-covered
3. top–right @ is green-covered; its spine has Z-redex $\Longrightarrow$ $\rightarrow_{\mathsf{sp}\kappa}$-step

Spine strategy

Definition (spine prefix)

λ-term-nodes of whnf (recursively; in tree; reachable from root)

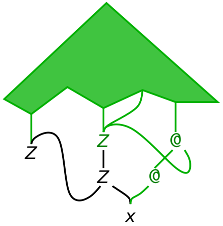


1. leftmost Z is non-green-covered

Spine strategy

Definition (spine prefix)

λ-term-nodes of whnf (recursively; in tree; reachable from root)



1. leftmost Z is non-green-covered
2. top–middle Z is green-covered; unfold $\Longrightarrow$ $\rightarrow_{\alpha}$-step

Spine strategy

Definition (spine prefix)

λ-term-nodes of whnf (recursively; in tree; reachable from root)

Lemma

- *graph G in normal form iff G is spine prefix*

Spine strategy

Definition (spine prefix)

λ-term-nodes of whnf (recursively; in tree; reachable from root)

Lemma

- *graph G in normal form iff G is spine prefix*
- $\rightarrow_{\mathsf{sp}\gamma}$*-step maps back to* $\twoheadrightarrow\!\!\!\!\!\shortmid\!\!\shortmid_{\mathsf{fsp}\beta}$*-step on* λ*-term*
  (parallel β*-step contracting family of* β*-redexes; at least one spine)*

Spine strategy

Definition (spine prefix)

λ-term-nodes of whnf (recursively; in tree; reachable from root)

Lemma

- *graph G in normal form iff G is spine prefix*
- $\rightarrow_{\mathsf{sp}\gamma}$*-step maps back to* $\twoheadrightarrow\!\!\!\!\!\!\!+\,_{\mathsf{fsp}\beta}$*-step on* λ*-term*
- $\rightarrow_{\alpha}$*-step maps back to* $\twoheadrightarrow\!\!\!\!\!\!\!+\,_{\alpha}$*-step on* λ*-term*

Theorem

1. *leftmost–outermost $\to_{\ell o \beta}$ is a spine-strategy ($\to_{\mathrm{sp}\beta}$-strategy) on λ-terms (not other way around)*

Theorem

1. *leftmost–outermost is a spine-strategy on λ-terms*
2. $\rightarrow_{\mathrm{sp}\beta}$ *is random descent (RD) strategy, so* $\twoheadrightarrow\!\!\!\!\!\!\shortmid\!\shortmid_{\mathrm{fsp}\beta}$ *is hyper-normalising*
   (RD: all maximal reductions yield same nf (if any) and of same length)

Theorem

1. *leftmost–outermost is a spine-strategy on λ-terms*
2. $\rightarrow_{\mathsf{sp}\beta}$ *is random descent (RD) strategy, so* $\twoheadrightarrow_{\mathsf{fsp}\beta}$ *is hyper-normalising*
3. $\#\mathsf{sp}\gamma \leq c \cdot \#\ell\mathsf{o}\beta$ *for reduction of M to nf, for constant c depending on M (in turn,* $\#\rightarrow_\alpha$ *bounded via* $\#\mathsf{sp}\gamma$*)*

Theorem

1. *leftmost–outermost is a spine-strategy on λ-terms*
2. $\rightarrow_{\mathsf{sp}\beta}$ *is random descent (RD) strategy, so* $\twoheadrightarrow\!\!\!\!\!\!\shortmid\shortmid\;_{\mathsf{fsp}\beta}$ *is hyper-normalising*
3. $\#\mathsf{sp}\gamma \leq c \cdot \#\ell\mathsf{o}\beta$ *for reduction of M to nf, for constant c depending on M*
4. $\rightarrow_{\mathsf{sp}\gamma}$ *maps to optimal strategy for* $\twoheadrightarrow\!\!\!\!\!\!\shortmid\shortmid\;_{\mathsf{fsp}\kappa}$

Theorem

1. *leftmost–outermost is a spine-strategy on λ-terms*
2. $\rightarrow_{\mathsf{sp}\beta}$ *is random descent (RD) strategy, so* $\twoheadrightarrow\!\!\!\!\!\!+\!\!+_{\mathsf{fsp}\beta}$ *is hyper-normalising*
3. $\#\mathsf{sp}\gamma \leq c \cdot \#\ell\mathsf{o}\beta$ *for reduction of M to nf, for constant c depending on M*
4. $\rightarrow_{\mathsf{sp}\gamma}$ *maps to optimal strategy for* $\twoheadrightarrow\!\!\!\!\!\!+\!\!+_{\mathsf{fsp}\kappa}$

Intermediate conclusions

1. classical term-graph rewrite techniques to implement fspβ; ℓoβ-cost model (natively allows for parallelism; contrast with (Accattoli, Dal Lago))

