Alpha-Renaming of Higher-Order Meta-Expressions

David Sabel[†]

Goethe-University Frankfurt am Main, Germany

PPDP 2017, Namur, Belgium

[†] Research supported by the Deutsche Forschungsgemeinschaft (DFG) under grant SA 2908/3-1.

Motivation

- **automated reasoning on programs and program transformations** w.r.t. operational semantics
- for program calculi with higher-order constructs and **recursive bindings**, e.g.

$$\texttt{letrec}\ x_1 = s_1; \ldots; x_n = s_n\ \texttt{in}\ t$$

- extended call-by-need lambda calculi with letrec that model core languages of **lazy functional programming languages** like Haskell

Requirements on the Meta-Language

Call-by-need standard reductions & program transformations (excerpt)

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \to \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ Env\ \texttt{in}\ e] \ \to\ T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

Meta-syntax must be capable to represent:

- contexts of different classes
- environments Env_i,

Syntax of the Meta-Language LRSX

Variables

$$\begin{array}{rcll} x \in \textbf{Var} & ::= & X & \text{(variable meta-variable)} \\ & \mid & \mathsf{x} & \text{(concrete variable)} \end{array}$$

Expressions

$$\begin{array}{rcll} s \in \textbf{Expr} & ::= & S & \text{(expression meta-variable)} \\ & \mid & \mathtt{var}\ x & \text{(variable)} \\ & \mid & (f\ r_1 \ldots r_{ar(f)}) & \text{(function application)} \\ & & \quad \text{where } r_i \text{ is } o_i, s_i, \text{ or } x_i \text{ specified by } f & \\ & \mid & D[s] & \text{(context meta-variable of class } cl(D)\text{)} \\ & \mid & \mathtt{letrec}\ env\ \mathtt{in}\ s & \text{(letrec-expression)} \\ o \in \textbf{HExpr}^n & ::= & x_1.\ldots.x_n.s & \text{(higher-order expression)} \end{array}$$

Environments:

$$\begin{array}{rcll} env \in \textbf{Env} & ::= & \emptyset & \text{(empty environment)} \\ & \mid & E; env & \text{(environment meta-variable)} \\ & \mid & x{=}s; env & \text{(binding)} \end{array}$$

Binding and Scoping Constraints

Call-by-need standard reductions & program transformations (excerpt)

(SR,lbeta) $R[(\lambda x.e_1)\ e_2] \to R[\texttt{letrec}\ x = e_2\ \texttt{in}\ e_1]$

(SR,llet) $\texttt{letrec}\ Env_1\ \texttt{in}\ \texttt{letrec}\ Env_2\ \texttt{in}\ e \to \texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ e$

(T,cpx) $T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]] \to T[\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y]]$

(T,gc) $T[\texttt{letrec}\ Env\ \texttt{in}\ e]\ \to\ T[e]$ if $LetVars(Env) \cap FV(e) = \emptyset$

restrictions on scoping and emptiness have to be respected, e.g.:

- (gc): Env must not be empty; side condition on variables
- (llet): $FV(Env_1) \cap LetVars(Env_2) = \emptyset$
- (cpx): x, y are not captured by C in $C[x]$

Constrained Expressions

- A **constraint tuple** $\Delta = (\Delta_1, \Delta_2, \Delta_3)$ consists of

  Δ_1: set of context variables (non-empty context constraint)

  Δ_2: set of environment variables (non-empty environment constraint)

  Δ_3: set of pairs (s, d) (non-capture constraint, NCC)
  (s an expression, d a context)

- Ground substitution ρ **satisfies** $(\Delta_1, \Delta_2, \Delta_3)$ iff
  - $\rho(D) \neq [\cdot]$ for all $D \in \Delta_1$
  - $\rho(E) \neq \emptyset$ for all $E \in \Delta_2$
  - hole of $\rho(d)$ does not capture variables of $\rho(s)$, for all $(s, d) \in \Delta_3$

Constrained Expressions

- A **constraint tuple** $\Delta = (\Delta_1, \Delta_2, \Delta_3)$ consists of
  - Δ_1: set of context variables (non-empty context constraint)
  - Δ_2: set of environment variables (non-empty environment constraint)
  - Δ_3: set of pairs (s, d) (non-capture constraint, NCC)
    (s an expression, d a context)

- Ground substitution ρ **satisfies** $(\Delta_1, \Delta_2, \Delta_3)$ iff
  - $\rho(D) \neq [\cdot]$ for all $D \in \Delta_1$
  - $\rho(E) \neq \emptyset$ for all $E \in \Delta_2$
  - hole of $\rho(d)$ does not capture variables of $\rho(s)$, for all $(s, d) \in \Delta_3$

- A pair (s, Δ) is called a **constrained expression**

$$sem(s, \Delta) = \{\rho(s) \mid \rho(s) \text{ fulfills LVC and } \rho \text{ satisfies } \Delta\}$$

(LVC = let variable convention, binders of the same environment are different)

