Algorithms for Extended Alpha-Equivalence and Complexity

Manfred Schmidt-Schauß, Conrad Rau, David Sabel

Goethe-University, Frankfurt, Germany

RTA 2013, Eindhoven, The Netherlands

Motivation

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Reasoning, deduction, rewriting, program transformation . . .
requires to **identify expressions**

Functional core languages have (recursive) **bindings**, e.g.

letrec
  map $= \lambda f, xs.$case xs of $\{$[] -> []; $(y : ys)$ -> $(f\ y) : ($map $f\ ys)\}$;
  square $= \lambda x.x * x$;
  myList $= [1, 2, 3]$
in map square myList

- These bindings are **sets**, i.e. they are **commutable**
- Identify expressions **upto extended α-equivalence**:

  α-renaming and commutation of bindings

Questions

- What is the **complexity** of deciding extended α-equivalence?
- Is there a difference for languages with **non-recursive** `let`?
- Find **efficient algorithms** for **special cases**.
- Complexity of extended α-equivalence in **process calculi**?

Extended α-Equivalence for `let`-languages

Abstract language CH with recursive `let`, where $c \in \Sigma$

$$s_i \in \mathcal{L}_{\mathsf{CH}} ::= x \mid c(s_1, \ldots, s_{\mathrm{ar}(c)}) \mid \lambda x.s \mid \texttt{letrec}\ x_1 = s_1; \ldots; x_n = s_n\ \texttt{in}\ s$$

Extended α-Equivalence $\simeq_{\alpha,\mathsf{CH}}$ in CH:

$$s \simeq_{\alpha,\mathsf{CH}} t \text{ iff } s \xleftrightarrow{\alpha \vee comm,*} t \text{ where}$$

- $s \xrightarrow{\alpha} t$ is α-renaming
- $C[\texttt{letrec}\ \ldots; x_i = s_i; \ldots, x_j = s_j; \ldots\ \texttt{in}\ s]$
  $\xrightarrow{comm} C[\texttt{letrec}\ \ldots; x_j = s_j; \ldots; x_i = s_i; \ldots \texttt{in}\ s]$

CHNR: Variant of CH with **non-recursive** `let` instead of `letrec`

Graph Isomorphism

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Graph Isomorphism

Undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** iff there exists a bijection $\phi : V_1 \to V_2$ such that $(v, w) \in E_1 \iff (\phi(v), \phi(w)) \in E_2$

Graph Isomorphism Problem ($\mathbf{GI}$)

Graph-isomorphism ($\mathbf{GI}$) is the following problem: Given two finite (unlabelled, undirected) graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are G_1 and G_2 isomorphic?

- $\mathbf{P} \subseteq \mathbf{GI} \subseteq \mathbf{NP}$
- $\mathbf{GI}$ is neither known to be in $\mathbf{P}$ nor $\mathbf{NP}$-hard
- A lot of other isomorphism problems on labelled / directed graphs are $\mathbf{GI}$-complete (see e.g. Booth & Colbourn' 79)

GI-Hardness of Extended α-Equivalence

Theorem

Deciding $\simeq_{\alpha,\mathsf{CH}}$ is **GI**-hard.

Proof: Polytime reduction of the Digraph-Isomorphism-Problem:

Digraph $G = (V, E)$ is encoded as:

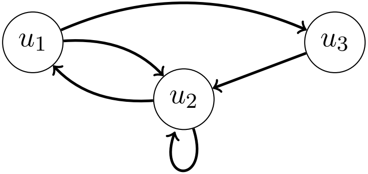
$$enc(G) = \texttt{letrec}\ Env_V, Env_E\ \texttt{in}\ x$$

such that

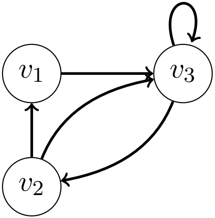
- $Env_V = \bigcup_{v_i \in V} \{v_i = a\}$ where $a \in \Sigma$
- $Env_E = \bigcup_{(v_i,v_j) \in E} \{x_{i,j} = c(v_i, v_j)\}$ where $c \in \Sigma$