Theorem

1. *leftmost–outermost is a spine-strategy on λ-terms*
2. $\rightarrow_{\mathsf{sp}\beta}$ *is random descent (RD) strategy, so* $\twoheadrightarrow\!\!\!\!\!\!\shortmid\!\shortmid_{\mathsf{fsp}\beta}$ *is hyper-normalising*
3. $\#\mathsf{sp}\gamma \leq c \cdot \#\ell\mathsf{o}\beta$ *for reduction of M to nf, for constant c depending on M*
4. $\rightarrow_{\mathsf{sp}\gamma}$ *maps to optimal strategy for* $\twoheadrightarrow\!\!\!\!\!\!\shortmid\!\shortmid_{\mathsf{fsp}\kappa}$

Intermediate conclusions

1. classical term-graph rewrite techniques to implement fspβ; ℓoβ-cost model
2. based on weak-β (Balabonski), naïve substitution, explicit α
   (no need for De Bruijn-indices; no need for machines)

Theorem

1. *leftmost–outermost is a spine-strategy on λ-terms*
2. $\rightarrow_{\text{sp}\beta}$ *is random descent (RD) strategy, so* $\twoheadrightarrow\!\!\!\!\!\!\!\;\shortmid\!\shortmid_{\text{fsp}\beta}$ *is hyper-normalising*
3. $\#\text{sp}\gamma \leq c \cdot \#\ell\text{o}\beta$ *for reduction of M to nf, for constant c depending on M*
4. $\rightarrow_{\text{sp}\gamma}$ *maps to optimal strategy for* $\twoheadrightarrow\!\!\!\!\!\!\!\;\shortmid\!\shortmid_{\text{fsp}\kappa}$

Intermediate conclusions

1. classical term-graph rewrite techniques to implement fspβ; ℓoβ-cost model
2. based on weak-β, naïve substitution, explicit α
3. $\rightarrow_{\text{sp}\gamma}$ optimal implementation of combinator system; cbv unproblematic? (since horizontal sharing suffices; cbv for weak values; WiP)

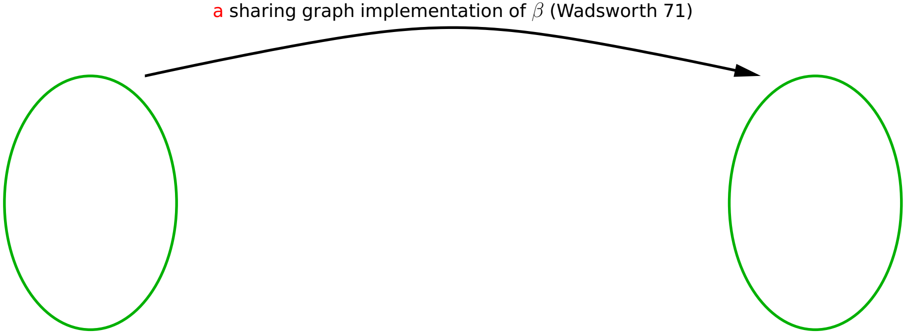
Theorem

1. *leftmost–outermost is a spine-strategy on λ-terms*
2. $\rightarrow_{\mathrm{sp}\beta}$ *is random descent (RD) strategy, so* $\twoheadrightarrow\!\!\!\!\!+_{\mathrm{fsp}\beta}$ *is hyper-normalising*
3. $\#\mathrm{sp}\gamma \leq c \cdot \#\ell\mathrm{o}\beta$ *for reduction of M to nf, for constant c depending on M*
4. $\rightarrow_{\mathrm{sp}\gamma}$ *maps to optimal strategy for* $\twoheadrightarrow\!\!\!\!\!+_{\mathrm{fsp}\kappa}$

Intermediate conclusions

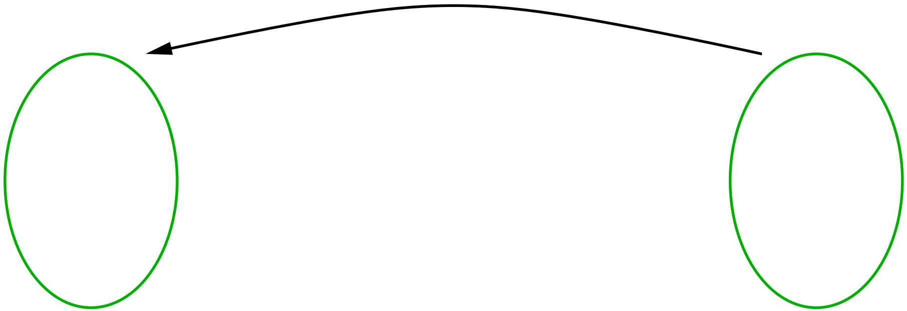
1. classical term-graph rewrite techniques to implement fspβ; ℓoβ-cost model
2. based on weak-β, naïve substitution, explicit α
3. $\rightarrow_{\mathrm{sp}\gamma}$ optimal implementation of combinator system; cbv unproblematic?
4. amortised analysis: discounting α-steps via β-steps initiating them

λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS, weak

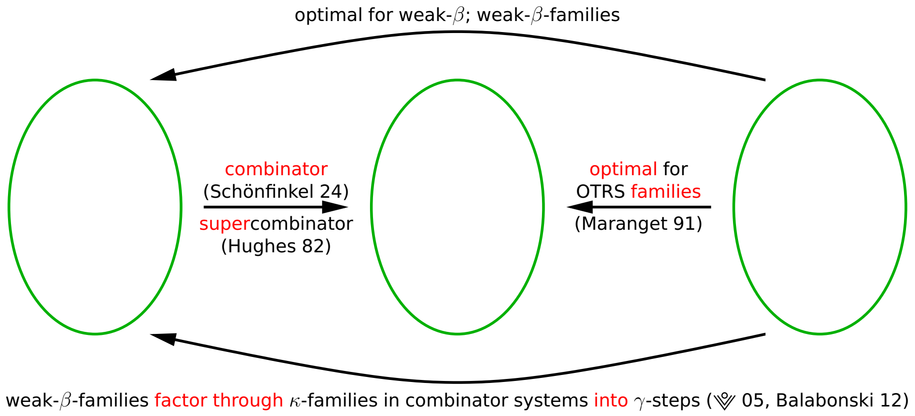


λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS, weak

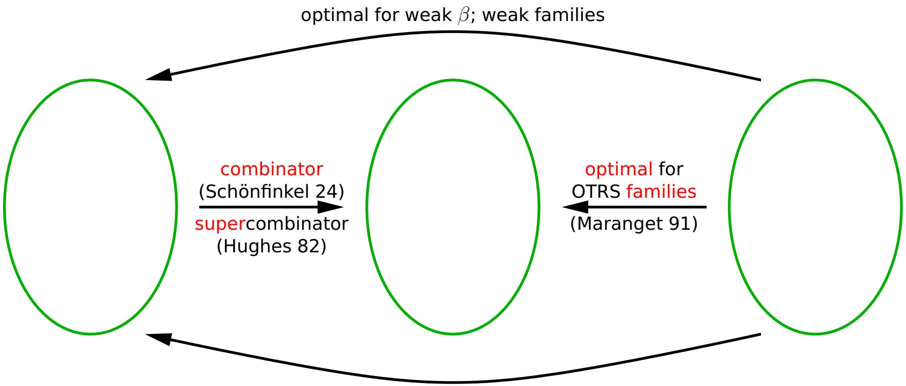
optimal (Blanc, Lévy, Maranget 05) for weak-β (Çağman, Hindley 98); weak-β-families



λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS, weak


optimal for weak-β; weak-β-families
combinator (Schönfinkel 24)
supercombinator (Hughes 82)
optimal for OTRS families (Maranget 91)
weak-β-families factor through κ-families in combinator systems into γ-steps (05, Balabonski 12)

λ-calculus $\Longleftrightarrow$ combinator system $\Longleftrightarrow$ TGRS, spβ



spβ-families factor through spκ-families in combinator systems into spγ-steps, with explicit-α (this talk)

Amortised complexity

Idea

measure complexity by averaging over reductions (Tarjan)
(instead of measuring per step)

Amortised complexity

Idea

measure complexity by averaging over reductions

Example

incrementing a counter in binary 011 $\to_{\text{inc}}$ 111 $\to_{\text{inc}}$ 0001 $\to_{\text{inc}}$ 1001 $\to_{\text{inc}}$ ...
($\to_{\text{inc}}$-steps not unit-time; #bit-flips unbounded)

Amortised complexity

Idea

measure complexity by averaging over reductions

Example

incrementing a counter in binary 011 $\to_{\text{inc}}$ 111 $\to_{\text{inc}}$ 0001 $\to_{\text{inc}}$ 1001 $\to_{\text{inc}} \dots$

Example (inc **as term rewrite system;** $\to_{\text{inc}} := \to_i \cdot \to_b^!$**)**

$$s \to_i i(s) \qquad i(0(x)) \to_b 1(x) \qquad i(1(x)) \to_b 0(i(x)) \qquad i(\bullet) \to_b 1(\bullet)$$

Amortised complexity

Idea

measure complexity by averaging over reductions

Example

incrementing a counter in binary 011 $\rightarrow_{\mathsf{inc}}$ 111 $\rightarrow_{\mathsf{inc}}$ 0001 $\rightarrow_{\mathsf{inc}}$ 1001 $\rightarrow_{\mathsf{inc}} \dots$