Application: Correctness of Program Transformations

Program transformation T is **correct** iff

$$\forall \ell \to r \in T\text{: } \forall \text{ contexts } C\text{: } C[\ell]{\downarrow} \iff C[r]{\downarrow}$$

where ${\downarrow} =$ successful evaluation w.r.t. standard reduction

Application: Correctness of Program Transformations

Program transformation T is **correct** iff

$$\forall \ell \to r \in T\text{: } \forall \text{ contexts } C\text{: } C[\ell]{\downarrow} \iff C[r]{\downarrow}$$

where $\downarrow =$ successful evaluation w.r.t. standard reduction

Diagram method to show correctness of transformations:

- Compute overlaps between standard reductions and program transformations (unification, see [SSS16, PPDP])
- Join the overlaps ⇒ forking and commuting diagrams (Meta-rewriting and matching [Sab17, UNIF])
- Induction using the diagrams (automatable, see [RSSS12, IJCAR])

$$\begin{tikzcd}
\sigma(\ell') = \sigma(\ell) \arrow[r, "\text{program transformation}"] \arrow[d, "\text{standard reduction}"'] & \sigma(r) \arrow[d, dashed, "*"] \\
\sigma(r') \arrow[r, dashed, "*"'] & \cdot
\end{tikzcd}
\qquad
\begin{tikzcd}
\sigma(\ell') = \sigma(r) \arrow[d, "\text{standard reduction}"'] & \sigma(\ell) \arrow[l, "\text{program transformation}"'] \arrow[d, dashed, "*"] \\
\sigma(r') & \cdot \arrow[l, dashed, "*"]
\end{tikzcd}$$

Computing Diagrams

$$\begin{tikzcd}
\cdot \arrow[r, "\text{program transformation}"] \arrow[d, "\text{standard reduction}"'] & t_1 \arrow[d, dashed, "*"] \\
t_2 \arrow[r, dashed, "*"'] & \cdot
\end{tikzcd}$$

- t_1, t_2 are **meta-expressions** restricted by **constraints** ∇
- computing joins $\xrightarrow{*}$ requires **abstract rewriting** by rewrite rules $\ell \to_\Delta r$ with Δ restricting ℓ and r
- applying rewrite rules: match the rule and show that the given constraints imply the needed constraints

$$(t, \nabla) \to (\sigma(r), \nabla \cup \sigma(\Delta)) \qquad \text{if } \ell \to_\Delta r,\ t = \sigma(l), \text{ and } \nabla \implies \sigma(\Delta)$$

Example: Overlap (SR,llet) with (T,llet)

(T,llet) $T[\texttt{letrec}\ E\ \texttt{in}\ \texttt{letrec}\ E'\ \texttt{in}\ S] \to T[\texttt{letrec}\ E; E'\ \texttt{in}\ S]$
where an NCC must hold s.t. $LetVars(E') \cap Vars(E) = \emptyset$

$\texttt{letrec}\ E_1\ \texttt{in}$
$\quad\texttt{letrec}\ E_2\ \texttt{in}$
$\quad\quad\texttt{letrec}\ E_3\ \texttt{in}\ S$

$$\text{SR,llet}\ \Big\downarrow$$

$\texttt{letrec}\ E_1; E_2\ \texttt{in}$
$\quad\texttt{letrec}\ E_3\ \texttt{in}\ S$

Given constraints:
- $LetVars(E_2) \cap Vars(E_1) = \emptyset$

Example: Overlap (SR,llet) with (T,llet)

(T,llet) $T[\texttt{letrec}\ E\ \texttt{in}\ \texttt{letrec}\ E'\ \texttt{in}\ S] \to T[\texttt{letrec}\ E; E'\ \texttt{in}\ S]$
where an NCC must hold s.t. $LetVars(E') \cap Vars(E) = \emptyset$

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec}\ E_1\ \texttt{in}\\ \quad\texttt{letrec}\ E_2\ \texttt{in}\\ \qquad\texttt{letrec}\ E_3\ \texttt{in}\ S\end{array} \arrow[r, "\text{T,llet}"] \arrow[d, "\text{SR,llet}"'] & \begin{array}{l}\texttt{letrec}\ E_1\ \texttt{in}\\ \quad\texttt{letrec}\ E_2; E_3\ \texttt{in}\ S\end{array} \\
\begin{array}{l}\texttt{letrec}\ E_1; E_2\ \texttt{in}\\ \quad\texttt{letrec}\ E_3\ \texttt{in}\ S\end{array} &
\end{tikzcd}$$

Given constraints:

- $LetVars(E_2) \cap Vars(E_1) = \emptyset$
- $LetVars(E_3) \cap Vars(E_2) = \emptyset$

Example: Overlap (SR,llet) with (T,llet)

(T,llet) $T[\texttt{letrec}\ E\ \texttt{in}\ \texttt{letrec}\ E'\ \texttt{in}\ S] \to T[\texttt{letrec}\ E; E'\ \texttt{in}\ S]$
where an NCC must hold s.t. $LetVars(E') \cap Vars(E) = \emptyset$

