Sharing Equality is Linear

Andrea Condoluci

Joint work with Beniamino Accattoli
Claudio Sacerdoti Coen

chocola meeting
Lyon, September 26th
Structure of the Presentation

Evaluation & Conversion
  Complexity
  Sharing

Related Works

The Theory of Sharing Equality
  $\lambda$-Graphs
  Queries
  Sharing Equivalences

Linear-Time Algorithm
  First-order Check
  Variables Check
Structure of the Presentation

Evaluation & Conversion
  Complexity
  Sharing

Related Works

The Theory of Sharing Equality
  $\lambda$-Graphs
  Queries
  Sharing Equivalences

Linear-Time Algorithm
  First-order Check
  Variables Check
Evaluation & Conversion

- Fix a dialect $\lambda_X$ of the $\lambda$-calculus
- With a deterministic evaluation strategy $\rightarrow_X$
- $nf_X(t)$ is the normal form of $t$ with respect to $\rightarrow_X$

Evaluation
Given $t$, computing $nf_X(t)$

Conversion
Given $t$ and $u$, checking whether $nf_X(t) =_\alpha nf_X(u)$
What is the complexity of evaluation and conversion?

Parameters

- **Input term**: size of the initial term $|t|$
- **Number of steps**: number $n$ such that $t \rightarrow^n_{\times} n f_{\times}(t)$
There exists a family \( \{t_n\}_{n \in \mathbb{N}} \) such that (for all \( \lambda_X \)):

\[
 t_n \xrightarrow[n]{} \_nf_X(t_n)
\]

with

\[
|t_n| \in O(n) \quad \text{and} \quad |n f_X(t_n)| \in \Omega(2^n)
\]

Consequences

- Evaluation is exponential in \( n \) and \( |t| \)
- Conversion is also exponential
Sharing is caring ❤️

Add **sharing** to $\lambda_X$, obtaining $\lambda_{shX}$:

$\lambda_X \quad \longrightarrow \quad \lambda_{shX} \quad \longrightarrow \quad \text{RAM}$

1. Turn to **shared evaluation** $\rightarrow_{shX}$

2. Compute **shared normal forms** $\text{nf}_{shX}(t)$

3. Simulating $\rightarrow_X$ up to **sharing unfolding**

   i.e. so that $\text{nf}_{shX}(t) \downarrow = \text{nf}_X(t)$
Evaluation w/ sharing

Compute $\text{nf}_{shX}(t)$ instead of $\text{nf}_{X}(t)$

Call-by-Value (CbV)

Let $t \rightarrow^{n}_{CbV} \text{nf}_{CbV}(t)$

- Blelloch & Greiner, 1995
  - Polynomial in $|t|$ and $n$
  - …

- Accattoli & Condoluci & Sacerdoti Coen, 2019
  - Linear in $|t|$ and $n$
Conversion w/ Sharing

Given $t$ and $u$:

1. **Evaluation**: computing $\text{nf}_{\text{shX}}(t)$ and $\text{nf}_{\text{shX}}(u)$

2. **Sharing equality**: checking $\text{nf}_{\text{shX}}(t) \downarrow =_{\alpha} \text{nf}_{\text{shX}}(u) \downarrow$

Evaluation is bilinear... what about sharing equality?

Problems

- It can be $t \not=_{\alpha} u$ and yet $t \downarrow =_{\alpha} u \downarrow$

- Sharing unfolding is **exponential**

**Polynomial** sharing equality

→ **Testing without** unfolding
Sharing Equality is Linear

Sharing equality

Given \textit{shared} \( t \) and \( u \), checking \( t\downarrow =_\alpha u\downarrow \)

- Accatoli & Dal Lago, 2012
  Sharing equality is \( O((|t| + |u|)^2) \)
  Conversion is \textit{biquadratic} — thus \textit{reasonable}