Example (inc **as term rewrite system;** $\rightarrow_{\mathsf{inc}} := \rightarrow_i \cdot \rightarrow_b^!$**)**

$$s \rightarrow_i i(s) \qquad i(0(x)) \rightarrow_b 1(x) \qquad i(1(x)) \rightarrow_b 0(i(x)) \qquad i(\bullet) \rightarrow_b 1(\bullet)$$

$0(1(1(\bullet))) \rightarrow_i i(0(1(1(\bullet)))) \rightarrow_b 1(1(1(\bullet))) \rightarrow_i i(1(1(1(\bullet)))) \rightarrow_b 0(i(1(1(\bullet)))) \rightarrow_b$
$0(0(i(1(\bullet)))) \rightarrow_b 0(0(0(i(\bullet)))) \rightarrow_b 0(0(0(1(\bullet)))) \rightarrow_i \dots$

Banker's / accounting method in TRSs

Idea

distinguish between charge $\hat{c}$ and cost c of steps. i-steps add charge to pay for cost of subsequent b-steps; labelled ($\mathbb{N}$) symbols as saving-account for charges

Banker's / accounting method in TRSs

Idea

distinguish between charge $\hat{c}$ and cost c of steps. i-steps add charge to pay for cost of subsequent b-steps; labelled ($\mathbb{N}$) symbols as saving-account for charges

Example

$s \to_{\hat{3},1} i^{\hat{2}}(s) \qquad i^{\hat{2}}(0(x)) \to_{\hat{0},1} 1^{\hat{1}}(x) \qquad i^{\hat{2}}(1^{\hat{1}}(x)) \to_{\hat{0},1} 0(i^{\hat{2}}(x)) \qquad i^{\hat{2}}(\bullet) \to_{\hat{0},1} 1^{\hat{1}}(\bullet)$
(no need to label 0's or •'s)

Banker’s / accounting method in TRSs

Idea

distinguish between charge $\hat{c}$ and cost c of steps. i-steps add charge to pay for cost of subsequent b-steps; labelled ($\mathbb{N}$) symbols as saving-account for charges

Example

$s \to_{\hat{3},1} i^{\hat{2}}(s) \qquad i^{\hat{2}}(0(x)) \to_{\hat{0},1} 1^{\hat{1}}(x) \qquad i^{\hat{2}}(1^{\hat{1}}(x)) \to_{\hat{0},1} 0(i^{\hat{2}}(x)) \qquad i^{\hat{2}}(\bullet) \to_{\hat{0},1} 1^{\hat{1}}(\bullet)$

- $\hat{\imath}$ initially labels (closed): charge i with $\hat{2}$ and 1 with $\hat{1}$; preserved by steps

Banker's / accounting method in TRSs

Idea

distinguish between charge $\hat{c}$ and cost c of steps. i-steps add charge to pay for cost of subsequent b-steps; labelled ($\mathbb{N}$) symbols as saving-account for charges

Example

$s \to_{\hat{3},1} i^{\hat{2}}(s) \qquad i^{\hat{2}}(0(x)) \to_{\hat{0},1} 1^{\hat{1}}(x) \qquad i^{\hat{2}}(1^{\hat{1}}(x)) \to_{\hat{0},1} 0(i^{\hat{2}}(x)) \qquad i^{\hat{2}}(\bullet) \to_{\hat{0},1} 1^{\hat{1}}(\bullet)$

- $\hat{\iota}$ initially labels: charge i with $\hat{2}$ and 1 with $\hat{1}$; preserved by steps
- is a labelling: if $t \twoheadrightarrow s$, then $t^{\hat{\iota}} \twoheadrightarrow s^{\hat{\iota}}$

Banker's / accounting method in TRSs

Idea

distinguish between charge $\hat{c}$ and cost c of steps. i-steps add charge to pay for cost of subsequent b-steps; labelled ($\mathbb{N}$) symbols as saving-account for charges

Example

$s \rightarrow_{\hat{3},1} i^{\hat{2}}(s) \qquad i^{\hat{2}}(0(x)) \rightarrow_{\hat{0},1} 1^{\hat{1}}(x) \qquad i^{\hat{2}}(1^{\hat{1}}(x)) \rightarrow_{\hat{0},1} 0(i^{\hat{2}}(x)) \qquad i^{\hat{2}}(\bullet) \rightarrow_{\hat{0},1} 1^{\hat{1}}(\bullet)$

- $\hat{\imath}$ initially labels: charge i with $\hat{2}$ and 1 with $\hat{1}$; preserved by steps
- is a labelling: if $t \twoheadrightarrow s$, then $t^{\hat{\imath}} \twoheadrightarrow s^{\hat{\imath}}$
  (in general: cost subtracted; charges must remain non-negative, cover costs of steps; $\hat{c} + \sum \ell \geq c + \sum r$ for each (linear) rule $\ell \rightarrow_{\hat{c},c} r$)