Verify: G_1, G_2 are isomorphic $\iff$ $enc(G_1) \simeq_{\alpha,\mathsf{CH}} enc(G_2)$

Example

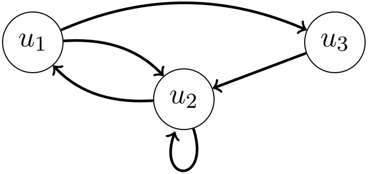


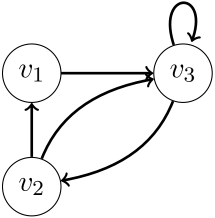
$$
\begin{aligned}
&\texttt{letrec}\ u_1 = a; u_2 = a; u_3 = a;\\
&\qquad x_{1,3} = c(u_1, u_3);\\
&\qquad x_{3,2} = c(u_3, u_2);\\
&\qquad x_{2,2} = c(u_2, u_2);\\
&\qquad x_{2,1} = c(u_2, u_1);\\
&\qquad x_{1,2} = c(u_1, u_2);\\
&\texttt{in}\ x
\end{aligned}
$$



$$
\begin{aligned}
&\texttt{letrec}\ v_1 = a; v_2 = a; v_3 = a;\\
&\qquad x_{1,3} = c(v_1, v_3);\\
&\qquad x_{3,3} = c(v_3, v_3);\\
&\qquad x_{3,2} = c(v_3, v_2);\\
&\qquad x_{2,3} = c(v_2, v_3);\\
&\qquad x_{2,1} = c(v_2, v_1)\\
&\texttt{in}\ x
\end{aligned}
$$

Example

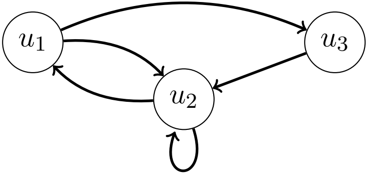


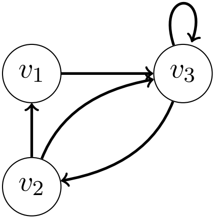


$$
\begin{aligned}
\texttt{letrec}\ & u_3 = a; u_1 = a; u_2 = a;\\
& x_{3,2} = c(u_3, u_2);\\
& x_{2,2} = c(u_2, u_2);\\
& x_{2,1} = c(u_2, u_1);\\
& x_{1,2} = c(u_1, u_2);\\
& x_{1,3} = c(u_1, u_3);\\
\texttt{in}\ & x
\end{aligned}
$$

$$
\begin{aligned}
\texttt{letrec}\ & v_1 = a; v_2 = a; v_3 = a;\\
& x_{1,3} = c(v_1, v_3);\\
& x_{3,3} = c(v_3, v_3);\\
& x_{3,2} = c(v_3, v_2);\\
& x_{2,3} = c(v_2, v_3);\\
& x_{2,1} = c(v_2, v_1)\\
\texttt{in}\ & x
\end{aligned}
$$

Example

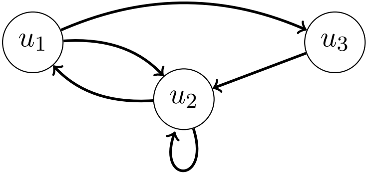


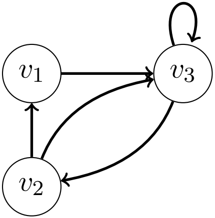


$$
\begin{array}{l}
\texttt{letrec}\ u_3 = a; u_1 = a; u_2 = a; \\
\qquad x_{1,3} = c(u_3, u_2); \\
\qquad x_{3,3} = c(u_2, u_2); \\
\qquad x_{3,2} = c(u_2, u_1); \\
\qquad x_{2,3} = c(u_1, u_2); \\
\qquad x_{2,1} = c(u_1, u_3); \\
\texttt{in}\ x
\end{array}
$$

$$
\begin{array}{l}
\texttt{letrec}\ v_1 = a; v_2 = a; v_3 = a; \\
\qquad x_{1,3} = c(v_1, v_3); \\
\qquad x_{3,3} = c(v_3, v_3); \\
\qquad x_{3,2} = c(v_3, v_2); \\
\qquad x_{2,3} = c(v_2, v_3); \\
\qquad x_{2,1} = c(v_2, v_1) \\
\texttt{in}\ x
\end{array}
$$