- \textbf{This talk}: sharing equality is \( O(|t| + |u|) \)
  Conversion is \textit{bilinear}
Structure of the Presentation

Evaluation & Conversion
  Complexity
  Sharing

Related Works

The Theory of Sharing Equality
  \(\lambda\)-Graphs
  Queries
  Sharing Equivalences

Linear-Time Algorithm
  First-order Check
  Variables Check
Algorithms for Sharing Equality

Sharing equality

Given shared $t$ and $u$, checking $t \downarrow =_{\alpha} u \downarrow$

Let $n := |t| + |u|$

Existing algorithms

- Accattoli & Dal Lago, 2012
  $O(n^2)$ algorithm based on dynamic programming

- Grabmayer & Rochel, 2014
  $O(n \log n)$ algorithm for $\lambda$-terms with letrec
  (more general problem)
Related Problems

- **First-order unification**

- **Nominal unification:**
  Two algorithms, adapting MM and PW, quadratic
  Calvès & Fernandez and Levy & Villaret, 2010-13

- **Nominal matching:** linear only on unshared terms
  Calvès & Fernandez 2010

- **Pattern unification:**
  PW based, claimed linear — seems quadratic
  Qian 1993

- **DFAs equivalence:** pseudo-linear
  Hopcroft & Karp 1971
A simple theory of sharing equality for $\lambda$-terms as DAGs $\rightarrow$ $\lambda$-graphs

A linear-time, two-phases algorithm

1. First-order check: based on PW
2. Variables check: binders and variables are shared correctly

The splitting in two steps comes from the developed theory
Structure of the Presentation

Evaluation & Conversion
  Complexity
  Sharing

Related Works

The Theory of Sharing Equality
  λ-Graphs
  Queries
  Sharing Equivalences

Linear-Time Algorithm
  First-order Check
  Variables Check
Syntax Tree

\[(\lambda x. x (\lambda y. z))((\lambda y. z) z)\]

Three kinds of nodes: App, Abs and Var
(λx. x (λy. z)) ((λy. z) z)

Bound variables (bVar) have a binding edge — the dashed one
Sharing in-degree > 1 (excluding binding edges)

Note: sharing of abstractions and under abstractions
Structural Conditions

\( \lambda (\lambda x.xx) \, xx \) ?

- **DAG**: the graph is acyclic (excluding binding edges)

- **Well-formed scopes**: each \( \lambda \)-node dominates the bVar-nodes it binds
Sharing Equality

Problem
Are the two \( \lambda \)-graphs sharing equal? iff there exists a sharing equivalence.
**Bisimulations**

Sharing equivalence is **roughly a bisimulation**

---

**Definition (Bisimulation)**

A binary relation $B$ over the nodes of a $\lambda$-graph is a **bisimulation** if it is:

- **Homogeneous**: $B$ relates only nodes of the same kind
- **Compatible**: $B$ is closed under the following rules

\[
\begin{align*}
\text{App}(n_1, n_2) & \sim B \text{ App}(m_1, m_2) \\
& \quad \Rightarrow n_1 \sim B m_1 \\
\text{Abs}(n) & \sim B \text{ Abs}(m) \\
& \quad \Rightarrow n \sim B m \\
\text{App}(n_1, n_2) & \sim B \text{ App}(m_1, m_2) \\
& \quad \Rightarrow n_2 \sim B m_2 \\
\text{bVar}(n) & \sim B \text{ bVar}(m) \\
& \quad \Rightarrow n \sim B m
\end{align*}
\]
Sharing Equivalence

**Definition (Sharing equivalence)**
A binary relation \( \equiv \) over the nodes of a \( \lambda \)-graph is a **sharing equivalence** if it is:

- **Equivalence** \( \equiv \) is an equivalence relation
- **Bisimulation** \( \equiv \) is a bisimulation
- **Open** if \( \text{fVar}(x) \equiv \text{fVar}(y) \) then \( x = y \)

**Sanity check**
If \( G \) is a \( \lambda \)-graph and \( \equiv \) is a sharing equivalence over \( G \), then \( G/\equiv \) is a \( \lambda \)-graph.
Example

\[ G \text{ and } \equiv \]

\[ G/\equiv \]
Sharing Equality Problem

Input  A λ-graph $G$ + two root nodes $n$ and $m$ of $G$

Problem  Is there a sharing equivalence $\equiv$ on $G$
such that $n \equiv m$?