Banker's / accounting method in TRSs

Idea

distinguish between charge $\hat{c}$ and cost c of steps. i-steps add charge to pay for cost of subsequent b-steps; labelled ($\mathbb{N}$) symbols as saving-account for charges

Example

$s \to_{\hat{3},1} i^{\hat{2}}(s) \qquad i^{\hat{2}}(0(x)) \to_{\hat{0},1} 1^{\hat{1}}(x) \qquad i^{\hat{2}}(1^{\hat{1}}(x)) \to_{\hat{0},1} 0(i^{\hat{2}}(x)) \qquad i^{\hat{2}}(\bullet) \to_{\hat{0},1} 1^{\hat{1}}(\bullet)$

- $\hat{\imath}$ initially labels: charge i with $\hat{2}$ and 1 with $\hat{1}$; preserved by steps
- is a labelling: if $t \twoheadrightarrow s$, then $t^{\hat{\imath}} \twoheadrightarrow s^{\hat{\imath}}$
- cost of reduction from t bounded by amortized cost, $\leq 3 \cdot \#i + \sum t^{\hat{\imath}}$

Notions from TRS theory for Banker’s account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Notions from TRS theory for Banker’s account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Definition

$\langle M, \bot, +, \leq \rangle$ derivation monoid if

- $\langle M, \bot, + \rangle$ a monoid;

Notions from TRS theory for Banker’s account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Definition

$\langle M, \perp, +, \leq \rangle$ derivation monoid if

- $\langle M, \perp, + \rangle$ a monoid;
- $\leq$ well-founded order with $\perp$ least;

Notions from TRS theory for Banker’s account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Definition

$\langle M, \perp, +, \leq \rangle$ derivation monoid if

- $\langle M, \perp, + \rangle$ a monoid;
- $\leq$ well-founded order with $\perp$ least;
- $+$ is $\leq$-monotonic in both arguments; strictly in 2nd.

Notions from TRS theory for Banker's account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Definition

$\langle M, \perp, +, \leq \rangle$ derivation monoid if

- $\langle M, \perp, + \rangle$ a monoid;
- $\leq$ well-founded order with $\perp$ least;
- $+$ is $\leq$-monotonic in both arguments; strictly in 2nd.

main example: ordinals with zero, addition, less–than–or–equal

Notions from TRS theory for Banker's account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Definition

$\langle M, \perp, +, \leq \rangle$ derivation monoid

- measure on $\rightarrow$ maps steps to $M - \{\perp\}$;

Notions from TRS theory for Banker’s account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Definition

$\langle M, \bot, +, \leq \rangle$ derivation monoid

- measure on $\rightarrow$ maps steps to $M - \{\bot\}$;
- measure of finite reduction is sum ($+$; tail to head) of steps (starting with $\bot$);

Notions from TRS theory for Banker’s account

Idea 1 (Toyama, ⩔ 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Definition

$\langle M, \bot, +, \leq \rangle$ derivation monoid

- measure on $\rightarrow$ maps steps to $M - \{\bot\}$;
- measure of finite reduction is sum of steps;
- measure of infinite reduction is $\top$ (fresh top greater than all $m \in M$);

Notions from TRS theory for Banker's account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Idea 2 (Terese, 03)

define a notion of labelling for abstract and term rewriting:

Notions from TRS theory for Banker’s account

Idea 1 (Toyama, ⩔ 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Idea 2 (Terese, 03)

define a notion of labelling for abstract and term rewriting:

- ARS: initial labelling of objects such that every step lifts uniquely (reductions lifts uniquely)

Notions from TRS theory for Banker’s account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Idea 2 (Terese, 03)

define a notion of labelling for abstract and term rewriting:

- ARS: initial labelling of objects such that every step lifts uniquely
- TRSs: label symbols and rules such that steps lift
  (local update; cf. Lévy, Hyland–Wadsworth etc.)

Notions from TRS theory for Banker’s account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Idea 2 (Terese, 03)

define a notion of labelling for abstract and term rewriting:

- ARS: initial labelling of objects such that every step lifts uniquely
- TRSs: label symbols and rules such that steps lift
- amortised: natural numbers to store charges locally
  (locality of TRS rules accounts for distributed nature of accounts)

Notions from TRS theory for Banker's account

Idea 1 (Toyama, 16, 22)

measure steps; assign appropriate weights in derivation monoid $\langle \mathbb{N}, 0, +, \leq \rangle$

Idea 2 (Terese, 03)

define a notion of labelling for abstract and term rewriting:

- ARS: initial labelling of objects such that every step lifts uniquely
- TRSs: label symbols and rules such that steps lift
- amortised: natural numbers to store charges locally

here: charging β-steps suffices to account for α-steps

Unit-time steps in structured rewrite systems?

