Algorithms for Extended Alpha-Equivalence and Complexity

Manfred Schmidt-Schauß, Conrad Rau, David Sabel

Goethe-University, Frankfurt, Germany

RTA 2013, Eindhoven, The Netherlands
Motivation

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

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

\[
\text{letrec}
\]
\[
\begin{align*}
\text{map} &= \lambda f, xs. \text{case } xs \text{ of } \\
& \quad \{ \text{[]} \to \text{[]} ; (y : ys) \to (f y) : (\text{map } f \ ys) \}; \\
\text{square} &= \lambda x. x \ast x; \\
\text{myList} &= [1, 2, 3] \\
in \text{map square myList}
\end{align*}
\]

- These bindings are sets, i.e. they are commutable
- Identify expressions upto extended \(\alpha\)-equivalence:
  \(\alpha\)-renaming and commutation of bindings
Questions

- What is the complexity of deciding extended $\alpha$-equivalence?
- Is there a difference for languages with non-recursive let?
- Find efficient algorithms for special cases.
- Complexity of extended $\alpha$-equivalence in process calculi?
Extended $\alpha$-Equivalence for let-languages

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

\[ s_i \in L_{CH} ::= x \mid c(s_1, \ldots, s_{\text{ar}(c)}) \mid \lambda x.s \mid \text{letrec } x_1 = s_1; \ldots; x_n = s_n \text{ in } s \]

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

\[ s \simeq_{\alpha, CH} t \text{ iff } s \xleftarrow{\alpha \cup \text{comm}, \ast} t \text{ where } \]

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

CHNR: Variant of CH with non-recursive let instead of letrec
Graph Isomorphism

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

Graph Isomorphism Problem (GI)

Graph-isomorphism (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?

- $P \subseteq GI \subseteq NP$
- GI is neither known to be in $P$ nor $NP$-hard
- A lot of other isomorphism problems on labelled / directed graphs are GI-complete (see e.g. Booth & Colboum’ 79)
GI-Hardness of Extended $\alpha$-Equivalence

**Theorem**

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

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

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

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

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 $\iff$ $\text{enc}(G_1) \simeq_{\alpha,CH} \text{enc}(G_2)$
Example

\[
\begin{align*}
\text{letrec } & u_1 = a; u_2 = a; u_3 = a; \\
& x_{1,3} = c(u_1, u_3); \\
& x_{3,2} = c(u_3, u_2); \\
& x_{2,2} = c(u_2, u_2); \\
& x_{2,1} = c(u_2, u_1); \\
& x_{1,2} = c(u_1, u_2); \\
\text{in } x
\end{align*}
\]

\[
\begin{align*}
\text{letrec } & v_1 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(v_1, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, v_1) \\
\text{in } x
\end{align*}
\]
Example

letrec \( u_3 = a; u_1 = a; u_2 = a \);

\[
\begin{align*}
x_{3,2} &= c(u_3, u_2); \\
x_{2,2} &= c(u_2, u_2); \\
x_{2,1} &= c(u_2, u_1); \\
x_{1,2} &= c(u_1, u_2); \\
x_{1,3} &= c(u_1, u_3);
\end{align*}
\]

in \( x \)

letrec \( v_1 = a; v_2 = a; v_3 = a \);

\[
\begin{align*}
x_{1,3} &= c(v_1, v_3); \\
x_{3,3} &= c(v_3, v_3); \\
x_{3,2} &= c(v_3, v_2); \\
x_{2,3} &= c(v_2, v_3); \\
x_{2,1} &= c(v_2, v_1)
\end{align*}
\]

in \( x \)
Example

\[
\begin{align*}
\text{letrec } & u_3 = a; u_1 = a; u_2 = a; \\
& x_{1,3} = c(u_3, u_2); \\
& x_{3,3} = c(u_2, u_2); \\
& x_{3,2} = c(u_2, u_1); \\
& x_{2,3} = c(u_1, u_2); \\
& x_{2,1} = c(u_1, u_3); \\
in x
\end{align*}
\]