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec}\ E_1\ \texttt{in}\\ \quad\texttt{letrec}\ E_2\ \texttt{in}\\ \quad\quad\texttt{letrec}\ E_3\ \texttt{in}\ S\end{array} \arrow[r, "\text{T,llet}"] \arrow[d, "\text{SR,llet}"'] & \begin{array}{l}\texttt{letrec}\ E_1\ \texttt{in}\\ \quad\texttt{letrec}\ E_2; E_3\ \texttt{in}\ S\end{array} \arrow[d, dashed, "\text{SR,llet}"] \\
\begin{array}{l}\texttt{letrec}\ E_1; E_2\ \texttt{in}\\ \quad\texttt{letrec}\ E_3\ \texttt{in}\ S\end{array} \arrow[r, dashed, "\text{T,llet}"'] & \texttt{letrec}\ E_1; E_2; E_3\ \texttt{in}\ S
\end{tikzcd}$$

Given constraints:

- $LetVars(E_2) \cap Vars(E_1) = \emptyset$
- $LetVars(E_3) \cap Vars(E_2) = \emptyset$

Example: Overlap (SR,llet) with (T,llet)

(T,llet) $T[\texttt{letrec}\ E\ \texttt{in}\ \texttt{letrec}\ E'\ \texttt{in}\ S] \rightarrow T[\texttt{letrec}\ E;E'\ \texttt{in}\ S]$
where an NCC must hold s.t. $LetVars(E') \cap Vars(E) = \emptyset$

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec}\ E_1\ \texttt{in}\\ \quad\texttt{letrec}\ E_2\ \texttt{in}\\ \quad\quad\texttt{letrec}\ E_3\ \texttt{in}\ S\end{array} \arrow[r, "\text{T,llet}"] \arrow[d, "\text{SR,llet}"'] & \begin{array}{l}\texttt{letrec}\ E_1\ \texttt{in}\\ \quad\texttt{letrec}\ E_2;E_3\ \texttt{in}\ S\end{array} \arrow[d, dashed, "\text{SR,llet}", "\times"'] \\
\begin{array}{l}\texttt{letrec}\ E_1;E_2\ \texttt{in}\\ \quad\texttt{letrec}\ E_3\ \texttt{in}\ S\end{array} \arrow[r, dashed, "\text{T,llet}"'] & \texttt{letrec}\ E_1;E_2;E_3\ \texttt{in}\ S
\end{tikzcd}$$

Given constraints:

- $LetVars(E_2) \cap Vars(E_1) = \emptyset$
- $LetVars(E_3) \cap Vars(E_2) = \emptyset$

Needed constraints:

- $LetVars(E_2;E_3) \cap Vars(E_1) = \emptyset$

Example: Overlap (SR,llet) with (T,llet)

(T,llet) $T[\texttt{letrec}\ E\ \texttt{in letrec}\ E'\ \texttt{in}\ S] \to T[\texttt{letrec}\ E; E'\ \texttt{in}\ S]$

where an NCC must hold s.t. $LetVars(E') \cap Vars(E) = \emptyset$

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \quad\texttt{letrec } E_2 \texttt{ in}\\ \qquad\texttt{letrec } E_3 \texttt{ in } S\end{array} \arrow[r, "\text{T,llet}"] \arrow[d, "\text{SR,llet}"'] & \begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \quad\texttt{letrec } E_2; E_3 \texttt{ in } S\end{array} \arrow[d, dashed, "\cancel{\text{SR,llet}}"] \\
\begin{array}{l}\texttt{letrec } E_1; E_2 \texttt{ in}\\ \quad\texttt{letrec } E_3 \texttt{ in } S\end{array} \arrow[r, dashed, "\cancel{\text{T,llet}}"'] & \texttt{letrec } E_1; E_2; E_3 \texttt{ in } S
\end{tikzcd}$$

Given constraints:

- $LetVars(E_2) \cap Vars(E_1) = \emptyset$
- $LetVars(E_3) \cap Vars(E_2) = \emptyset$

Needed constraints:

- $LetVars(E_2; E_3) \cap Vars(E_1) = \emptyset$
- $LetVars(E_3) \cap Vars(E_1; E_2) = \emptyset$

$$\not\Longrightarrow$$

Considering an Instance

Instance: $E_1 \mapsto x{=}z, \quad E_2 \mapsto y{=}1, \quad E_3 \mapsto z{=}2, \quad S \mapsto 3$

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec}\ x{=}z\ \texttt{in}\\ \quad\texttt{letrec}\ y{=}1\ \texttt{in}\\ \quad\quad\texttt{letrec}\ z{=}2\ \texttt{in}\ 3\end{array} \arrow[r, "\text{T,llet}"] \arrow[d, "\text{SR,llet}"'] & \begin{array}{l}\texttt{letrec}\ x{=}z\ \texttt{in}\\ \quad\texttt{letrec}\ y{=}1; z{=}2\ \texttt{in}\ 3\end{array} \arrow[d, dashed, "\cancel{\text{SR,llet}}"] \\
\begin{array}{l}\texttt{letrec}\ x{=}z; y{=}1\ \texttt{in}\\ \quad\texttt{letrec}\ z{=}2\ \texttt{in}\ 3\end{array} \arrow[r, dashed, "\cancel{\text{T,llet}}"'] & \texttt{letrec}\ \boxed{x{=}z; y{=}1; z{=}2}\ \texttt{in}\ 3
\end{tikzcd}$$

illegal capture of z

Given constraints:

- $\mathit{LetVars}(y{=}1) \cap \mathit{Vars}(x{=}z) = \emptyset$
- $\mathit{LetVars}(z{=}2) \cap \mathit{Vars}(y{=}1) = \emptyset$