Example

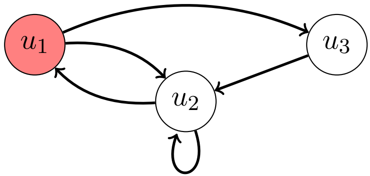


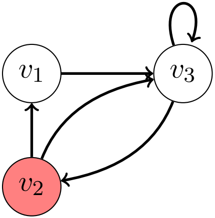


$$
\begin{aligned}
\texttt{letrec}\ & u_3 = a; u_1 = a; u_2 = a; \\
& x_{1,3} = c(u_3, u_2); \\
& x_{3,3} = c(u_2, u_2); \\
& x_{3,2} = c(u_2, u_1); \\
& x_{2,3} = c(u_1, u_2); \\
& x_{2,1} = c(u_1, u_3); \\
\texttt{in}\ & x
\end{aligned}
$$

$$
\begin{aligned}
\texttt{letrec}\ & v_1 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(v_1, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, v_1) \\
\texttt{in}\ & x
\end{aligned}
$$

Example

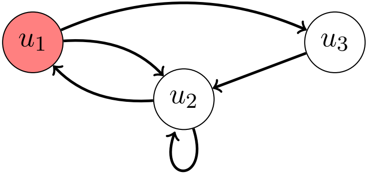


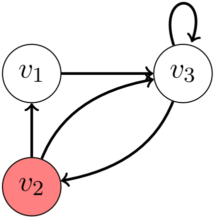


$$\texttt{letrec}\ u_3 = a; v_2 = a; u_2 = a; \\ x_{1,3} = c(u_3, u_2); \\ x_{3,3} = c(u_2, u_2); \\ x_{3,2} = c(u_2, v_2); \\ x_{2,3} = c(v_2, u_2); \\ x_{2,1} = c(v_2, u_3); \\ \texttt{in}\ x$$

$$\texttt{letrec}\ v_1 = a; v_2 = a; v_3 = a; \\ x_{1,3} = c(v_1, v_3); \\ x_{3,3} = c(v_3, v_3); \\ x_{3,2} = c(v_3, v_2); \\ x_{2,3} = c(v_2, v_3); \\ x_{2,1} = c(v_2, v_1) \\ \texttt{in}\ x$$

Example

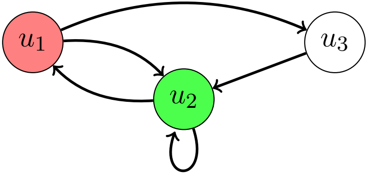


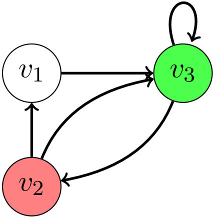


$$\begin{aligned}
\texttt{letrec}\ & u_3 = a; v_2 = a; u_2 = a; \\
& x_{1,3} = c(u_3, u_2); \\
& x_{3,3} = c(u_2, u_2); \\
& x_{3,2} = c(u_2, v_2); \\
& x_{2,3} = c(v_2, u_2); \\
& x_{2,1} = c(v_2, u_3); \\
\texttt{in}\ & x
\end{aligned}$$

$$\begin{aligned}
\texttt{letrec}\ & v_1 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(v_1, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, v_1) \\
\texttt{in}\ & x
\end{aligned}$$

Example

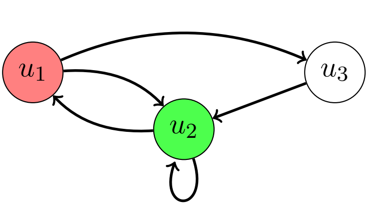


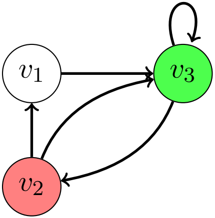


$$\begin{aligned}
\texttt{letrec}\ & u_3 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(u_3, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, u_3); \\
\texttt{in}\ & x
\end{aligned}$$

$$\begin{aligned}
\texttt{letrec}\ & v_1 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(v_1, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, v_1) \\
\texttt{in}\ & x
\end{aligned}$$