\[
\begin{align*}
\text{letrec } & v_1 = a; v_2 = a; v_3 = a; \\
& x_{1,3} = c(v_1, v_3); \\
& x_{3,3} = c(v_3, v_3); \\
& x_{3,2} = c(v_3, v_2); \\
& x_{2,3} = c(v_2, v_3); \\
& x_{2,1} = c(v_2, v_1); \\
in x
\end{align*}
\]
Example

letrec $u_3 = a; u_1 = a; u_2 = a$;

\[
\begin{align*}
    x_{1,3} &= c(u_3, u_2); \\
    x_{3,3} &= c(u_2, u_2); \\
    x_{3,2} &= c(u_2, u_1); \\
    x_{2,3} &= c(u_1, u_2); \\
    x_{2,1} &= c(u_1, u_3);
\end{align*}
\]

in $x$

letrec $v_1 = a; v_2 = a; v_3 = a$;

\[
\begin{align*}
    x_{1,3} &= c(v_1, v_3); \\
    x_{3,3} &= c(v_3, v_3); \\
    x_{3,2} &= c(v_3, v_2); \\
    x_{2,3} &= c(v_2, v_3); \\
    x_{2,1} &= c(v_2, v_1)
\end{align*}
\]

in $x$
Example

letrec \( u_3 = a; v_2 = a; u_2 = a; \)
\[
\begin{align*}
x_{1,3} &= c(u_3, u_2); \\
x_{3,3} &= c(u_2, u_2); \\
x_{3,2} &= c(u_2, v_2); \\
x_{2,3} &= c(v_2, u_2); \\
x_{2,1} &= c(v_2, u_3);
\end{align*}
\]
in \( x \)

\[
\begin{align*}
x_{1,3} &= c(v_1, v_3); \\
x_{3,3} &= c(v_3, v_3); \\
x_{3,2} &= c(v_3, v_2); \\
x_{2,3} &= c(v_2, v_3); \\
x_{2,1} &= c(v_2, v_1);
\end{align*}
\]
in \( x \)
Example

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

Isomorphism: \{ \]
\[ u_1 \mapsto v_2, \quad u_2 \mapsto v_3, \quad u_3 \mapsto v_1 \]
\[ \]

\[ \text{letrec } v_1 = a; v_2 = a; v_3 = a; \]
\[ x_{1,3} = c(v_1, v_3); \]
\[ x_{3,3} = c(v_3, v_3); \]
\[ x_{3,2} = c(v_3, v_2); \]
\[ x_{2,3} = c(v_2, v_3); \]
\[ x_{2,1} = c(v_2, v_1); \]
\[ \text{in } x \]
Example

\[
\text{letrec } u_3 = a; v_2 = a; v_3 = a; \\
x_{1,3} = c(u_3, v_3); \\
x_{3,3} = c(v_3, v_3); \\
x_{3,2} = c(v_3, v_2); \\
x_{2,3} = c(v_2, v_3); \\
x_{2,1} = c(v_2, u_3); \\
\text{in } x
\]

\[
\text{letrec } v_1 = a; v_2 = a; v_3 = a; \\
x_{1,3} = c(v_1, v_3); \\
x_{3,3} = c(v_3, v_3); \\
x_{3,2} = c(v_3, v_2); \\
x_{2,3} = c(v_2, v_3); \\
x_{2,1} = c(v_2, v_1); \\
\text{in } x
\]
**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

\[
\text{letrec } v_1 = a; v_2 = a; v_3 = a; \\
\quad x_{1,3} = c(v_1, v_3); \\
\quad x_{3,3} = c(v_3, v_3); \\
\quad x_{3,2} = c(v_3, v_2); \\
\quad x_{2,3} = c(v_2, v_3); \\
\quad x_{2,1} = c(v_2, v_1); \\
\text{in } x
\]

