Sharing Equality is Linear

Andrea Condoluci

Joint work with Beniamino Accattoli
Claudio Sacerdoti Coen

chocola meeting
Lyon, September 26th

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

inventeurs du monde numérique

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

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

Evaluation & Conversion

- Fix a dialect λ_X of the λ-calculus
- With a deterministic evaluation strategy $\rightarrow_X$
- $\mathtt{nf}_X(t)$ is the normal form of t with respect to $\rightarrow_X$

Evaluation

Given t, computing $\mathtt{nf}_X(t)$

Conversion

Given t and u, checking whether $\mathtt{nf}_X(t) =_\alpha \mathtt{nf}_X(u)$

Complexity

What is the complexity

of evaluation and conversion?

Parameters

- Input term: size of the initial term $|t|$
- Number of steps: number n such that $t \to_X^n \mathtt{nf}_X(t)$

Size explosion 💥

There exists a family $\{t_n\}_{n\in\mathbb{N}}$ such that (for all λ_X):

$$t_n \ \to_X^n \ \mathtt{nf}_X(t_n)$$

with

$$|t_n| \in O(n) \ \text{ and } \ |\mathtt{nf}_X(t_n)| \in \Omega(2^n)$$

Consequences

- Evaluation is exponential in n and $|t|$
- Conversion is also exponential

Sharing is caring ♥

Add sharing to λ_X, obtaining $\lambda_{\mathtt{sh}X}$:

$$\begin{tikzcd} \lambda_X \arrow[rr, dotted] \arrow[dr] & & \mathrm{RAM} \\ & \lambda_{\mathtt{sh}X} \arrow[ur] & \end{tikzcd}$$

1. Turn to shared evaluation $\to_{\mathtt{sh}X}$
2. Compute shared normal forms $\mathtt{nf}_{\mathtt{sh}X}(t)$
3. Simulating $\to_X$ up to sharing unfolding

   i.e. so that $\mathtt{nf}_{\mathtt{sh}X}(t)\downarrow = \mathtt{nf}_X(t)$

Evaluation w/ sharing

Compute $\mathtt{nf}_{\mathtt{sh}X}(t)$ instead of $\mathtt{nf}_X(t)$

Call-by-Value (CbV)

Let $t \to_{CbV}^{n} \mathtt{nf}_{CbV}(t)$

- 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 $\mathtt{nf}_{\mathtt{sh}X}(t)$ and $\mathtt{nf}_{\mathtt{sh}X}(u)$
2. Sharing equality: checking $\mathtt{nf}_{\mathtt{sh}X}(t)\!\downarrow\, =_\alpha \mathtt{nf}_{\mathtt{sh}X}(u)\!\downarrow$

Evaluation is bilinear... what about sharing equality?

Problems

- It can be $t \neq_\alpha u$ and yet $t\!\downarrow\, =_\alpha u\!\downarrow$
- Sharing unfolding is exponential

Polynomial sharing equality

→ Testing without unfolding

Sharing Equality is Linear

Sharing equality

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

- Accatoli & Dal Lago, 2012

  Sharing equality is $O((|t| + |u|)^2)$

  ➡ Conversion is biquadratic — thus reasonable

- This talk: sharing equality is $O(|t| + |u|)$

  ➡ Conversion is bilinear

Structure of the Presentation

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 λ-terms with `letrec`
  (more general problem)

Related Problems

- First-order unification

  Martelli & Montanari, 1977; Paterson & Wegman, 1978

- 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

This Talk

Andrea Condoluci, Beniamino Accattoli and Claudio Sacerdoti Coen, Sharing Equality is Linear, PPDP 2019, October 7^{th}, 2019 Porto, Portugal.

https://arxiv.org/abs/1907.06101

- A simple theory of sharing equality for λ-terms as DAGs $\Rightarrow$ λ-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

