Maximal Sharing in the Lambda Calculus with letrec

Clemens Grabmayer
VU University Amsterdam (Dept. of CS)

Jan Rochel
Utrecht University (Dept. of CS)

ICFP 2014
September 1–3, 2014
motivation, questions, and results

motivation

- desirable: increase sharing in programs
  - code that is as compact as possible
  - avoid duplication of reduction work at run-time
- useful: check equality of unfolding semantics of programs

questions

(1): how to maximize sharing in programs?
(2): how to check for unfolding equivalence?

we restrict to $\lambda_{letrec}$, the $\lambda$-calculus with $letrec$

- as abstraction & syntactical core of functional languages

our results:

- efficient methods solving questions (1) and (2) for $\lambda_{letrec}$
outline

- methods consist of the steps:
  - interpretation of \( \lambda_{\text{letrec}} \)-terms as term graphs
    - higher-order: \( \lambda \)-ho-term-graphs
    - first-order: \( \lambda \)-term-graphs
  - bisimilarity & bisimulation collapse of \( \lambda \)-term-graphs
  - readback of \( \lambda \)-term-graphs as \( \lambda_{\text{letrec}} \)-terms

- implementation

- complexity

- extensions and applications
contribution

conceptually

- reason about syntactically expressed sharing via an adequate term graph semantics
- reduction to problems accessible by standard methods

maximal sharing method

- extends ‘maximal sharing’ from first-order terms to higher-order terms (with binding)
- significantly extends common subexpression elimination
- is targeted at maximizing sharing statically
  - with respect to the unfolding semantics
  - not: organize/maximize sharing dynamically during evaluation
maximal sharing: example (fix)

\[ \lambda f. \text{let } r = f(f \ r) \text{ in } r \]
maximal sharing: example (fix)

$$\lambda f. \text{let } r = f (f \ r) \text{ in } r$$

$L$

$L_0$

$$\lambda f. \text{let } r = f \ r \text{ in } r$$
maximal sharing: the method

\[ \lambda f. \text{let } r = f (f \ r) \text{ in } r \]

\[ \lambda f. f (f (\ldots)) \]

\[ \lambda f. \text{let } r = f \ r \text{ in } r \]
maximal sharing: the method

\[ \lambda f. \text{let } r = f (f \ r) \text{ in } r \]

\[ L \]

\[ L_0 \]

\[ \lambda f. \text{let } r = f \ r \text{ in } r \]
maximal sharing: the method

\[
\lambda f. \text{let } r = f (f \; r) \text{ in } r
\]
maximal sharing: the method

\[ \lambda f. \text{let } r = f(f\ r) \text{ in } r \]

\[ [\cdot]_T \]

Maximal Sharing in the Lambda Calculus with letrec

Grabmayer, Rochel
maximal sharing: the method

\[ \lambda f. \text{let } r = f(f \ r) \text{ in } r \]

\[ \lambda f. \text{let } r = f \ r \text{ in } r \]
maximal sharing: the method

\[ \lambda f. \text{let } r = f(f \ r) \text{ in } r \]

\[ \lambda f. f(f(\ldots)) \]

\[ \lambda f. \text{let } r = f\ r \text{ in } r \]
maximal sharing: the method

1. term graph interpretation $\int$ of $\lambda_{\text{letrec}}$-term $L$ as:
   a. higher-order term graph $G = \int L_H$

\[ L \xrightarrow{\int} G \]
maximal sharing: the method

1. term graph interpretation $\llbracket \cdot \rrbracket$.
   of $\lambda_{\text{letrec}}$-term $L$ as:
   a. higher-order term graph $G = \llbracket L \rrbracket^H$
   b. first-order term graph $G = \llbracket L \rrbracket^T$

$L \mapsto \llbracket \cdot \rrbracket^H \mapsto G \mapsto G$
maximal sharing: the method

1. term graph interpretation $\llbracket \cdot \rrbracket$. of $\lambda_{\text{letrec}}$-term $L$ as:
   a. higher-order term graph $G = \llbracket L \rrbracket_{H}$
   b. first-order term graph $G = \llbracket L \rrbracket_{T}$
maximal sharing: the method

1. term graph interpretation $\llbracket \cdot \rrbracket_T$.
   of $\lambda_{\text{letrec}}$-term $L$ as:
   a. higher-order term graph $G = \llbracket L \rrbracket_H$
   b. first-order term graph $G = \llbracket L \rrbracket_T$