\[
\text{letrec } v_1 = a; v_2 = a; v_3 = a; \\
\quad x_{1,3} = c(v_1, v_3); \\
\quad x_{3,3} = c(v_3, v_3); \\
\quad x_{3,2} = c(v_3, v_2); \\
\quad x_{2,3} = c(v_2, v_3); \\
\quad x_{2,1} = c(v_2, v_1); \\
\text{in } x
\]
Example

\[
\text{letrec } v_1 = a; v_2 = a; v_3 = a; \\
\quad x_{1,3} = c(v_1, v_3); \\
\quad x_{3,3} = c(v_3, v_3); \\
\quad x_{3,2} = c(v_3, v_2); \\
\quad x_{2,3} = c(v_2, v_3); \\
\quad x_{2,1} = c(v_2, v_1); \\
\text{in } x
\]

Isomorphism: \{ \text{red} u_1 \mapsto \text{green} v_2, \text{green} u_2 \mapsto \text{red} v_3, \text{red} u_3 \mapsto \text{green} v_1 \}
Easy Variations / Consequences

- Deciding $\simeq_{\alpha,CH}$ is still GI-hard if expressions are restricted to one-level letrecs (since our encoding uses a one-level letrec)

- **Non-recursive let**: Deciding $\simeq_{\alpha,CHNR}$ is GI-hard:
  Use $\text{enc}(G) = \text{let Env}_V \text{ in } (\text{let Env}_E \text{ in } x)$

- Hardness also holds for empty signature $\Sigma$:
  - replace $a$ by a free variable $x_a$,
  - replace $c(v_i, v_j)$ by let $y = v_i$ in $v_j$
GI-Completeness of Extended $\alpha$-Equivalence

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

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

![Diagram of labelled digraph](image)

- 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.
GI-Completeness of Extended $\alpha$-Equivalence

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

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

**Diagram:**

- $G(s) =$
- 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 \((gc)\): removing unused bindings

\[
\text{letrec } x_1 = s_1; \ldots; x_n = s_n \text{ in } t \xrightarrow{gc} t \quad \text{if } FV(t) \cap \{x_1, \ldots, x_n\} = \emptyset
\]

\[
\text{letrec } x_1 = s_1; \ldots; x_n = s_n; \\
y_1 = t_1; \ldots; y_m = t_m \xrightarrow{gc} \text{letrec } y_1 = t_1; \ldots; y_m = t_m \text{ in } t_{m+1} \\
\text{in } t_{m+1} \quad \text{if } \bigcup_{i=1}^{m+1} FV(t_i) \cap \{x_1, \ldots, x_n\} = \emptyset
\]

Expression \(s\) is \textbf{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 \( s_1 \simeq_{\alpha, \text{CH}} s_2 \) 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:
  
  \[
  (\text{letrec Env in } s)^* \rightarrow (\text{letrec Env in } s^*)
  \]
  
  \[
  \text{letrec } \ldots x = s \ldots C[x^*] \rightarrow \text{letrec } \ldots x = s^* \ldots C[x]
  \]
  
  (if \( x = s \) was not visited already)

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

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

- Now usual algorithms for deciding \( \alpha \)-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.

Rooted OOLDG:

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

$$
\begin{align*}
&v \\
&\quad \downarrow \quad \downarrow \\
&l_1 \quad w_1 \quad l_2 \quad w_2 \\
\end{align*}
$$

$\implies l_1 \neq l_2$

**Rooted OOLDG**: 
- weakly-connected
- exists root $v$: every other node is reachable from $v$
OOLDGs vs. OLDGs

- **Outgoing ordered LDG (OOLDG):**
  \[ l_1 \neq l_2, \text{ but } l_3 = l_4 \text{ or } l_3 = l_1 \text{ allowed} \]

- **Ordered LDG (OLDG):**
  \{l_1, l_2, l_3, l_4\} required to be pairwise distinct