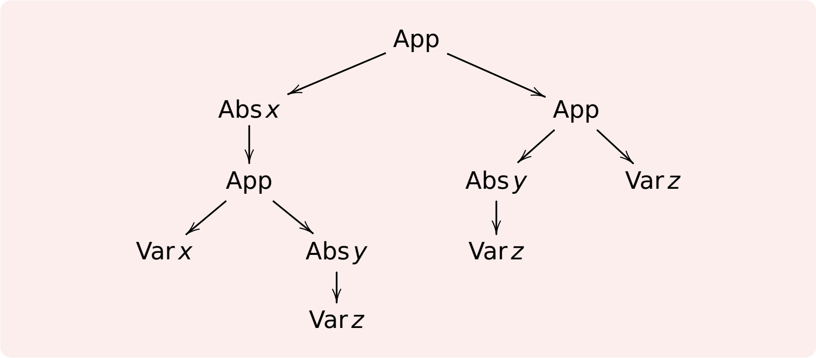
Related Works

The Theory of Sharing Equality
- λ-Graphs
- Queries
- Sharing Equivalences

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

Syntax Tree

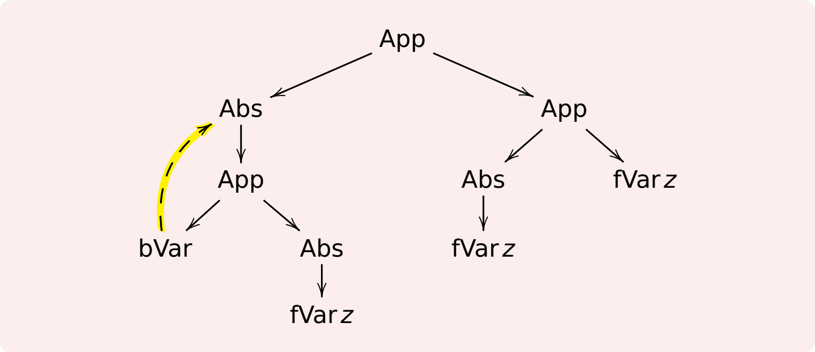
$$(\lambda x.\, x\,(\lambda y.z))\,((\lambda y.z)\,z)$$



Three kinds of nodes: App, Abs and Var

Syntax Tree (ii)

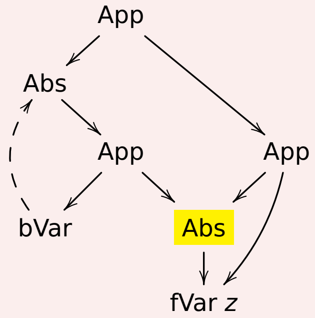
$$(\lambda x.\, x\,(\lambda y.z))\,((\lambda y.z)\,z)$$



Bound variables (bVar) have a binding edge — the dashed one

λ-graph

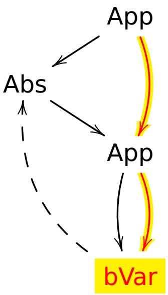
$$(\lambda x.\, x\ \boxed{\lambda y.z}\,)\,(\,\boxed{\lambda y.z}\ z)$$



Sharing in-degree > 1 (excluding binding edges)

Note: sharing of abstractions and under abstractions

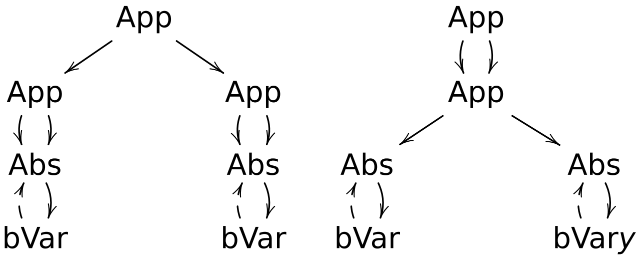
Structural Conditions



¿ $(\lambda x.xx)$ xx ?

- DAG: the graph is acyclic (excluding binding edges)
- Well-formed scopes: each λ-node dominates the bVar-nodes it binds

Sharing Equality



$$((\lambda x.x)\ (\lambda x.x))\ ((\lambda y.y)\ (\lambda y.y))$$

vs.

$$((\lambda x.x)\ (\lambda y.y))\ ((\lambda x.x)\ (\lambda y.y))$$

Problem

Are the two λ-graphs sharing equal?

