Linear Logic and Optimal Reductions

Sharing Graphs

Stefano Guerrini

LIPN - Institut Galilée, Université Paris Nord 13

MPRI - Linear logic and logical paradigms of computation

2009–2010

Our target

To show that sharing graphs are

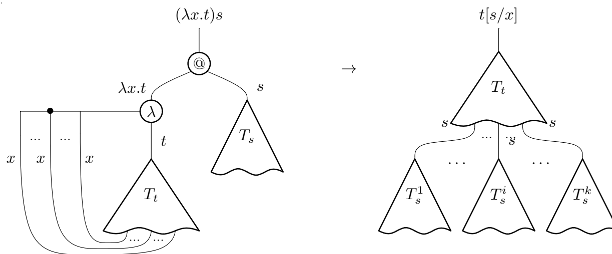
1. a very easy and natural approach to the implementation of rewriting rules *à la* β-rule of λ-calculus
2. easily implementable
3. applicable to any "orthogonal" graph rewriting system
4. an optimal implementation (according to Lèvy) of the underlying system
   - as efficient as the underlying system

Decomposing β-rule

$$(\lambda x.t)s \longrightarrow_\beta t[s/x]$$

β-rule involves two main operations

1. the assignment of the argument t to the variable x
2. the duplication and displacement of t
   - a new copy of t is created for every occurrence of x
   - any copy of t is put in the place of the corresponding occurrence of x



Decomposing β-rule (cont.)

The usual formulation of β-rule (given above)

1. is global
   - it is not an atomic step of some abstract reduction machine
   - counting the number of such rules is not a good measure of reduction complexity
2. it is not efficient (the rule duplicates the redexes in the argument)

Some attempts to address the previous problems:

- Wadsworth's **graph reduction**
  duplicates by-need, but the duplication process remains a global step
- Lévy's **optimal reduction**
  defines a lower-bound to the number of β-rules in a shared setting that tries to avoid useless duplications
- **explicit substitutions**
  internalize into the calculus the atomic steps required by the replacement of variables by terms
- **sharing graphs**
  implement optimal reductions but *go beyond them*

The basic ideas of graph reduction

Some sharing can be achieved by using DAGs in the place of trees

1. We can keep a unique access point to all the occurrences of a variable x (e.g., by backward connecting the occurrences of a variables to its binder).
2. In order to denote that t replaces x, we can replace the pointers to the binder λx with pointers to the root of t.
3. Duplication (the creation of a new instance of t) can be performed only when actually needed, e.g., when we want to reduce a redex $(\lambda y.s)x$ in which x has been replaced by a shared term t.

Not yet a complete and *optimal* solution!

1. From time to time, some global rewriting is required
2. There remain useless duplications of work

An easy exercise

We want to:

define a graph rewriting system that
implements β-rule by means of
local and atomic rewritings only.

or equivalently

We want to:

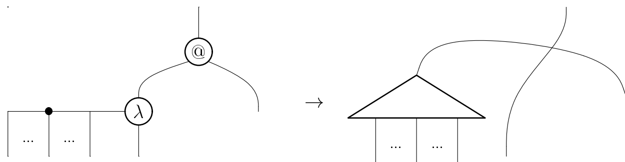
decompose β-rule into
a sequence of atomic graph rewritings.

A solution to the exercise

Add a new node in the graph for representing sharing:

multiplexer (mux) or **fan**

The reduction of a β-rule $(\lambda x.s)t$ introduces a mux that connects the edges corresponding to the occurrences of the variable x to the root of t.

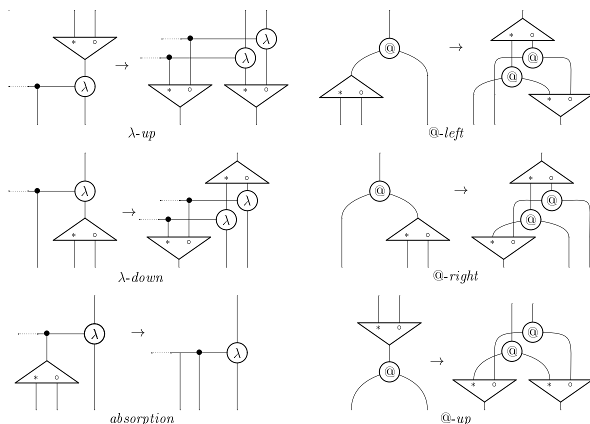


λ-calculus graph duplication rules

After the execution of a β-rule the mux propagates into the argument, duplicating it step-by-step.

Remark

To simplify the presentation, the rules are given for the case of binary muxes only. In general, muxes have n auxiliary ports and one principal port.



Some remarks on the previous rules

- When a mux reaches a $\lambda/$@-node, the mux duplicates the node and splits in two copies.
  The duplication thread of the original mux gives rise to two new duplication threads that can proceed independently and in parallel.
- Duplication threads does not propagate downward only, but upward too, according to the orientation of the corresponding mux.
  E.g., a λ-up rule introduces a mux that propagates upward.
  There are other rules that change the direction of one of the resulting muxes.

Some remarks on absorption

The absorption rule requires some additional words.

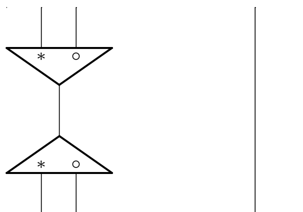
- Absorption is the only rule of the previous ones that erases a mux.
- Absorption corresponds to the end of a duplication thread:
  it corresponds to a duplication that reaches a free variable of the duplicating term
- Absorption is not always sound:
  if during the reduction a mux reaches the occurrence of a variable bound in t, a sound rule would be to duplicate the λ-node and not to erase the mux.