Remark:

- **OOLDG-Isomorphism** is GI-complete (proof in the paper)
- **OLDG-Isomorphism** is in \( \mathbb{P} \) (Jian & Bunke, 99)
Further consequences:

**Extended α-Equivalence up to Garbage-Collection**

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

**Theorem**

$s_1 \simeq_{\alpha, gc, CH} s_2$ can be decided in $O(n \log n)$ where $n = |s_1| + |s_2|$. 
Extended $\alpha$-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 $\pi$-Calculus
The \( \pi \)-calculus

Syntax: \( P ::= \pi.P \mid (P_1 \mid P_2) \mid !P \mid 0 \mid \nu x.P \)

\( \pi ::= x(y) \mid \overline{x}\langle y \rangle \)

Milner’s structural congruence \( \equiv \):

The least congruence satisfying the equations

\[
\begin{align*}
P & \equiv Q, \text{ if } P \text{ and } Q \text{ are } \alpha\text{-equivalent} \\
P_1 \mid (P_2 \mid P_3) & \equiv (P_1 \mid P_2) \mid P_3 \\
P_1 \mid P_2 & \equiv P_2 \mid P_1 \\
P \mid 0 & \equiv P \\
\nu z.\nu w.P & \equiv \nu w.\nu z.P \\
\nu z.0 & \equiv 0 \\
\nu z.(P_1 \mid P_2) & \equiv P_1 \mid \nu z.P_2, \text{ if } z \notin \text{fn}(P_1) \\
!P & \equiv P \mid !P
\end{align*}
\]

Open Question: Is \( \equiv \) decidable?
**Lemma (see also (Khomenko & Meyer ’09))**

Structural congruence $\equiv$ is 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

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

where

- for $v_i \in V$: $\varphi(v_i) = \overline{v_i}(a).0$
- 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.
\(\pi\)-Calculus: Specific Cases and Results (2)

Fragment **with replication but without binders**

\[ s, s_i \in \mathcal{P\mathcal{I}\mathcal{R}} := C \mid (s_1 \mid s_2) \mid !s \quad (C \text{ represents constants}) \]

Structural congruence \(\equiv_{\mathcal{P\mathcal{I}\mathcal{R}}}\) is the least congruence satisfying

\[
\begin{align*}
(s_1 \mid s_2) & \equiv_{\mathcal{P\mathcal{I}\mathcal{R}}} (s_2 \mid s_1) \\
(s_1 \mid (s_2 \mid s_3)) & \equiv_{\mathcal{P\mathcal{I}\mathcal{R}}} ((s_1 \mid s_2) \mid s_3) \\
!s & \equiv_{\mathcal{P\mathcal{I}\mathcal{R}}} s \mid !s
\end{align*}
\]
**Fragment** with replication but without binders

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

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

\[
\begin{align*}
(s_1 \mid s_2) & \equiv_{\mathcal{PIR}} (s_2 \mid s_1) \\
(s_1 \mid (s_2 \mid s_3)) & \equiv_{\mathcal{PIR}} ((s_1 \mid s_2) \mid s_3) \\
!s & \equiv_{\mathcal{PIR}} s \mid !s
\end{align*}
\]

**Theorem**

Deciding \(s_1 \equiv_{\mathcal{PIR}} s_2\) is EXPSPACE-complete

**Proof:** In EXPSPACE was shown by Engelfriet & Gelsema’ 07.

**Hardness:** Reduction of the word problem over commutative semigroups

**Remark:** Structural congruence in the full \(\pi\)-calculus with replication is thus EXPSPACE-hard, however decidability is still open.
Conclusion

- Extended $\alpha$-equivalence in let- / letrec-calculi is GI-complete

- Complexity arises from garbage bindings (unless GI $\neq P$)

- Including garbage-collection in the equivalence makes the decision problem efficiently solvable.

- $\pi$-calculus with replication:
  Deciding structural congruence is a very hard problem