➡ iff there exists a sharing equivalence

Bisimulations

Sharing equivalence is roughly a bisimulation

Definition (Bisimulation)

A binary relation $\mathcal{B}$ over the nodes of a λ-graph is a bisimulation if it is:

- Homogeneous: $\mathcal{B}$ relates only nodes of the same kind
- Compatible: $\mathcal{B}$ is closed under the following rules

$$\frac{\mathsf{App}(n_1, n_2)\ \mathcal{B}\ \mathsf{App}(m_1, m_2)}{n_1\ \mathcal{B}\ m_1}\ \boxed{\swarrow} \qquad \frac{\mathsf{App}(n_1, n_2)\ \mathcal{B}\ \mathsf{App}(m_1, m_2)}{n_2\ \mathcal{B}\ m_2}\ \boxed{\searrow}$$

$$\frac{\mathsf{Abs}(n)\ \mathcal{B}\ \mathsf{Abs}(m)}{n\ \mathcal{B}\ m}\ \boxed{\downarrow} \qquad \frac{\mathsf{bVar}(n)\ \mathcal{B}\ \mathsf{bVar}(m)}{n\ \mathcal{B}\ m}\ \boxed{\circlearrowright}$$

Sharing Equivalence

Definition (Sharing equivalence)

A binary relation $\equiv$ over the nodes of a λ-graph is a sharing equivalence if it is:

Equivalence $\equiv$ is an equivalence relation

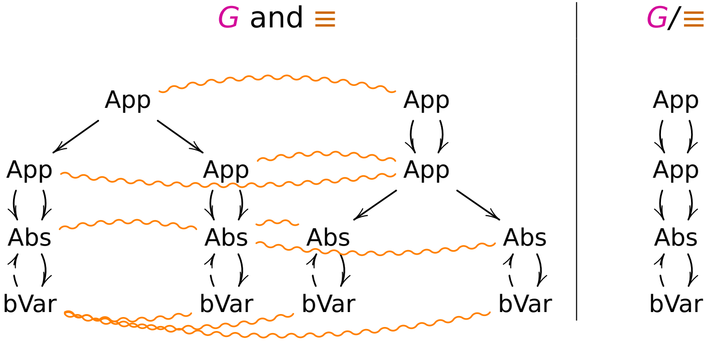
Bisimulation $\equiv$ is a bisimulation

Open if $\mathrm{fVar}(x) \equiv \mathrm{fVar}(y)$ then $x = y$

Sanity check

If G is a λ-graph and $\equiv$ is a sharing equivalence over G, then $G/\equiv$ is a λ-graph.

Example



G and ≡
App
App
App
App
Abs
Abs
Abs
Abs
bVar
bVar
bVar
bVar
G/≡
App
App
Abs
bVar

Sharing Equality Problem

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

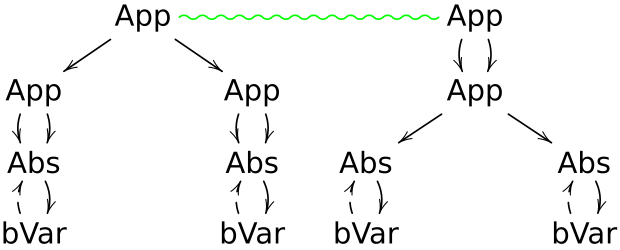
Problem Is there a sharing equivalence $\equiv$ on G such that $n \equiv m$?

More generally:

Input G + a query $\mathcal{Q}$ (any relation over roots)

Problem Is there a sharing equivalence $\equiv$ on G containing $\mathcal{Q}$?