Needed constraints:

- $\mathit{LetVars}(y{=}1; z{=}2) \cap Vars(x{=}z) = \emptyset$
- $\mathit{LetVars}(z{=}2) \cap \mathit{Vars}(x{=}z; y{=}1) = \emptyset$

Considering an Instance

Instance: $E_1 \mapsto x{=}z,\quad E_2 \mapsto y{=}1,\quad E_3 \mapsto z{=}2,\quad S \mapsto 3$

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec } x{=}z \texttt{ in}\\ \quad\texttt{letrec } y{=}1 \texttt{ in}\\ \qquad\texttt{letrec } z{=}2 \texttt{ in } 3\end{array} \arrow[r, "\text{T,llet}"] \arrow[d, "\text{SR,llet}"'] & \begin{array}{l}\texttt{letrec } x{=}z \texttt{ in}\\ \quad\texttt{letrec } y{=}1; z{=}2 \texttt{ in } 3\end{array} \\
\begin{array}{l}\texttt{letrec } x{=}z; y{=}1 \texttt{ in}\\ \quad\texttt{letrec } z{=}2 \texttt{ in } 3\end{array} \arrow[r, dashed, "\alpha"'] & \begin{array}{l}\texttt{letrec } x_1{=}z; y_1{=}1 \texttt{ in}\\ \quad\texttt{letrec } z_1{=}2 \texttt{ in } 3\end{array}
\end{tikzcd}$$

solution: use fresh α-renamings

Given constraints:

- $\mathit{LetVars}(y{=}1) \cap \mathit{Vars}(x{=}z) = \emptyset$
- $\mathit{LetVars}(z{=}2) \cap \mathit{Vars}(y{=}1) = \emptyset$

Needed constraints:

Considering an Instance

Instance: $E_1 \mapsto x{=}z, \quad E_2 \mapsto y{=}1, \quad E_3 \mapsto z{=}2, \quad S \mapsto 3$

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec}\ x{=}z\ \texttt{in}\\ \quad\texttt{letrec}\ y{=}1\ \texttt{in}\\ \qquad\texttt{letrec}\ z{=}2\ \texttt{in}\ 3\end{array} \arrow[rr, "\text{T,llet}"] \arrow[d, "\text{SR,llet}"'] & & \begin{array}{l}\texttt{letrec}\ x{=}z\ \texttt{in}\\ \quad\texttt{letrec}\ y{=}1; z{=}2\ \texttt{in}\ 3\end{array} \\
\begin{array}{l}\texttt{letrec}\ x{=}z; y{=}1\ \texttt{in}\\ \quad\texttt{letrec}\ z{=}2\ \texttt{in}\ 3\end{array} \arrow[r, dashed, "\alpha"'] & \begin{array}{l}\texttt{letrec}\ x_1{=}z; y_1{=}1\ \texttt{in}\\ \quad\texttt{letrec}\ z_1{=}2\ \texttt{in}\ 3\end{array} \arrow[r, dashed, "\text{T,llet}"'] & \texttt{letrec}\ x_1{=}z; y_1{=}1; z_1{=}2\ \texttt{in}\ 3
\end{tikzcd}$$

solution: use fresh α-renamings

Given constraints:

- $\mathit{LetVars}(y{=}1) \cap \mathit{Vars}(x{=}z) = \emptyset$
- $\mathit{LetVars}(z{=}2) \cap \mathit{Vars}(y{=}1) = \emptyset$

Needed constraints:

- $\mathit{LetVars}(y_1{=}1; z_1{=}2) \cap Vars(x_1{=}z) = \emptyset$

Considering an Instance

Instance: $E_1 \mapsto x{=}z, \quad E_2 \mapsto y{=}1, \quad E_3 \mapsto z{=}2, \quad S \mapsto 3$

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec } x{=}z \texttt{ in}\\ \quad\texttt{letrec } y{=}1 \texttt{ in}\\ \qquad\texttt{letrec } z{=}2 \texttt{ in } 3\end{array} \arrow[rr, "\text{T,llet}"] \arrow[ddd, "\text{SR,llet}"'] & & \begin{array}{l}\texttt{letrec } x{=}z \texttt{ in}\\ \quad\texttt{letrec } y{=}1; z{=}2 \texttt{ in } 3\end{array} \arrow[d, dashed, "\alpha"] \\
 & & \begin{array}{l}\texttt{letrec } x_2{=}z \texttt{ in}\\ \quad\texttt{letrec } y_2{=}1; z_2{=}2 \texttt{ in } 3\end{array} \arrow[d, dashed, "\text{SR,llet}"] \\
 & & \texttt{letrec } x_2{=}z; y_2{=}1; z_2{=}2 \texttt{ in } 3 \arrow[d, phantom, "\sim_\alpha"] \\
\begin{array}{l}\texttt{letrec } x{=}z; y{=}1 \texttt{ in}\\ \quad\texttt{letrec } z{=}2 \texttt{ in } 3\end{array} \arrow[r, dashed, "\alpha"'] & \begin{array}{l}\texttt{letrec } x_1{=}z; y_1{=}1 \texttt{ in}\\ \quad\texttt{letrec } z_1{=}2 \texttt{ in } 3\end{array} \arrow[r, dashed, "\text{T,llet}"'] & \texttt{letrec } x_1{=}z; y_1{=}1; z_1{=}2 \texttt{ in } 3
\end{tikzcd}$$

solution: use fresh α-renamings

Given constraints:

- $\mathit{LetVars}(y{=}1) \cap \mathit{Vars}(x{=}z) = \emptyset$
- $\mathit{LetVars}(z{=}2) \cap \mathit{Vars}(y{=}1) = \emptyset$

Needed constraints:

- $\mathit{LetVars}(y_1{=}1; z_1{=}2) \cap Vars(x_1{=}z) = \emptyset$
- $\mathit{LetVars}(z_2{=}2) \cap \mathit{Vars}(x_2{=}z; y_2{=}1) = \emptyset$

Our Approach

- α-renaming **on the meta-level**
- Instances must fulfill the **distinct variable convention (DVC)**:

Distinct variable convention DVC

A ground `LRSX`-expression fulfills the DVC iff

- the bound variables are disjoint from the free variables
- variables on binders are pairwise disjoint

- How to rename meta-variables X, S, E, D?

$\Rightarrow$ Requires meta-notations for symbolic α-renamings

Syntax of the Extended Meta-Language $\mathtt{LRSX}\alpha$

Variables

$x \in \textbf{Var} ::= \langle rc_1, \dots, rc_n \rangle \cdot X$ (variable meta-variable)

$\quad\mid\ \langle rc_1, \dots, rc_n \rangle \cdot \mathsf{x}$ (concrete variable)

Expressions

$s \in \textbf{Expr} ::= \langle \alpha_{S,i}, rc_1, \dots, rc_n \rangle \cdot S$ (expression meta-variable)

$\quad\mid\ \langle \alpha_{D,i}, rc_1, \dots, rc_n \rangle \cdot D[s]$ (context meta-variable)

$\quad\mid\ \dots$

Environments

$env \in \textbf{Env} ::= \langle \alpha_{E,i}, rc_1, \dots, rc_n \rangle \cdot E; env$ (environment meta-variable)

$\quad\mid\ \dots$

a component $\alpha_{U,i}$ α-renames instances of U

Atomic renaming components

$rc \in \textbf{ARC} ::= \alpha_{x,i}$ (fresh renaming of variable x)

$\quad\mid\ LV(\alpha_{E,i})$ (restriction of $\alpha_{E,i}$ on $\mathit{LetVars}(E)$)

$\quad\mid\ CV(\alpha_{D,i})$ (restriction of $\alpha_{D,i}$ on $\mathit{CapVars}(D)$)

Semantics of $\mathtt{LRSX}\alpha$-Expressions

- For expression s, an interpretation is given by $(\tau \circ \rho)(s)$ where
  - **ground instantiation** ρ instantiates the X-, S-, E-, D-meta variables
  - τ instantiates the renamings $\alpha_{U,i}$ as fresh α-renamings of $\rho(U)$ and:
    - $\tau(\alpha_{x,i})$ is the substitution $\{x \mapsto y_i\}$ where y_i is fresh
    - $\tau(LV(\alpha_{E,i}))$ is the restriction of $\alpha_{E,i}$ to the let-variables of $\rho(E)$
    - $\tau(CV(\alpha_{D,i}))$ is the restriction of $\alpha_{D,i}$ to the capture variables of $\rho(D)$
    - sequences are interpreted as compositions
- $sem(s, \Delta) = \{(\tau \circ \rho)(s) \mid s$ fulfills the LVC and $\tau \circ \rho$ satisfies $\Delta\}$

Example: $s = \mathtt{letrec}\ \langle\alpha_{E,1}\rangle \cdot E\ \mathtt{in}\ \langle\alpha_S, LV(\alpha_{E,1})\rangle \cdot S$

- Let $\rho = \{E \mapsto \mathsf{x} = f\ (\mathtt{var}\ \mathsf{x})\ \lambda\mathsf{x}.\mathtt{var}\ \mathsf{x}, S \mapsto \lambda\mathsf{z}.\mathtt{var}\ \mathsf{x}\}$
- $(\tau \circ \rho)(s)$
  $= \mathtt{letrec}\ \tau(\alpha_{E,1})\rho(E)\ \mathtt{in}\ \tau(LV(\alpha_{E,1}))(\tau(\alpha_S)(\rho(S)))$

  $= \mathtt{letrec}\ \tau(\alpha_{E,1})(\mathsf{x} = f\ (\mathtt{var}\ \mathsf{x})\ \lambda\mathsf{x}.\mathtt{var}\ \mathsf{x})\ \mathtt{in}\ \tau(LV(\alpha_{E,1}))(\tau(\alpha_S)(\lambda\mathsf{z}.\mathtt{var}\ \mathsf{x})))$

  $= \mathtt{letrec}\ \mathsf{x}_1 = f\ (\mathtt{var}\ \mathsf{x}_1)\ \lambda\mathsf{x}_2.\mathtt{var}\ \mathsf{x}_2\ \mathtt{in}\ \lambda\mathsf{z}_1.\mathtt{var}\ \mathsf{x}_1$