Structured rewriting

step $C[\varrho]$ from s to t (three structures) for (closed) rule $\varrho : \ell \to r$ if

$$s \leftrightarrow^*_{\mathcal{SC}} C[\ell] \to_\varrho C[r] \leftrightarrow^*_{\mathcal{SC}} t$$

with s, t unique $\mathcal{SC}$-normal forms of $C[\ell], C[r]$ (94, van Raamsdonk 96)

Unit-time steps in structured rewrite systems?

Structured rewriting

step $C[\varrho]$ from s to t for rule $\varrho : \ell \to r$ if

$$s \; {}_{\mathcal{SC}}\!\twoheadleftarrow C[\ell] \to_{\varrho} C[r] \twoheadrightarrow_{\mathcal{SC}} t$$

$\mathcal{SC}$ substitution calculus; $s \; {}_{\mathcal{SC}}\!\twoheadleftarrow C[\ell]$ matching of ℓ; $C[r] \twoheadrightarrow_{\mathcal{SC}} t$ substitution of r

Unit-time steps in structured rewrite systems?

Structured rewriting

step $C[\varrho]$ from s to t for rule $\varrho : \ell \to r$ if $s \; {}_{\mathcal{SC}}\!\twoheadleftarrow C[\ell] \to_{\varrho} C[r] \twoheadrightarrow_{\mathcal{SC}} t$

(string) rule $\varrho : bc \to e$, step

$$a\varrho d : abcd \to aed$$

Unit-time steps in structured rewrite systems?

Structured rewriting

step $C[\varrho]$ from s to t for rule $\varrho : \ell \to r$ if $s \; {}_{\mathcal{SC}}\!\twoheadleftarrow C[\ell] \to_{\varrho} C[r] \twoheadrightarrow_{\mathcal{SC}} t$

(string) rule $\varrho : bc \to e$, step $a\varrho d : abcd \to aed$
(first-order term) rule $x.\varrho[x] : x.g[x,x] \to x.i$, step

$$f[\varrho[h[a]]] : f[g[h[a],h[a]]] \to f[i]$$

where $\mathcal{SC}$ has rules $(x.x)t \to t$, $(x.y)t \to y$ if $x \neq y$, $(x.f[\vec{s}])t \to f[\overrightarrow{(x.s_i)t}]$

Unit-time steps in structured rewrite systems?

Structured rewriting

step $C[\varrho]$ from s to t for rule $\varrho : \ell \to r$ if $s \;{}_{\mathcal{SC}}\!\twoheadleftarrow C[\ell] \to_\varrho C[r] \twoheadrightarrow_{\mathcal{SC}} t$

(string) rule $\varrho : bc \to e$, step $a\varrho d : abcd \to aed$
(first-order term) rule $x.\varrho[x] : x.g[x,x] \to x.i$, step $f[\varrho[h[a]]] : f[g[h[a],h[a]]] \to f[i]$
(higher-order term) rule $\xi : P,Q.\forall x.P \land (Q\,x) \to P,Q.P \land \forall x.Qx$, step

$$(y=0)\lor(\xi\,(y\le 6)\,(x.y\le x)):(y=0)\lor\forall x.(y\le 6)\land(y\le x) \to (y=0)\lor((y\le 6)\land\forall x.(y\le x))$$

where $\mathcal{SC}$ is $\lambda^{\rightarrow}_{\alpha\beta\overline{\eta}}$ (writing $x.M$ for abstraction)

Unit-time steps in structured rewrite systems?

Structured rewriting

step $C[\varrho]$ from s to t for rule $\varrho : \ell \to r$ if $s \; {}_{\mathcal{SC}}{\twoheadleftarrow} \; C[\ell] \to_{\varrho} C[r] \twoheadrightarrow_{\mathcal{SC}} t$

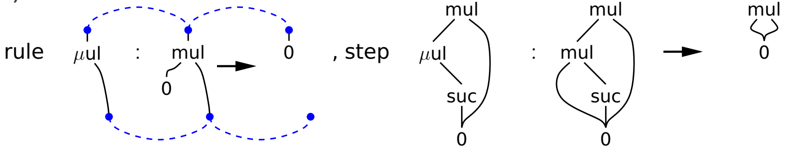
(string) rule $\varrho : bc \to e$, step $a\varrho d : abcd \to aed$

(first-order term) rule $x.\varrho[x] : x.g[x,x] \to x.i$, step $f[\varrho[h[a]]] : f[g[h[a],h[a]]] \to f[i]$

(higher-order term) rule $\xi : P,Q.\forall x.P \wedge (Q\,x) \to P,Q.P \wedge \forall x.Qx$, step

$(y=0)\vee(\xi\,(y\leq 6)\,(x.y\leq x)) : (y=0)\vee\forall x.(y\leq 6)\wedge(y\leq x) \to (y=0)\vee((y\leq 6)\wedge\forall x.(y\leq x))$

(term-graph)



Unit-time steps in structured rewrite systems?