2. bisimulation collapse $\Downarrow$
   of f-o term graph $G$ into $G_0$
maximal sharing: the method

1. term graph interpretation $\llbracket \cdot \rrbracket_T$.
   of $\lambda_{\text{letrec}}$-term $L$ as:
   a. higher-order term graph $G = \llbracket L \rrbracket_H$
   b. first-order term graph $G = \llbracket L \rrbracket_T$

2. bisimulation collapse $\downarrow$
   of f-o term graph $G$ into $G_0$
maximal sharing: the method

1. term graph interpretation $\llbracket \cdot \rrbracket_T$.
   of $\lambda_{\text{letrec}}$-term $L$ as:
   a. higher-order term graph $G = \llbracket L \rrbracket_H$
   b. first-order term graph $G = \llbracket L \rrbracket_T$

2. bisimulation collapse $\Downarrow$
   of f-o term graph $G$ into $G_0$

3. readback $rb$
   of f-o term graph $G_0$
   yielding program $L_0 = rb(G_0)$. 

Maximal Sharing in the Lambda Calculus with letrec

Grabmayer, Rochel
 maximal sharing: the method

1. term graph interpretation $\llbracket \cdot \rrbracket$. of $\lambda_{\text{letrec}}$-term $L$ as:
   a. higher-order term graph $G = \llbracket L \rrbracket_H$
   b. first-order term graph $G = \llbracket L \rrbracket_T$

2. bisimulation collapse $\downarrow$ of f-o term graph $G$ into $G_0$

3. readback $\text{rb}$ of f-o term graph $G_0$ yielding program $L_0 = \text{rb}(G_0)$. 

Maximal Sharing in the Lambda Calculus with letrec

Grabmayer, Rochel
unfolding equivalence: example

\[ \lambda f. \text{let } r = f (f \ r) \text{ in } r \]

\[ \lambda f. f (f (\ldots)) \]

\[ \lambda f. \text{let } r = f \ r \text{ in } r \]
unfolding equivalence: example

\[ \lambda f. \text{let } r = f (f \ r) \text{ in } r \]

interpret

\[ L_1 \]

\[ M \]

\[ G_1 \]

\[ \lambda f. \text{let } r = f (f \ r) \text{ in } r \]

\[ \llbracket \cdot \rrbracket_{\infty} \]

\[ \lambda f. f (f (\ldots)) \]

\[ \llbracket \cdot \rrbracket_{\infty} \]

\[ \lambda f. \text{let } r = f \ r \text{ in } r \]
unfolding equivalence: the method

\[ \lambda f. \text{let } r = f (f \ r) \text{ in } r \]

\[ \lambda f. f (f (\ldots)) \]

\[ \lambda f. \text{let } r = f \ r \text{ in } r \]
unfolding equivalence: the method

\[ \lambda f. \text{let } r = f (f r) \text{ in } r \]

\[ \lambda f. f (f (\ldots)) \]

\[ \lambda f. \text{let } r = f r \text{ in } r \]
unfolding equivalence: the method

\[ L_1 \xrightarrow{\lambda \infty} M \]

\[ L_2 \xleftarrow{\lambda \infty} ? \]
unfolding equivalence: the method

1. term graph interpretation $\llbracket \cdot \rrbracket$. of $\lambda_{\text{letrec}}$-term $L_1$ and $L_2$ as:

   a. higher-order term graphs
   \[ G_1 = \llbracket L_1 \rrbracket_H \]

   b. first-order term graphs
   \[ G_1 = \llbracket L_1 \rrbracket_T \]
1. term graph interpretation $\llbracket \cdot \rrbracket$.

   of $\lambda_{\text{letrec}}$-term $L_1$ and $L_2$ as:

   a. higher-order term graphs
   
   \[ G_1 = \llbracket L_1 \rrbracket_H \text{ and } G_2 = \llbracket L_2 \rrbracket_H \]

   b. first-order term graphs
   
   \[ G_1 = \llbracket L_1 \rrbracket_T \text{ and } G_2 = \llbracket L_2 \rrbracket_T \]
unfolding equivalence: the method

1. term graph interpretation $\llbracket \cdot \rrbracket$. 
   of $\lambda_{\text{letrec}}$-term $L_1$ and $L_2$ as:
   a. higher-order term graphs 
      \[ G_1 = \llbracket L_1 \rrbracket_H \text{ and } G_2 = \llbracket L_2 \rrbracket_H \]
   b. first-order term graphs 
      \[ G_1 = \llbracket L_1 \rrbracket_T \text{ and } G_2 = \llbracket L_2 \rrbracket_T \]