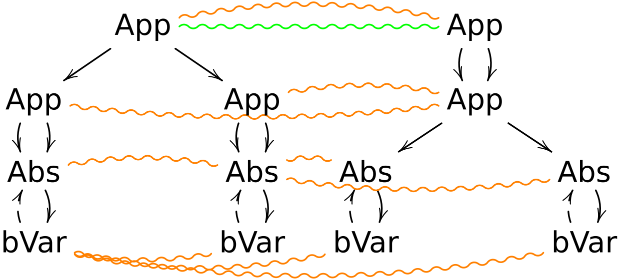
Example



Problem

Is there a sharing equivalence $\equiv$ containing a given query $\mathcal{Q}$?

Example



Problem

Is there a sharing equivalence $\equiv$ containing a given query $\mathcal{Q}$? Yep.

Propagated Queries

Propagation $(\cdot)\Downarrow$

Let $\mathcal{Q}$ be a query. Its propagation $\mathcal{Q}\Downarrow$ is the smallest relation closed under the following rules:

$$\swarrow\ \frac{\mathsf{App}(n_1, n_2)\ \mathcal{B}\ \mathsf{App}(m_1, m_2)}{n_1\ \mathcal{B}\ m_1} \qquad \searrow\ \frac{\mathsf{App}(n_1, n_2)\ \mathcal{B}\ \mathsf{App}(m_1, m_2)}{n_2\ \mathcal{B}\ m_2}$$

$$\downarrow\ \frac{\mathsf{Abs}(n)\ \mathcal{B}\ \mathsf{Abs}(m)}{n\ \mathcal{B}\ m} \qquad \circlearrowright\ \cancel{\frac{\mathsf{bVar}(n)\ \mathcal{B}\ \mathsf{bVar}(m)}{n\ \mathcal{B}\ m}}$$

Universality of $\mathcal{Q}\Downarrow$

If there is an open bisimulation containing $\mathcal{Q}$, then $\mathcal{Q}\Downarrow$ is the smallest open bisimulation containing $\mathcal{Q}$

Spreaded Queries

Spreading $(\cdot)\#$

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

$$\swarrow \quad \frac{\mathsf{App}(n_1, n_2) \mathrel{\mathcal{B}} \mathsf{App}(m_1, m_2)}{n_1 \mathrel{\mathcal{B}} m_1} \qquad \searrow \quad \frac{\mathsf{App}(n_1, n_2) \mathrel{\mathcal{B}} \mathsf{App}(m_1, m_2)}{n_2 \mathrel{\mathcal{B}} m_2}$$

$$\downarrow \quad \frac{\mathsf{Abs}(n) \mathrel{\mathcal{B}} \mathsf{Abs}(m)}{n \mathrel{\mathcal{B}} m} \qquad \circlearrowleft \quad \xcancel{\frac{\mathsf{bVar}(n) \mathrel{\mathcal{B}} \mathsf{bVar}(m)}{n \mathrel{\mathcal{B}} m}}$$

Universality of $\mathcal{Q}\#$

If there exists a sharing equivalence containing $\mathcal{Q}$, then $\mathcal{Q}\#$ is the smallest sharing equivalence containing $\mathcal{Q}$

Sharing Equality Theorem

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

$\Updownarrow$

$[\![\mathcal{Q}]\!]$ holds, *i.e.* $[\![n]\!] = [\![m]\!]$ for all $n \; \mathcal{Q} \; m$

Sharing Equality Theorem

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

$\Updownarrow$

$\mathcal{Q}\#$ is a sharing equivalence

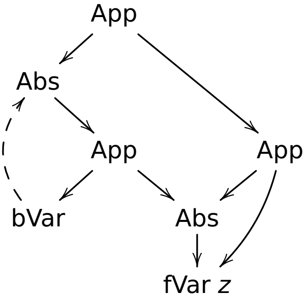
$\Updownarrow$

$\mathcal{Q}\Downarrow$ is an open bisimulation

$\Updownarrow$

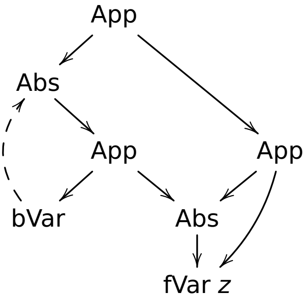
$[\![\mathcal{Q}]\!]$ holds, *i.e.* $[\![n]\!] = [\![m]\!]$ for all $n \; \mathcal{Q} \; m$