More generally:

Input  $G$ + a query $Q$ (any relation over roots)

Problem  Is there a sharing equivalence $\equiv$ on $G$
containing $Q$?
Example

Problem
Is there a sharing equivalence $\equiv$ containing a given query $Q$?
Example

Problem
Is there a sharing equivalence \(\equiv\) containing a given query \(Q\)? **Yep.**
Let $Q$ be a query. Its propagation $Q\downarrow$ is the smallest relation closed under the following rules:

1. $\text{App}(n_1, n_2) \ B \text{App}(m_1, m_2)$
   \[ \frac{\ B}{n_1 \ B \ m_1} \]

2. $\text{Abs}(n) \ B \text{Abs}(m)$
   \[ \frac{\ B}{n \ B \ m} \]

Universality of $Q\downarrow$

If there is an open bisimulation containing $Q$, then $Q\downarrow$ is the smallest open bisimulation containing $Q$.
Spreading Queries

Spreading (·)#

Let $Q$ be a query. Its spreading $Q$# is the smallest equivalence relation closed under:

\[
\begin{align*}
\text{App}(n_1, n_2) & \sim \text{App}(m_1, m_2) \\
& \overset{\text{App}}{\rightarrow} n_1 \sim m_1 \\
& \text{Abs}(n) \sim \text{Abs}(m) \\
& \overset{\text{Abs}}{\rightarrow} n \sim m
\end{align*}
\]

Universality of $Q$#

If there exists a sharing equivalence containing $Q$, then $Q$# is the smallest sharing equivalence containing $Q$. 
Sharing Equality Theorem

There exists a sharing equivalence containing $Q$

$Q \Rightarrow$ is an open bisimulation

$\text{[Q]}$ holds, i.e. $[n] = [m]$ for all $n \sim Q \sim m$
Sharing Equality Theorem

There exists a sharing equivalence containing $Q$.

$Q$ is a sharing equivalence.

$Q$ is an open bisimulation.

$\left\llbracket Q \right\rrbracket$ holds, i.e. $[n] = [m]$ for all $n Q m$. 
Read Back

Application: \([\text{App}(n, m)] := [n][m]\\)

Abstraction: \([\text{Abs}(n)] := \lambda ?. [n]\\)

Bound Variable: \([\text{bVar}(n)] := ?\\)

Free Variable: \([\text{fVar}(x)] := x\\)
Read Back — Locally Nameless

\[
\begin{align*}
\text{Abs} & \quad \text{App} \\
\text{bVar} & \quad \text{App} \\
\text{Abs} & \quad \text{App} \\
\text{fVar} z & \quad \text{App}
\end{align*}
\]

- \( [\tau: r \rightsquigarrow \text{App}(n, m)] := ((\tau \leftarrow): r \rightsquigarrow n)[(\tau \leftarrow): r \rightsquigarrow m] \)
- \( [\tau: r \rightsquigarrow \text{Abs}(n)] := \lambda[(\tau \downarrow): r \rightsquigarrow n] \)
- \( [\tau: r \rightsquigarrow \text{bVar}(n)] := \text{indexOf}(n \mid \tau: r) \)
- \( [\tau: r \rightsquigarrow \text{fVar}(x)] := x \)

\( [\varepsilon: r \rightsquigarrow r] = (\lambda 0 (\lambda z)) ((\lambda z) z) \)
Sharing Equality Theorem