Structured rewriting

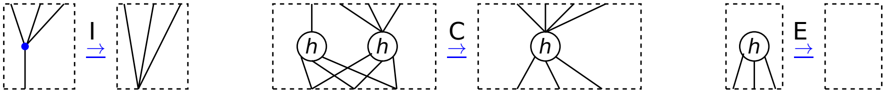
step $C[\varrho]$ from s to t for rule $\varrho : \ell \to r$ if $s \;{}_{\mathcal{SC}}\!\twoheadleftarrow C[\ell] \to_{\varrho} C[r] \twoheadrightarrow_{\mathcal{SC}} t$

(string) rule $\varrho : bc \to e$, step $a\varrho d : abcd \to aed$

(first-order term) rule $x.\varrho[x] : x.g[x,x] \to x.i$, step $f[\varrho[h[a]]] : f[g[h[a],h[a]]] \to f[i]$

(higher-order term) rule $\xi : P,Q.\forall x.P \wedge (Q\,x) \to P,Q.P \wedge \forall x.Qx$, step

$(y = 0) \vee (\xi\,(y \leq 6)\,(x.y \leq x)) : (y = 0) \vee \forall x.(y \leq 6) \wedge (y \leq x) \to (y = 0) \vee ((y \leq 6) \wedge \forall x.(y \leq x))$

(term-graph) 

$\mathcal{SC}$ is ж-calculus for indirection nodes (●) with gc and maximal sharing

Unit-time steps in structured rewrite systems?

Structured rewriting

step $C[\varrho]$ from s to t for rule $\varrho : \ell \to r$ if $s \;{}_{\mathcal{SC}}\!\twoheadleftarrow C[\ell] \to_{\varrho} C[r] \twoheadrightarrow_{\mathcal{SC}} t$

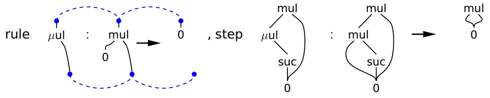
(string) rule $\varrho : bc \to e$, step $a\varrho d : abcd \to aed$

(first-order term) rule $x.\varrho[x] : x.g[x,x] \to x.i$, step $f[\varrho[h[a]]] : f[g[h[a],h[a]]] \to f[i]$

(higher-order term) rule $\xi : P,Q.\forall x.P \wedge (Q\,x) \to P,Q.P \wedge \forall x.Qx$, step

$(y=0)\vee(\xi\,(y\le 6)\,(x.y\le x)) : (y=0)\vee\forall x.(y\le 6)\wedge(y\le x) \to (y=0)\vee((y\le 6)\wedge\forall x.(y\le x))$

(term-graph)



Observation

$\mathcal{SC}$ complex; unit-time steps *a priori* unreasonable for structured rewriting

Conclusions

- rewriting useful both for simple description and efficient implementation (no intermediate abstract machines (Krivine))

Conclusions

- rewriting useful both for simple description and efficient implementation
- higher-order rewriting useful to bridge λ-calculus $\iff$ supercombinators (rid of binders, no intermediate let-calculus; combinator system novel?)

Conclusions

- rewriting useful both for simple description and efficient implementation
- higher-order rewriting useful to bridge λ-calculus $\iff$ supercombinators
- substitution calculus (⩔, van Raamsdonk) useful to modularise TGR
  (ж makes matching and substitution explicit; see paper)

Conclusions

- rewriting useful both for simple description and efficient implementation
- higher-order rewriting useful to bridge λ-calculus $\iff$ supercombinators
- substitution calculus (, van Raamsdonk) useful to modularise TGR
- classical techniques (Schönfinkel, Wadsworth, Hughes,...) for complexity

Conclusions

- rewriting useful both for simple description and efficient implementation
- higher-order rewriting useful to bridge λ-calculus $\iff$ supercombinators
- substitution calculus (, van Raamsdonk) useful to modularise TGR
- classical techniques (Schönfinkel, Wadsworth, Hughes,...) for complexity
- labelling (Terese) and derivation monoids, random descent (Toyama,) techniques give smooth theoretical basis for complexity analysis

Conclusions

- rewriting useful both for simple description and efficient implementation
- higher-order rewriting useful to bridge λ-calculus $\iff$ supercombinators
- substitution calculus (, van Raamsdonk) useful to modularise TGR
- classical techniques (Schönfinkel, Wadsworth, Hughes,…) for complexity
- labelling (Terese) and derivation monoids, random descent (Toyama,) techniques give smooth theoretical basis for complexity analysis
- Gödel not convinced by $\lambda\beta$ / TRS; me neither because no unit-time steps (abstract from replication; cf. Java abstracting from garbage collection)