2. check bisimilarity 
   of f-o term graphs $G_1$ and $G_2$
interpretation

\[ \text{interpret} \quad L \rightarrow G \]

\[ \text{readback} \quad L_0 \rightarrow G_0 \]

\[ \text{collapse} \quad G \rightarrow G_0 \]
running example

instead of:
\[ \lambda f. \text{let } r = f (f \ r) \text{ in } r \]

we use:
\[ \lambda x. \lambda f. \text{let } r = f (f \ r \ x) \ x \text{ in } r \]
graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r$
**graph interpretation (example 1)**

\[ L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r \]

**syntax tree**
graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in} \ r$

syntax tree (+ recursive backlink)
graph interpretation (example 1)

\[ L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r \]

Syntax tree (+ recursive backlink)
graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f \, r \, x \text{ in } r$

syntax tree (+ recursive backlink, + scopes)
graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r$

syntax tree (+ recursive backlink, + scopes, + binding links)
graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r$

first-order term graph with binding backlinks (+ scope sets)
graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r$

first-order term graph with binding backlinks (+ scope sets)
graph interpretation (example 1)

\[ L_0 = \lambda x. \lambda f. \text{let } r = f \; r \; x \text{ in } r \]
graph interpretation (example 1)

\[ L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r \]

first-order term graph with binding backlinks (+ scope sets)
graph interpretation (example 1)

\[ L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r \]

first-order term graph with scope vertices with backlinks (+ scope sets)
graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r$

first-order term graph with scope vertices with backlinks
graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f \ r \ x \ \text{in } r$

λ-term-graph $[[L_0]]_T$
graph interpretation (example 2)

\[ L = \lambda x. \lambda f. \text{let } r = f (f r x) x \text{ in } r \]
graph interpretation (example 2)

\[ L = \lambda x. \lambda f. \text{let } r = f (f r x) \times \text{in } r \]

\[ \begin{array}{c}
\lambda x \\
\lambda f \\
r \\
\emptyset \\
\emptyset \\
\emptyset \\
f \\
r
\end{array} \]

syntax tree
graph interpretation (example 2)

\[ L = \lambda x. \lambda f. \text{let } r = f (f \ r \ x) \ x \ \text{in} \ r \]

syntax tree (+ recursive backlink)
**graph interpretation (example 2)**

\[ L = \lambda x. \lambda f. \text{let } r = f (f \ r \ x) \ x \text{ in } r \]

\[
\begin{array}{c}
\lambda x \\
\lambda f \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\emptyset \\
\end{array}
\]

\[
\begin{array}{c}
f \\
f \\
x \\
x \\
\end{array}
\]

syntax tree (+ recursive backlink)
graph interpretation (example 2)

\[ L = \lambda x. \lambda f. \text{let } r = f (f \, r \, x) \, x \text{ in } r \]

syntax tree (+ recursive backlink, + scopes)
graph interpretation (example 2)

\[ L = \lambda x. \lambda f. \text{let } r = f (f r x) x \text{ in } r \]

first-order term graph with binding backlinks (+ scope sets)
graph interpretation (example 2)

\( L = \lambda x. \lambda f. \text{let } r = f (f \, r \, x) \text{ in } r \)
graph interpretation (example 2)

\[ L = \lambda x. \lambda f. \text{let } r = f (f r x) x \text{ in } r \]

first-order term graph with scope vertices with backlinks (+ scope sets)
graph interpretation (example 2)

\[ L = \lambda x. \lambda f. \text{let } r = f (f \ r \ x) \ x \ \text{in } r \]

\[ \lambda \text{-term-graph } [L]_T \]
graph interpretation (examples 1 and 2)

\[ \Lambda \]

\[ \Lambda \]

\[ O \]

\[ O \]

\[ O \]

\[ S \]

\[ S \]

\[ L_0 \] \_\_\_

\[ L \] \_\_\_

Maximal Sharing in the Lambda Calculus with letrec

Grabmayer, Rochel
interpretation $\llbracket \cdot \rrbracket_T$: properties (cont.)

interpretation $\lambda_{\text{letrec}}$-term $L \mapsto \lambda$-term-graph $\llbracket L \rrbracket_T$

- defined by induction on structure of $L$
- similar analysis as fully-lazy lambda-lifting
- yields eager-scope $\lambda$-term-graphs: $\sim$ minimal scopes