There exists a sharing equivalence containing $Q$

$Q\#$ is a sharing equivalence

$Q\downarrow$ is an open bisimulation

$[Q]$ holds, i.e. $[n] = [m]$ for all $n \sim Q m$
Sharing Equality Theorem

There exists a sharing equivalence containing $\mathcal{Q}$

$\mathcal{Q}\#$ is a sharing equivalence
Structure of the Presentation

Evaluation & Conversion
  Complexity
  Sharing

Related Works

The Theory of Sharing Equality
  λ-Graphs
  Queries
  Sharing Equivalences

Linear-Time Algorithm
  First-order Check
  Variables Check
The Two Phases

Checking sharing equality

Compute $Q\#$, then check that it is a sharing equivalence

Two phases

1. First-order check: is $Q\#$ a FO bisimulation?
2. Variables check: check variables and scopes

Once the first phase is solved, the second one is straightforward

Note: the decomposition relies on the theory
Checking Sharing Equality

Compute $Q^#$, then check that it is a sharing equivalence

Difficulty 1

- $Q^#$ is an equivalence relation
- Equivalence relations have size quadratic in the no. of nodes
- **Idea**: use a *canonic element*-based representation of $Q^#$

Difficulty 2

- Linearity requires to *never* merge equivalence classes
- **Idea** (naïve): propagating $Q$ downward by levels
Paterson & Wegman

**Difficulty 2:** linearity requires to *never* merge equivalence classes

**Paterson & Wegman idea**

1. Start *wherever*
2. To process each node:
   1. Process first the parent nodes
   2. Process the $\sim$-neighbors
   3. Propagate only

Linearity achieved because:

- no global synchronisation
- *canonic*-based representation $\equiv_c$
- PW’s *smart* visit
**First-order Check**

**Data:** an initial state

**Result:** \(\mathcal{F}ail\) or a final state

**Procedure** Main()

```plaintext
dataset foreach node n do
  if canonic(n) undefined then
    BuildClass(n)
  end
end
```

**Procedure** Enqueue\((m, c)\)

```plaintext
case m, c of
  Abs\((m'), Abs\((c')\) ⇒
    create edge \(m' \sim c'\)
  App\((m_1, m_2), App\((c_1, c_2)\) ⇒
    create edges \(m_1 \sim c_1\)
    and \(m_2 \sim c_2\)
  bVar\(_\), bVar\(_\) ⇒ ()
  fVar\(_\), fVar\(_\) ⇒ ()
  _\ refresh \_ ⇒ fail
end

```

**Procedure** BuildClass\((c)\)

```plaintext
canonic(c) := c
visiting(c) := true
queue(c) := \{c\}
while queue(c) is non-empty do
  n := queue(c).pop()
  foreach parent m of n do
    case canonic(m) of
      undefined ⇒ BuildClass(m)
      c' ⇒ if visiting(c') then fail
    end
  end
  foreach \(\sim\) neighbour m of n do
    case canonic(m) of
      undefined ⇒ Enqueue\((m, c)\)
      c' ⇒ if c' ≠ c then fail
    end
  end
visiting(c) := false
```


Variables Check

**Data:** canonic(·) representation of $Q#$

**Result:** is $Q#$ a sharing equivalence?

**Procedure** VarsCheck()

```plaintext
foreach variable node $n$ do
  case $n, \text{canonic}(n)$ of
    fVar($l$), fVar($l'$) =>
      assert \text{canonic}(l) = \text{canonic}(l')
    bVar($x$), bVar($y$) =>
      assert $x = y$
  end
end
```
Conclusions

▶ Consequence: \(\beta\)-conversion is bilinear

▶ A theory of sharing equality **independent** of algorithms

▶ A **first** / **higher**-order decomposition of the problem

▶ A linear **PW-like** algorithm for sharing equality

▶ We implemented the algorithm and verified its complexity

▶ On **ArXiv**: detailed proofs of correctness, completeness, and linearity
Thanks for your attention!