Conclusions

- rewriting useful both for simple description and efficient implementation
- higher-order rewriting useful to bridge λ-calculus $\iff$ supercombinators
- substitution calculus (, van Raamsdonk) useful to modularise TGR
- classical techniques (Schönfinkel, Wadsworth, Hughes, . . .) for complexity
- labelling (Terese) and derivation monoids, random descent (Toyama,) techniques give smooth theoretical basis for complexity analysis
- Gödel not convinced by $\lambda\beta$ / TRS; me neither because no unit-time steps

full paper in preparation (with Clemens Grabmayer)

Conclusions

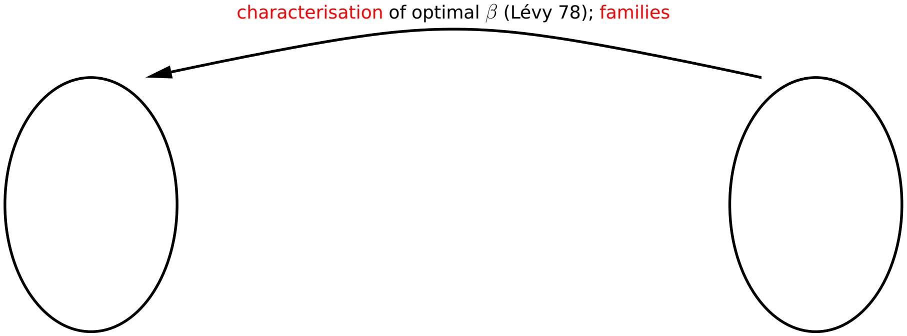
- rewriting useful both for simple description and efficient implementation
- higher-order rewriting useful to bridge λ-calculus $\iff$ supercombinators
- substitution calculus (, van Raamsdonk) useful to modularise TGR
- classical techniques (Schönfinkel, Wadsworth, Hughes,. . .) for complexity
- labelling (Terese) and derivation monoids, random descent (Toyama,) techniques give smooth theoretical basis for complexity analysis
- Gödel not convinced by $\lambda\beta$ / TRS; me neither because no unit-time steps

full paper in preparation (with Clemens Grabmayer) thanks: students, co-workers (Zwitserlood, Hendriks, Heijltjes,. . .)

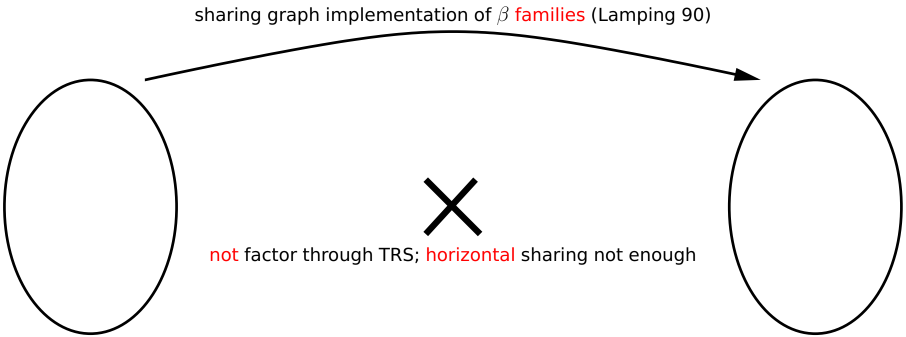
Reduction to (wh)nf in $\lambda\beta$, naïvely, in Haskell

```
data Lam = Lam Head [Lam] deriving (Show)
data Head = Var String | Abs String Lam deriving (Show)
subst x s (Lam h l) = let
  (Lam h' l') = case h of
     (Var y)   | x == y -> s
     (Abs y u) | x /= y -> Lam (Abs y (subst x s u)) []
     _                  -> Lam h [] in (Lam h' (l'++(map (subst x s) l)))
whnf (Lam (Abs x t) (u:l)) = let Lam h s = subst x u t in whnf (Lam h (s++l))
whnf t = t
nf = rnf (\x -> 1)
rnf f t = let
  (Lam h l) = whnf t
  f' x      = \y -> f y + (if (x==y) then 1 else 0)
  v x       = x++"_"++show (f x) in case h of
    (Abs x _) -> Lam (Abs (v x) (rnf (f' x) (Lam h [Lam (Var (v x)) []]))) []
    _         -> Lam h (map (rnf f) l)
```

λ-calculus $\Longleftrightarrow$ interaction nets (Lafont 90), strong



λ-calculus $\Longleftrightarrow$ interaction nets (Lafont 90), strong


sharing graph implementation of β families (Lamping 90)
not factor through TRS; horizontal sharing not enough

A puzzle to ponder on α-conversion

- give an upperbound on the #α-renamings needed to β-reduce $((\underline{2}\,\underline{8})\,(\underline{4}\,\underline{9}))\,(\underline{5}\,\underline{7})\,(\underline{4}\,\underline{2})$ to normal form?
- note 1: application of Church-numerals is exponentiation; $\underline{k}\,\underline{n} \twoheadrightarrow_\beta \underline{n^k}$
- note 2: whether α-conversion is needed in a β-reduction is undecidable