Diagram-Computation with Symbolic α-Renaming

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec}\ E_1\ \texttt{in}\\ \ \texttt{letrec}\ E_2\ \texttt{in}\\ \ \ \texttt{letrec}\ E_3\ \texttt{in}\ S\end{array} \arrow[rr, "\mathsf{T,llet}"] \arrow[ddd, "\mathsf{SR,llet}"'] & & \begin{array}{l}\texttt{letrec}\ E_1\ \texttt{in}\\ \ \ \texttt{letrec}\ E_2, E_3\ \texttt{in}\ S\end{array} \arrow[d, dashed, "\alpha"] \\
& & \begin{array}{l}\texttt{letrec}\ \langle\alpha_{E_1}\rangle\cdot E_1\\ \texttt{in letrec}\ \langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad\qquad\ \langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \quad\texttt{in}\ \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array} \arrow[d, dashed, "\mathsf{SR,llet}"] \\
& & \begin{array}{l}\texttt{letrec}\ \langle\alpha_{E_1}\rangle\cdot E_1;\\ \qquad\qquad\langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad\qquad\langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \texttt{in}\ \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array} \arrow[d, phantom, "\sim_\alpha"] \\
\begin{array}{l}\texttt{letrec}\\ \ E_1; E_2\\ \texttt{in letrec}\\ \qquad E_3\\ \ \ \texttt{in}\ S\end{array} \arrow[r, dashed, "\alpha"'] & \begin{array}{l}\texttt{letrec}\\ \ \langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2\\ \texttt{in letrec}\\ \qquad\langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \quad\ \texttt{in}\ \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array} \arrow[r, dashed, "\mathsf{T,llet}"'] & \begin{array}{l}\texttt{letrec}\\ \ \langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2;\\ \ \langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \texttt{in}\ \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array}
\end{tikzcd}$$

Diagram-Computation with Symbolic α-Renaming

Tasks:
- Perform renaming

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \quad\texttt{letrec } E_2 \texttt{ in}\\ \qquad\texttt{letrec } E_3 \texttt{ in } S\end{array} \arrow[rr, "\text{T,llet}"] \arrow[dd, "\text{SR,llet}"'] & & \boxed{\begin{array}{c}\begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \quad\texttt{letrec } E_2, E_3 \texttt{ in } S\end{array}\\ \downarrow \alpha \\ \begin{array}{l}\texttt{letrec } \langle\alpha_{E_1}\rangle\cdot E_1\\ \texttt{in letrec } \langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad\qquad \langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \quad\texttt{in } \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array}\end{array}} \arrow[d, dashed, "\text{SR,llet}"] \\
 & & \begin{array}{l}\texttt{letrec } \langle\alpha_{E_1}\rangle\cdot E_1;\\ \qquad\quad \langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad\quad \langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \texttt{in } \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array} \arrow[d, phantom, "\sim_\alpha"] \\
\boxed{\begin{array}{l}\texttt{letrec}\\ \quad E_1; E_2\\ \texttt{in letrec}\\ \qquad E_3\\ \quad\texttt{in } S\end{array} \xrightarrow{\alpha} \begin{array}{l}\texttt{letrec}\\ \quad \langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2\\ \texttt{in letrec}\\ \qquad \langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \quad\texttt{in } \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array}} \arrow[rr, dashed, "\text{T,llet}"'] & & \begin{array}{l}\texttt{letrec}\\ \quad \langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2;\\ \quad \langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \texttt{in } \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array}
\end{tikzcd}$$

Diagram-Computation with Symbolic α-Renaming

Tasks:
- Perform renaming
- Rewriting for renamed expressions

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \quad\texttt{letrec } E_2 \texttt{ in}\\ \quad\quad\texttt{letrec } E_3 \texttt{ in } S\end{array} \arrow[rr, "\text{T,llet}"] \arrow[ddd, "\text{SR,llet}"'] & & \begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \quad\texttt{letrec } E_2, E_3 \texttt{ in } S\end{array} \arrow[d, dashed, "\alpha"] \\
 & & \begin{array}{l}\texttt{letrec } \langle\alpha_{E_1}\rangle\cdot E_1\\ \texttt{in letrec } \langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad\qquad \langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \quad\texttt{in } \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array} \arrow[d, dashed, "\text{SR,llet}"] \\
 & & \begin{array}{l}\texttt{letrec } \langle\alpha_{E_1}\rangle\cdot E_1;\\ \qquad \langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad \langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \texttt{in } \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array} \arrow[d, phantom, "\sim_\alpha"] \\
\begin{array}{l}\texttt{letrec}\\ \ E_1; E_2\\ \texttt{in letrec}\\ \quad E_3\\ \ \texttt{in } S\end{array} \arrow[r, dashed, "\alpha"'] & \begin{array}{l}\texttt{letrec}\\ \ \langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2\\ \texttt{in letrec}\\ \qquad \langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \quad\texttt{in } \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array} \arrow[r, dashed, "\text{T,llet}"'] & \begin{array}{l}\texttt{letrec}\\ \ \langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2;\\ \ \langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \texttt{in } \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array}
\end{tikzcd}$$

Diagram-Computation with Symbolic α-Renaming

Tasks:
- Perform renaming
- Rewriting for renamed expressions
- Check alpha-equivalence