Example

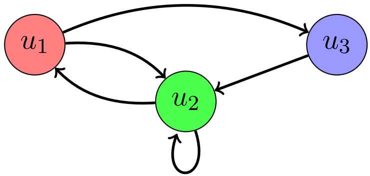


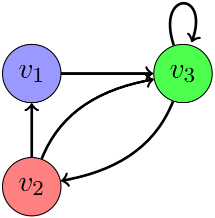


letrec $u_3 = a; v_2 = a; v_3 = a;$
$x_{1,3} = c(u_3, v_3);$
$x_{3,3} = c(v_3, v_3);$
$x_{3,2} = c(v_3, v_2);$
$x_{2,3} = c(v_2, v_3);$
$x_{2,1} = c(v_2, u_3);$
in x

letrec $v_1 = a; v_2 = a; v_3 = a;$
$x_{1,3} = c(v_1, v_3);$
$x_{3,3} = c(v_3, v_3);$
$x_{3,2} = c(v_3, v_2);$
$x_{2,3} = c(v_2, v_3);$
$x_{2,1} = c(v_2, v_1)$
in x

Example

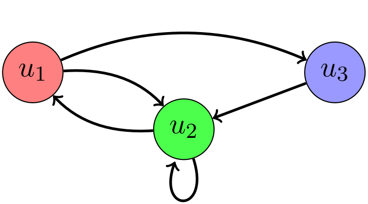


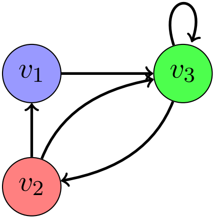


$$
\begin{aligned}
\texttt{letrec}\ & v_1 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(v_1, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, v_1); \\
\texttt{in}\ & x
\end{aligned}
$$

$$
\begin{aligned}
\texttt{letrec}\ & v_1 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(v_1, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, v_1) \\
\texttt{in}\ & x
\end{aligned}
$$

Example





$$
\begin{aligned}
\texttt{letrec}\ & v_1 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(v_1, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, v_1); \\
\texttt{in}\ & x
\end{aligned}
$$

$$
\begin{aligned}
\texttt{letrec}\ & v_1 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(v_1, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, v_1) \\
\texttt{in}\ & x
\end{aligned}
$$

Isomorphism: $\{u_1 \mapsto v_2, u_2 \mapsto v_3, u_3 \mapsto v_1\}$

Easy Variations / Consequences

- Deciding $\simeq_{\alpha,\mathsf{CH}}$ is still **GI**-hard if expressions are **restricted to one-level `letrec`s** (since our encoding uses a one-level `letrec`)
- **Non-recursive `let`**: Deciding $\simeq_{\alpha,\mathsf{CHNR}}$ is **GI**-hard: Use $enc(G) = \texttt{let}\ Env_V\ \texttt{in}\ (\texttt{let}\ Env_E\ \texttt{in}\ x)$
- Hardness also holds for empty signature Σ:
  - replace a by a free variable x_a,
  - replace $c(v_i, v_j)$ by $\texttt{let}\ y = v_i\ \texttt{in}\ v_j$

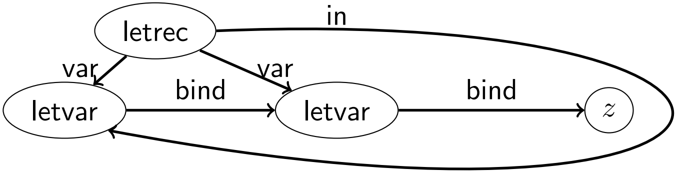
GI-Completeness of Extended α-Equivalence