**Theorem**

For $\lambda_{\text{letrec}}$-terms $L_1$ and $L_2$ it holds: Equality of infinite unfolding coincides with bisimilarity of $\lambda$-term-graph interpretations:

$$\llbracket L_1 \rrbracket_{\lambda} = \llbracket L_2 \rrbracket_{\lambda} \iff \llbracket L_1 \rrbracket_T \leftrightarrow \llbracket L_2 \rrbracket_T$$
interpretation $\llbracket \cdot \rrbracket_T$: properties (cont.)

interpretation $\lambda_{letrec}$-term $L \quad \mapsto \quad \lambda$-term-graph $\llbracket L \rrbracket_T$

- defined by induction on structure of $L$
- similar analysis as fully-lazy lambda-lifting
- yields eager-scope $\lambda$-term-graphs: $\sim$ minimal scopes

**Theorem**

*For $\lambda_{letrec}$-terms $L_1$ and $L_2$ it holds: Equality of infinite unfolding coincides with bisimilarity of $\lambda$-term-graph interpretations:*

$$\llbracket L_1 \rrbracket_{\lambda_{\infty}} = \llbracket L_2 \rrbracket_{\lambda_{\infty}} \quad \iff \quad \llbracket L_1 \rrbracket_T \leftrightarrow \llbracket L_2 \rrbracket_T$$
bisimulation check and collapse

\[ L_1 \xrightarrow{\text{interpret}} G_1 \xrightarrow{\text{check}} G_2 \xrightarrow{\text{interpret}} L_2 \]

\[ L \xrightarrow{\text{interpret}} G \xrightarrow{\text{collapse}} G_0 \xrightarrow{\text{readback}} L_0 \]
bisimulation check between $\lambda$-term-graphs

\[ [L_0]_T \]

\[ [L]_T \]
bisimulation check between $\lambda$-term-graphs

\[
\left[ L_0 \right]_\mathcal{T}
\]

\[
\left[ L \right]_\mathcal{T}
\]
Bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs

\[\llbracket L_0 \rrbracket_T \quad \llbracket L \rrbracket_T\]
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs

\[ [L_0]_T \quad [L]_T \]
bisimulation check between $\lambda$-term-graphs

Maximal Sharing in the Lambda Calculus with letrec

Grabmayer, Rochel
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs

Maximal Sharing in the Lambda Calculus with letrec

Grabmayer, Rochel
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs

$[L_0]_\mathcal{T}$

$[L]_\mathcal{T}$
bisimulation check between $\lambda$-term-graphs

Maximal Sharing in the Lambda Calculus with letrec

Grabmayer, Rochel
bisimulation check between $\lambda$-term-graphs

\[
\begin{align*}
\left[ L_0 \right]_T & \quad \text{and} \quad \left[ L \right]_T
\end{align*}
\]
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs

$\left[ L_0 \right]_T \quad \left[ L \right]_T$
bisimulation check between $\lambda$-term-graphs

\[
\begin{align*}
\mathcal{L}_0 & \quad \mathcal{L} \\
\end{align*}
\]
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs

\[
\left[ L_0 \right]_T \quad \left[ L \right]_T
\]
bisimulation check between $\lambda$-term-graphs

$$\left[ L_0 \right]_\mathcal{T} \quad \left[ L \right]_\mathcal{T}$$
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between \( \lambda \)-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation check between $\lambda$-term-graphs
bisimulation between $\lambda$-term-graphs
bisimilarity between $\lambda$-term-graphs

$$\llbracket L_0 \rrbracket_T \leftrightarrow \llbracket L \rrbracket_T$$
functional bisimilarity and bisimulation collapse
bisimulation collapse: property

Theorem

The class of eager-scope $\lambda$-term-graphs is closed under functional bisimilarity $\Rightarrow$.

$\Rightarrow$ For a $\lambda_{letrec}$-term $L$

the bisimulation collapse of $[L]_T$ is again an eager-scope $\lambda$-term-graph.
readback

defined with property:

\[ L \xrightarrow{\text{eager-scope}} G \xrightarrow{\text{rb}} \]

The readback \( rb \) is a right-inverse of \( J \cdot K^T \) modulo isomorphism /uni2243.

main idea:
1. construct a spanning tree \( T \) of \( G \)
2. using local rules, in a bottom-up traversal of \( T \) synthesize \( L = rb(G) \)

Maximal Sharing in the Lambda Calculus with letrec
Grabmayer, Rochel
readback

defined with property:

\[
L = \text{rb}(G)
\]

deep eager-scope

\[
\boxed{[\cdot]_T}
\]
The readback $rb$ is a right-inverse of $\mathbb{L} \cdot J \cdot K^T$ modulo isomorphism $\simeq$.