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \ \texttt{letrec } E_2 \texttt{ in}\\ \ \ \texttt{letrec } E_3 \texttt{ in } S\end{array} \arrow[rr, "\text{T,llet}"] \arrow[ddd, "\text{SR,llet}"'] & & \begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \ \texttt{letrec } E_2, E_3 \texttt{ in } S\end{array} \arrow[d, dashed, "\alpha"] \\
 & & \begin{array}{l}\texttt{letrec } \langle\alpha_{E_1}\rangle\cdot E_1\\ \texttt{in letrec } \langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad \langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \quad \texttt{in } \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array} \arrow[d, dashed, "\text{SR,llet}"] \\
 & & \begin{array}{l}\texttt{letrec } \langle\alpha_{E_1}\rangle\cdot E_1;\\ \qquad \langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad \langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \texttt{in } \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array} \arrow[d, phantom, "\sim_\alpha"] \\
\begin{array}{l}\texttt{letrec}\\ \ E_1; E_2\\ \texttt{in letrec}\\ \quad E_3\\ \ \texttt{in } S\end{array} \arrow[r, dashed, "\alpha"'] & \begin{array}{l}\texttt{letrec}\\ \ \langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2\\ \texttt{in letrec}\\ \qquad \langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \quad \texttt{in } \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array} \arrow[r, dashed, "\text{T,llet}"'] & \begin{array}{l}\texttt{letrec}\\ \ \langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2;\\ \ \langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \texttt{in } \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array}
\end{tikzcd}$$

Diagram-Computation with Symbolic α-Renaming

Tasks:

- Perform renaming
- Rewriting for renamed expressions
- Check alpha-equivalence
- Simplify renamings
- Refresh renamings

$$\begin{tikzcd}
\begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \quad\texttt{letrec } E_2 \texttt{ in}\\ \quad\quad\texttt{letrec } E_3 \texttt{ in } S\end{array} \arrow[rr, "\mathsf{T,llet}"] \arrow[ddd, "\mathsf{SR,llet}"'] & & \begin{array}{l}\texttt{letrec } E_1 \texttt{ in}\\ \quad\texttt{letrec } E_2, E_3 \texttt{ in } S\end{array} \arrow[d, dashed, "\alpha"] \\
 & & \begin{array}{l}\texttt{letrec } \langle\alpha_{E_1}\rangle\cdot E_1\\ \texttt{in letrec } \langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad\qquad\langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \quad\texttt{in } \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array} \arrow[d, dashed, "\mathsf{SR,llet}"] \\
 & & \begin{array}{l}\texttt{letrec } \langle\alpha_{E_1}\rangle\cdot E_1;\\ \qquad\langle\alpha_{E_2}, LV(\alpha_{E_3}), LV(\alpha_{E_1})\rangle\cdot E_2;\\ \qquad\langle\alpha_{E_3}, LV(\alpha_{E_2}), LV(\alpha_{E_1})\rangle\cdot E_3\\ \texttt{in } \langle\alpha_S, \{LV(\alpha_{E_3}), LV(\alpha_{E_2})\}, LV(\alpha_{E_1})\rangle\cdot S\end{array} \arrow[d, phantom, "\sim_\alpha"] \\
\begin{array}{l}\texttt{letrec}\\ \quad E_1; E_2\\ \texttt{in letrec}\\ \quad\quad E_3\\ \quad\texttt{in } S\end{array} \arrow[r, dashed, "\alpha"'] & \begin{array}{l}\texttt{letrec}\\ \quad\langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2\\ \texttt{in letrec}\\ \qquad\langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \quad\texttt{in } \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array} \arrow[r, dashed, "\mathsf{T,llet}"'] & \begin{array}{l}\texttt{letrec}\\ \quad\langle\alpha_{E_1}, LV(\alpha_{E_2})\rangle\cdot E_1; \langle\alpha_{E_2}, LV(\alpha_{E_1})\rangle\cdot E_2;\\ \quad\langle\alpha_{E_3}, \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot E_3\\ \texttt{in } \langle\alpha_S, LV(\alpha_{E_3}), \{LV(\alpha_{E_2}), LV(\alpha_{E_1})\}\rangle\cdot S\end{array}
\end{tikzcd}$$

Algorithms for the Tasks

In the paper we provide:

- An **algorithm to α-rename** $s \in \texttt{LRSX}$ into $AR(s) \in \texttt{LRSX}\alpha$

Proposition (AR preserves the semantics w.r.t. α-equivalence classes)

- For each $\mathsf{t} \in sem(s)$, there exists $\mathsf{t}' \in sem(AR(s))$ such that $\mathsf{t} \sim_\alpha \mathsf{t}'$.
- For each $\mathsf{t}' \in sem(AR(s))$, there exists $\mathsf{t} \in sem(s)$ such that $\mathsf{t} \sim_\alpha \mathsf{t}'$.
- All $\mathsf{t}' \in sem(AR(s))$ fulfill the DVC.