Read Back



$$
\begin{array}{rrcl}
\textit{Application}: & [\![\mathrm{App}(n,m)]\!] & := & [\![n]\!]\,[\![m]\!] \\
\textit{Abstraction}: & [\![\mathrm{Abs}(n)]\!] & := & \lambda ?.\ [\![n]\!] \\
\textit{Bound Variable}: & [\![\mathrm{bVar}(n)]\!] & := & ? \\
\textit{Free Variable}: & [\![\mathrm{fVar}(x)]\!] & := & x
\end{array}
$$

Read Back — Locally Nameless



- $[\![\tau : r \leadsto \mathrm{App}(n, m)]\!] \quad := \quad [\![(\tau\swarrow) : r \leadsto n]\!]\,[\![(\tau\searrow) : r \leadsto m]\!]$
- $[\![\tau : r \leadsto \mathrm{Abs}(n)]\!] \quad := \quad \lambda [\![(\tau\downarrow) : r \leadsto n]\!]$
- $[\![\tau : r \leadsto \mathrm{bVar}(n)]\!] \quad := \quad \underline{\mathrm{indexOf}(n \mid \tau : r)}$
- $[\![\tau : r \leadsto \mathrm{fVar}(x)]\!] \quad := \quad x$

$$[\![\epsilon : r \leadsto r]\!] = (\lambda\ \underline{0}\ (\lambda\ z))\ ((\lambda\ z)\ z)$$

Sharing Equality Theorem

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

$\Updownarrow$

$\mathcal{Q}\#$ is a sharing equivalence

$\Updownarrow$

$\mathcal{Q}\Downarrow$ is an open bisimulation

$\Updownarrow$

$[\![\mathcal{Q}]\!]$ holds, *i.e.* $[\![n]\!] = [\![m]\!]$ for all $n \ \mathcal{Q} \ m$

Sharing Equality Theorem

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

$$\Updownarrow$$

$\mathcal{Q}\#$ 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 $\mathcal{Q}\#$, then check that it is a sharing equivalence

Two phases

1. First-order check: is $\mathcal{Q}\#$ 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 $\mathcal{Q}\#$, then check that it is a sharing equivalence

Difficulty 1

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

Difficulty 2

- Linearity requires to never merge equivalence classes
- Idea (naïve): propagating $\mathcal{Q}$ 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:
   2.1 Process first the parent nodes
   2.2 Process the ∼neighbors
   2.3 Propagate only

Linearity achieved because:

- no global synchronisation
- canonic-based representation $=_c$
- PW's smart visit

First-order Check

Data: an initial state
Result: $\mathcal{F}ail$ or a final state

```
Procedure Main()
  foreach node n do
    if canonic(n) undefined
     then
      BuildClass(n)
    end
  end
```

```
Procedure Enqueue(m, c)
  case m, c of
    Abs(m′), Abs(c′) ⇒
      create edge m′ ∼ c′
    App(m1, m2), App(c1, c2) ⇒
      create edges m1 ∼ c1
       and m2 ∼ c2
    bVar(_), bVar(_) ⇒ ()
    fVar_, fVar(_) ⇒ ()
    _, _ ⇒ fail
  end
  canonic(m) := c
  queue(c).push(m)
```

```
Procedure BuildClass(c)
  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 ∼neighbour m of n do
      case canonic(m) of
        undefined ⇒ Enqueue(m, c)
        c′ ⇒ if c′ ≠ c then fail
      end
    end
  end
  visiting(c) := false
```

Variables Check

Data: canonic($\cdot$) representation of $\mathcal{Q}\#$
Result: is $\mathcal{Q}\#$ a sharing equivalence?

Procedure `VarsCheck()`

```
foreach variable node n do
    case n, canonic(n) of
        fVar(l), fVar(l') ⟹
            assert canonic(l) = canonic(l')
        bVar(x), bVar(y) ⟹
            assert x = y
    end
end
```

Conclusions

- Consequence: ⚡ β-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!