**Theorem**

For all eager-scope $\lambda$-term-graphs $G$:

$$(\mathbb{L} \cdot J \cdot K^T \circ rb)(G) \simeq G$$

The readback $rb$ is a right-inverse of $\mathbb{L} \cdot J \cdot K^T$ modulo isomorphism $\simeq$. 
**readback**

defined with property:

\[
\mathcal{L} = \text{rb}(G)
\]

**Theorem**

For all eager-scope λ-term-graphs \(G\):

\[
([\cdot]_T \circ \text{rb})(G) \simeq G
\]

The readback \(\text{rb}\) is a right-inverse of \([\cdot]_T\) modulo isomorphism \(\simeq\).

main idea:

1. construct a spanning tree \(T\) of \(G\)
2. using local rules, in a bottom-up traversal of \(T\) synthesize \(L = \text{rb}(G)\)
implementation

- tool maxsharing on hackage.haskell.org
  - uses Utrecht University Attribute Grammar Compiler (UUAGC)

- examples and explanation
  - in accompanying report
Demo: console output

```
Maximal Sharing in the Lambda Calculus with letrec

Grabmayer, Rochel
```
Demo: generated DFAs
maximal sharing: complexity

1. interpretation
   of \( \lambda_{\text{letrec}} \)-term \( L \)
   as \( \lambda \)-term-graph \( G = \llbracket L \rrbracket_T \)

2. bisimulation collapse \( \downarrow \)
   of f-o term graph \( G \) into \( G_0 \)

3. readback rb
   of f-o term graph \( G_0 \)
   yielding \( \lambda_{\text{letrec}} \)-term \( L_0 = \text{rb}(G_0) \).
maximal sharing: complexity

1. interpretation
   of \( \lambda_{\text{letrec}} \)-term \( L \)
   as \( \lambda \)-term-graph \( G = \llbracket L \rrbracket_T \)

2. bisimulation collapse \( \downarrow \)
   of f-o term graph \( G \) into \( G_0 \)

3. readback rb
   of f-o term graph \( G_0 \)
   yielding \( \lambda_{\text{letrec}} \)-term \( L_0 = \text{rb}(G_0) \).
maximal sharing: complexity

1. interpretation
   of \( \lambda_{\text{letrec}} \)-term \( L \) with \( |L| = n \)
   as \( \lambda \)-term-graph \( G = [L]_\mathcal{T} \)
   \( \blacktriangleright \) in time \( O(n^2) \), size \( |G| \in O(n^2) \).

2. bisimulation collapse \( \downarrow \)
   of f-o term graph \( G \) into \( G_0 \)

3. readback \( \text{rb} \)
   of f-o term graph \( G_0 \)
   yielding \( \lambda_{\text{letrec}} \)-term \( L_0 = \text{rb}(G_0) \).
maximal sharing: complexity

1. interpretation
   of $\lambda_{\text{letrec}}$-term $L$ with $|L| = n$
   as $\lambda$-term-graph $G = \llbracket L \rrbracket_T$
   $\triangleright$ in time $O(n^2)$, size $|G| \in O(n^2)$.

2. bisimulation collapse $\downarrow$
   of f-o term graph $G$ into $G_0$
   $\triangleright$ in time $O(|G| \log |G|) = O(n^2 \log n)$

3. readback rb
   of f-o term graph $G_0$
   yielding $\lambda_{\text{letrec}}$-term $L_0 = \text{rb}(G_0)$.
maximal sharing: complexity

1. interpretation
   of $\lambda_{\text{letrec}}$-term $L$ with $|L| = n$
   as $\lambda$-term-graph $G = \llbracket L \rrbracket_T$
   $\triangleright$ in time $O(n^2)$, size $|G| \in O(n^2)$.