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

- We use **labelled digraph isomorphism**
- Encode CH-expressions s into a labelled digraph $G(s)$, example:

$$s = \texttt{letrec}\ x = y\ ;\ y = z\ \texttt{in}\ x$$

$G(s) =$



- Full encoding is given in the paper
- Verify: $G(s_1), G(s_2)$ are isomorphic iff $s_1 \simeq_{\alpha,\mathsf{CH}} s_2$

Theorem

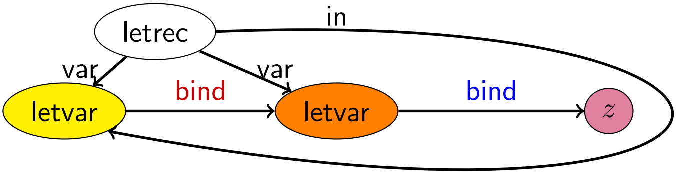
Deciding $\simeq_{\alpha,\mathsf{CH}}$ is **GI**-complete.

GI-Completeness of Extended α-Equivalence

- We use **labelled digraph isomorphism**
- Encode CH-expressions s into a labelled digraph $G(s)$, example:

$$s = \texttt{letrec}\ x = y;\ y = z\ \texttt{in}\ x$$

$G(s) =$



- Full encoding is given in the paper
- Verify: $G(s_1), G(s_2)$ are isomorphic iff $s_1 \simeq_{\alpha,\text{CH}} s_2$

Theorem

Deciding $\simeq_{\alpha,\text{CH}}$ is **GI**-complete.

Special Case: Removing Garbage

Garbage Collection

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Garbage collection (gc): removing unused bindings

$$\texttt{letrec}\ x_1 = s_1; \ldots; x_n = s_n\ \texttt{in}\ t \xrightarrow{gc} t \qquad \text{if } FV(t) \cap \{x_1, \ldots, x_n\} = \emptyset$$

$$\begin{array}{l} \texttt{letrec}\ x_1 = s_1; \ldots; x_n = s_n; \\ \qquad\quad y_1 = t_1; \ldots; y_m = t_m \\ \texttt{in}\ t_{m+1} \end{array} \xrightarrow{gc} \begin{array}{l} \texttt{letrec}\ y_1 = t_1; \ldots; y_m = t_m \\ \texttt{in}\ t_{m+1} \end{array}$$

$$\text{if } \bigcup_{i=1}^{m+1} FV(t_i) \cap \{x_1, \ldots, x_n\} = \emptyset$$

Expression s is **garbage-free** if it is in (gc)-normal form

Lemma

For every CH-expression, its (gc)-normal form can be computed in time $O(n \log n)$

The Garbage-Free Case

Theorem

If s_1, s_2 are garbage free then $s_1 \simeq_{\alpha,\mathsf{CH}} s_2$ can be decided in $O(n \log n)$ where $n = |s_1| + |s_2|$.

Informal argument:

- Since the s_1, s_2 are garbage free they can be **uniquely traversed**:

$$(\texttt{letrec}\ Env\ \texttt{in}\ s)^* \rightarrow (\texttt{letrec}\ Env\ \texttt{in}\ s^*)$$

$$\texttt{letrec}\ \ldots x = s\ \ldots C[x^*] \rightarrow \texttt{letrec}\ \ldots x = s^*\ \ldots C[x]$$

(if $x = s$ was not visited already)

$\ldots$

- This traversal can be used to **fix an order** of the bindings

$$\texttt{letrec}\ x_1 = s_1; \ldots; x_n = s_n\ \texttt{in}\ t \rightarrow \texttt{lrin}(x_{\pi(1)} = s_{\pi(1)}, \ldots, x_{\pi(n)} = s_{\pi(n)}, t)$$

- Now usual algorithms for deciding α-equivalence of terms can be used (see e.g. Calvès & Fernández '10)

The Garbage-Free Case (2)

Formal proof in the paper (sketch):

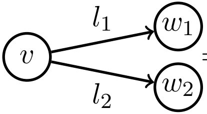
- Compute $G(s_i)$, $i = 1, 2$
- $OO(\cdot)$ removes all var-edges from $G(s_i)$ resulting in $OO(G(s_i))$