Algorithms for the Tasks

In the paper we provide:

- An **algorithm to α-rename** $s \in \texttt{LRSX}$ into $AR(s) \in \texttt{LRSX}\alpha$

  Proposition (AR preserves the semantics w.r.t. α-equivalence classes)

  - For each $\mathsf{t} \in sem(s)$, there exists $\mathsf{t}' \in sem(AR(s))$ such that $\mathsf{t} \sim_\alpha \mathsf{t}'$.
  - For each $\mathsf{t}' \in sem(AR(s))$, there exists $\mathsf{t} \in sem(s)$ such that $\mathsf{t} \sim_\alpha \mathsf{t}'$.
  - All $\mathsf{t}' \in sem(AR(s))$ fulfill the DVC.

- A **matching algorithm** to solve $(s, \nabla) \trianglelefteq (s', \Delta)$ where $s \in \texttt{LRSX}$, $s' \in \texttt{LRSX}\alpha$, such that for matcher σ: $\sigma(s) = s'$ and $\nabla \implies \sigma(\Delta)$.

  Theorem: The matching algorithm for $\texttt{LRSX}\alpha$ is sound.

Algorithms for the Tasks

In the paper we provide:

- An **algorithm to α-rename** $s \in \texttt{LRSX}$ into $AR(s) \in \texttt{LRSX}\alpha$

Proposition (AR preserves the semantics w.r.t. α-equivalence classes)

- For each $\mathsf{t} \in sem(s)$, there exists $\mathsf{t}' \in sem(AR(s))$ such that $\mathsf{t} \sim_\alpha \mathsf{t}'$.
- For each $\mathsf{t}' \in sem(AR(s))$, there exists $\mathsf{t} \in sem(s)$ such that $\mathsf{t} \sim_\alpha \mathsf{t}'$.
- All $\mathsf{t}' \in sem(AR(s))$ fulfill the DVC.

- A **matching algorithm** to solve $(s, \nabla) \trianglelefteq (s', \Delta)$ where $s \in \texttt{LRSX}$, $s' \in \texttt{LRSX}\alpha$, such that for matcher σ: $\sigma(s) = s'$ and $\nabla \implies \sigma(\Delta)$.

Theorem: The matching algorithm for $\texttt{LRSX}\alpha$ is sound.

- A **test for extended α-equivalence** for constrained $\texttt{LRSX}\alpha$-expressions

Theorem (Soundness of the extended α-equivalence test)

If (s, Δ) and (s', Δ') pass the extended α-equivalence test, then

- for all τ, ρ: $\tau \circ \rho$ satisfies Δ iff $\tau \circ \rho$ satisfies Δ'
- for all τ, ρ: $\tau(\rho(s)) \sim_\alpha \tau(\rho(s'))$.

Simplification of Renamings

- Simplification removes renaming components that **have no effect**
- Simplification is required for other tasks, for instance before checking α-equivalence
- Example:
  In $\langle \dots LV(\alpha_E) \dots \rangle \cdot S$, component $LV(\alpha_E)$ can be removed, if there is an NCC $(S, \texttt{letrec}\ E\ \texttt{in}\ [\cdot])$.
- In the paper: An **inference system to simplify renamings**

Theorem (Soundness of simplification)

If the inference system simplifies (s, Δ) into (s', Δ), then $sem(s, \Delta) = sem(s', \Delta)$.

Refreshing α-Renamings

Rewriting with copying rules may destroy the DVC:

$$\begin{array}{ll} & \texttt{letrec}\ \alpha_{X,1} \cdot X = \alpha_{S,1} \cdot S\ \texttt{in var}\ \alpha_{X,1} \cdot X \\ \to & \texttt{letrec}\ \alpha_{X,1} \cdot X = \alpha_{S,1} \cdot S\ \texttt{in var}\ \alpha_{S,1} \cdot S \end{array}$$

Solution: Refresh the α-renaming:

- **Renumber** the α-renamings (using fresh numbers)
- **Add renaming components** if neccessary.

Proposition (Refreshing preserves the semantics w.r.t. α-equivalence)

- $\mathsf{t} \in sem(s, \Delta) \implies \exists \mathsf{t}' \in sem(refresh(s, \Delta))$ with $\mathsf{t} \sim_\alpha \mathsf{t}'$
- $\mathsf{t}' \in sem(refresh(s, \Delta)) \implies \exists \mathsf{t} \in sem(s, \Delta)$ with $\mathsf{t} \sim_\alpha \mathsf{t}'$.

Experiments

- LRSX Tool available from `http://goethe.link/LRSXTOOL`
- computes diagrams and performs the automated induction
- two exemplary call-by-need calculi:
  - L_{need}: letrec-lambda-calculus, analysis of 16 program transformations
  - LR: extension of L_{need} by case, data constructors, Haskell's seq-operator, analysis of 43 program transformations

	# overlaps	# joins	**# joins using α-renaming**
Calculus L_{need}			
forking	2 242	5 425	**93**
commuting	3 001	7 273	**1 402**
Calculus LR			
forking	87 041	391 264	**73 601**
commuting	107 333	429 104	**93 230**

Conclusion & Further work

Conclusion

- Formalism for representing symbolic α-renamings
- Sound algorithms for renaming, matching, equivalence test, refreshing, and simplification
- Experiments show that in about 20% of the overlaps, α-renaming helps for joining

Further work

- Other applications / meta-languages for our approach
- Completeness of the algorithms