2. bisimulation collapse $\downarrow$ of f-o term graph $G$ into $G_0$
   $\triangleright$ in time $O(|G| \log |G|) = O(n^2 \log n)$

3. readback $\text{rb}$ of f-o term graph $G_0$
   yielding $\lambda_{\text{letrec}}$-term $L_0 = \text{rb}(G_0)$.
   $\triangleright$ in time $O(|G| \log |G|) = O(n^2 \log n)$
maximal sharing: complexity

1. interpretation
   of $\lambda_{\text{letrec}}$-term $L$ with $|L| = n$
   as $\lambda$-term-graph $G = \llbracket L \rrbracket_T$
   $\triangleright$ in time $O(n^2)$, size $|G| \in O(n^2)$.

2. bisimulation collapse $\downarrow$
   of f-o term graph $G$ into $G_0$
   $\triangleright$ in time $O(|G| \log |G|) = O(n^2 \log n)$

3. readback $\text{rb}$
   of f-o term graph $G_0$
   yielding $\lambda_{\text{letrec}}$-term $L_0 = \text{rb}(G_0)$.
   $\triangleright$ in time $O(|G| \log |G|) = O(n^2 \log n)$

**Theorem**

*Computing a maximally compact form $L_0 = (\text{rb} \circ \downarrow \circ \llbracket \cdot \rrbracket_T)(L)$ of $L$ for a $\lambda_{\text{letrec}}$-term $L$ requires time $O(n^2 \log n)$, where $|L| = n$.**
unfolding equivalence: complexity

1. interpretation of $\lambda_{letrec}$-terms $L_1, L_2$ as $\lambda$-term-graphs $G_1 = [L_1]_T$ and $G_2 = [L_2]_T$

2. check bisimilarity of $\lambda$-term-graphs $G_1$ and $G_2$
unfolding equivalence: complexity

1. **interpretation**
   
of \( \lambda_{\text{letrec}} \)-term \( L_1, L_2 \) with \( n = \max \{|L_1|, |L_2|\} \)
   
as \( \lambda \)-term-graphs \( G_1 = \llbracket L_1 \rrbracket_T \) and \( G_2 = \llbracket L_2 \rrbracket_T \)
   
   ▶ in time \( O(n^2) \), sizes \( |G_1|, |G_2| \in O(n^2) \).

2. **check bisimilarity**
   
of \( \lambda \)-term-graphs \( G_1 \) and \( G_2 \)
unfolding equivalence: complexity

1. **interpretation**
   
   of $\lambda_{\text{letrec}}$-term $L_1, L_2$ with $n = \max \{|L_1|, |L_2|\}$
   
   as $\lambda$-term-graphs $G_1 = [[L_1]]_T$ and $G_2 = [[L_2]]_T$

   $\blacktriangleright$ in time $O(n^2)$, sizes $|G_1|, |G_2| \in O(n^2)$.

2. **check bisimilarity**
   
   of $\lambda$-term-graphs $G_1$ and $G_2$

   $\blacktriangleright$ in time $O(|G_i| \alpha(|G_i|)) = O(n^2 \alpha(n))$
unfolding equivalence: complexity

1. interpretation
   of $\lambda_{\text{letrec}}$-term $L_1$, $L_2$ with $n = \max \{|L_1|, |L_2|\}$
   as $\lambda$-term-graphs $G_1 = \llbracket L_1 \rrbracket_T$ and $G_2 = \llbracket L_2 \rrbracket_T$
   in time $O(n^2)$, sizes $|G_1|, |G_2| \in O(n^2)$.

2. check bisimilarity
   of $\lambda$-term-graphs $G_1$ and $G_2$
   in time $O(|G_i| \alpha(|G_i|)) = O(n^2 \alpha(n))$

Theorem

*Deciding whether $\lambda_{\text{letrec}}$-terms $L_1$ and $L_2$ are unfolding-equivalent requires almost quadratic time $O(n^2 \alpha(n))$ for $n = \max \{|L_1|, |L_2|\}.*
extensions

- support for full functional languages
  - work on a Core language with constructors, case statements
  - model these by enriching $\lambda_{letrec}$ with function symbols
  - adapt our method to this $\lambda_{letrec}$-extension

- prevent space leaks caused by disadvantageous sharing
  - identify ‘sharing-unfit’ positions/vertices
  - modify $\lambda$-term-graph interpretation
    in order to constrain the bisimulation collapse
applications

- maximal sharing at run-time
  - repeatedly compactify at run-time
  - possible directly on supercombinator graphs
  - can be coupled with garbage collection

- code improvement
  - detect code duplication
  - provide guidance on how to obtain a more compact form

- function equivalence
  - detecting unfolding equivalence provides partial solution
  - relevant for proof assistants, theorem provers, dependently-typed programming languages