The Garbage-Free Case (2)

Formal proof in the paper (sketch):

- Compute $G(s_i)$, $i = 1, 2$
- $OO(\cdot)$ removes all var-edges from $G(s_i)$ resulting in $OO(G(s_i))$
- Since s_i are garbage-free, the graphs $OO(G(s_i))$ are **rooted outgoing-ordered labelled digraphs** (OOLDGs)
- Isomorphism of rooted OOLDGs can be decided in $O(n \log n)$
- $G(s_1)$ and $G(s_2)$ are isom. iff $OO(G(s_1))$ and $OO(G(s_2))$ are isom.

OOLDG: Labelled digraph s.t.



$\implies l_1 \neq l_2$

Rooted OOLDG:

- weakly-connected
- exists root v: every other node is reachable from v

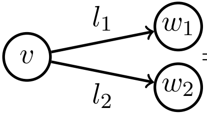
The Garbage-Free Case (2)

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Formal proof in the paper (sketch):

- Compute $G(s_i)$, $i = 1, 2$
- $OO(\cdot)$ removes all var-edges from $G(s_i)$ resulting in $OO(G(s_i))$
- Since s_i are garbage-free, the graphs $OO(G(s_i))$ are **rooted outgoing-ordered labelled digraphs** (OOLDGs)
- Isomorphism of rooted OOLDGs can be decided in $O(n \log n)$
- $G(s_1)$ and $G(s_2)$ are isom. iff $OO(G(s_1))$ and $OO(G(s_2))$ are isom.

OOLDG: Labelled digraph s.t.

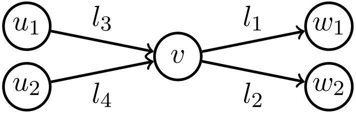


$\implies l_1 \neq l_2$

Rooted OOLDG:

- weakly-connected
- exists root v: every other node is reachable from v

OOLDGs vs. OLDGs



- **Outgoing ordered LDG (OOLDG)**:
  $l_1 \neq l_2$, but $l_3 = l_4$ or $l_3 = l_1$ allowed
- **Ordered LDG (OLDG)**:
  $\{l_1, l_2, l_3, l_4\}$ required to be pairwise distinct

Remark:

- **OOLDG-Isomorphism** is $\mathbf{GI}$-complete (proof in the paper)
- **OLDG-Isomorphism** is in $\mathbf{P}$ (Jian & Bunke, 99)

Alpha-Equivalence Including Garbage Collection

Further consequences:

Extended α-Equivalence up to Garbage-Collection

CH-expressions s, t are **alpha-equivalent up to garbage-collection** written as $s \simeq_{\alpha,gc,\mathsf{CH}} t$, iff the (gc)-normal forms s' and t' of s and t are alpha-equivalent.

Theorem

$s_1 \simeq_{\alpha,gc,\mathsf{CH}} s_2$ can be decided in $O(n \log n)$ where $n = |s_1| + |s_2|$.

Applications

Extended α-equivalence is **GI**-complete in

- several `letrec`-calculi (Ariola'95, Ariola & Blom'97,. . .)
- extended and non-deterministic `letrec`-calculi (Moran, Sands & Carlsson '03, S. & Schmidt-Schauß'08,. . .)
- fragment of Haskell: Recursive functions, data constructors, `letrec`-expressions

Remark: The result **does not hold** for `let`-calculi with non-recursive, **single-binding** let-expressions (e.g. Maraist, Odersky & Wadler '98)

Structural Congruence in the π-Calculus

The π-calculus

Syntax: $P ::= \pi.P \mid (P_1 \mid P_2) \mid\, !\,P \mid \mathbf{0} \mid \nu x.P$
$\pi ::= x(y) \mid \overline{x}\langle y\rangle$ where $x, y \in \mathcal{N}$

Milner's structural congruence $\equiv$:
The least congruence satisfying the equations

$$
\begin{array}{rcl}
P & \equiv & Q, \text{ if } P \text{ and } Q \text{ are } \alpha\text{-equivalent} \\
P_1 \mid (P_2 \mid P_3) & \equiv & (P_1 \mid P_2) \mid P_3 \\
P_1 \mid P_2 & \equiv & P_2 \mid P_1 \\
P \mid \mathbf{0} & \equiv & P \\
\nu z.\nu w.P & \equiv & \nu w.\nu z.P \\
\nu z.\mathbf{0} & \equiv & \mathbf{0} \\
\nu z.(P_1 \mid P_2) & \equiv & P_1 \mid \nu z.P_2, \text{ if } z \notin \text{fn}(P_1) \\
!\,P & \equiv & P \mid\, !\,P
\end{array}
$$

Open Question: Is $\equiv$ decidable?

π-Calculus: Specific Cases and Results (1)

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Lemma (see also (Khomenko & Meyer '09))

Structural congruence $\equiv$ is $\mathbf{GI}$-hard even **without replication**.

Alternative proof: Polytime reduction of Digraph-Isomorphism:

Encode digraph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, $E = \{e_1, \ldots, e_m\}$ as

$\varphi(G) := \nu v_1, \ldots, v_n.(\varphi(v_1) \mid \ldots \mid \varphi(v_n) \mid \varphi(e_1) \mid \ldots \mid \varphi(e_m))$ where

- for $v_i \in V$: $\varphi(v_i) = \overline{v_i}\langle a\rangle.0$
- for $e_i = (v_j, v_k) \in E$: $\varphi(e_i) = v_j(v_k).0$

Then $\varphi(G_1) \equiv \varphi(G_2) \iff G_1, G_2$ are isomorphic.

π-Calculus: Specific Cases and Results (2)

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Fragment **with replication** but **without binders**

$$s, s_i \in \mathcal{PIR} := C \mid (s_1 \mid s_2) \mid \mathbf{!}\, s \qquad (C \text{ represents constants})$$

Structural congruence $\equiv_{\mathcal{PIR}}$ is the least congruence satisfying

$$\begin{array}{lcl} (s_1 \mid s_2) & \equiv_{\mathcal{PIR}} & (s_2 \mid s_1) \\ (s_1 \mid (s_2 \mid s_3)) & \equiv_{\mathcal{PIR}} & ((s_1 \mid s_2) \mid s_3) \\ \mathbf{!}\, s & \equiv_{\mathcal{PIR}} & s \mid \mathbf{!}\, s \end{array}$$

π-Calculus: Specific Cases and Results (2)

Fragment **with replication** but **without binders**

$$s, s_i \in \mathcal{PIR} := C \mid (s_1 \mathbin{|} s_2) \mid \mathbf{!}\, s \qquad (C \text{ represents constants})$$

Structural congruence $\equiv_{\mathcal{PIR}}$ is the least congruence satisfying

$$\begin{array}{lll} (s_1 \mathbin{|} s_2) & \equiv_{\mathcal{PIR}} & (s_2 \mathbin{|} s_1) \\ (s_1 \mathbin{|} (s_2 \mathbin{|} s_3)) & \equiv_{\mathcal{PIR}} & ((s_1 \mathbin{|} s_2) \mathbin{|} s_3) \\ \mathbf{!}\, s & \equiv_{\mathcal{PIR}} & s \mathbin{|} \mathbf{!}\, s \end{array}$$

Theorem

Deciding $s_1 \equiv_{\mathcal{PIR}} s_2$ is $\mathbf{EXPSPACE}$-complete

Proof: In $\mathbf{EXPSPACE}$ was shown by Engelfriet & Gelsema' 07.
Hardness: Reduction of the word problem over commutative semigroups

Remark: Structural congruence in the full π-calculus with replication is thus $\mathbf{EXPSPACE}$-hard, however **decidability** is **still open**.

Conclusion

- Extended α-equivalence in `let`- / `letrec`-calculi is $\mathbf{GI}$-complete
- Complexity arises from garbage bindings (unless $\mathbf{GI} \neq \mathbf{P}$)
- Including garbage-collection in the equivalence makes the decision problem efficiently solvable.
- π-calculus with replication:
  Deciding structural congruence is a very hard problem