Mux annihilation rule

We can distinguish two kinds of complementary muxes, according to their orientation.

Mux or **fan-in** nodes are downward oriented.
Demux or **fan-out** nodes are upward oriented.

A pair of facing complementary muxes annihilates.



This corresponds to two duplication threads that meet, one moving downward and the other one upward.
The two threads that meet have completed their task.

Soundness

Let us restrict w.l.o.g. to closed λ-terms.

$$
\begin{array}{ccccccccc}
t & & & \xrightarrow[\beta]{} & & & t' & & \lambda\text{-terms} \\
\downarrow & & & & & & \downarrow & & \\
T & \xrightarrow[\beta_s]{} & G_1 & \xrightarrow[\pi_{\not a}]{+} & G_2 & \xrightarrow[\pi_a]{+} & T' & & \text{graphs}
\end{array}
$$

We get soundness provided that one applies the algorithm:

1. rewrite a β-redex by means of a shared β-rule;
2. before the firing of another shared β-rule or of any absorption rule, propagate all the muxes in the graph until they disappear or they reach the binding port of a λ-node;

   NB In this phase, no absorption rule can be fired. At the end of this phase, all the muxes left in the graph are at the binding ports of λ-nodes.
3. remove the muxes left in the graph by means of absorption rules.

Troubles and limits of this solution

- The rewriting system implements β-rule under a given reduction strategy.
- We would like to have a way to determine when absorption rule is sound: the soundness of the rule rests on the particular strategy used.
  In the particular strategy considered above, a mux that reaches the binding port o a λ-node in t will be erased by another mux that will cross the λ-node through its principal (its upper) port.

A more liberal reduction strategy

After firing a β-redex, can we apply another shared β-rule before the completion of the duplication process started by the first shared rule?

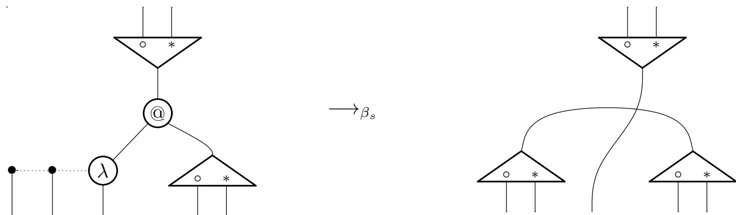
We have to solve at least two problems:

1. a better way to determine when to apply absorption;
2. if we simultaneously fire two β-redexes, it is no longer true that all the muxes in the graph are duplicating the same term and

it is no longer true that two facing muxes annihilate

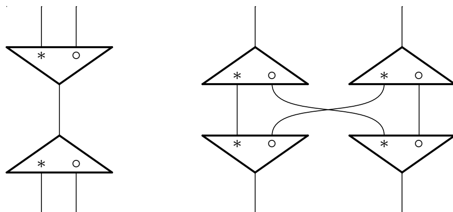
in some cases, two facing muxes must
swap and duplicate each other

Facing muxes that do not annihilate



The mux swap rule

In addition to the annihilation rule we have then the following swap rule:



Problem

How to decide when to apply an annihilation or a swap?

Example

@ — λx — λy

λx: @ (x, x)

λy: @ (λz, @)

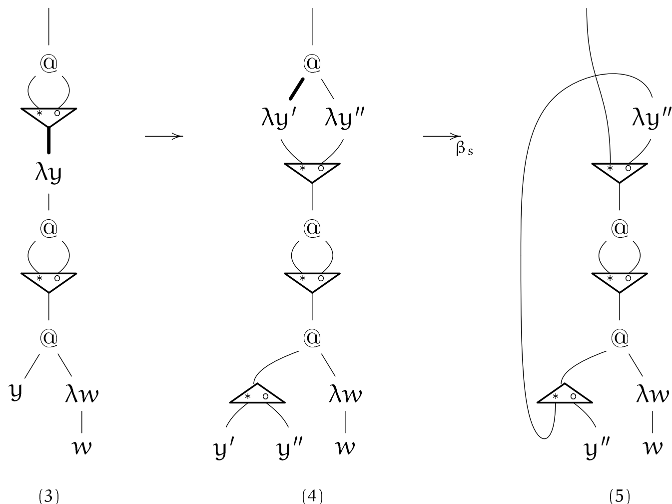
λz: @ (z, z)

@ (y, λw), λw: w

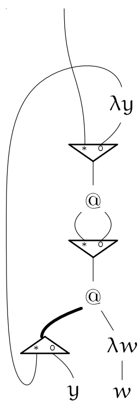
$\longrightarrow_{\beta_s}$

(1) (2) (3)

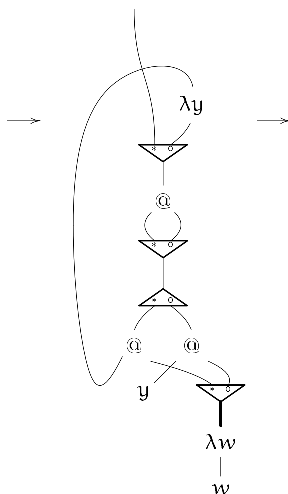
Example (cont.)



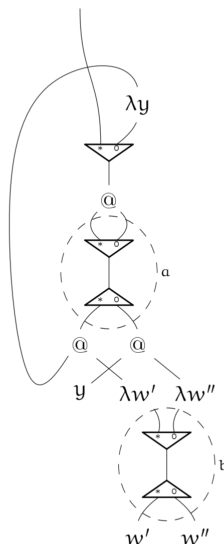
Example (cont.)



→

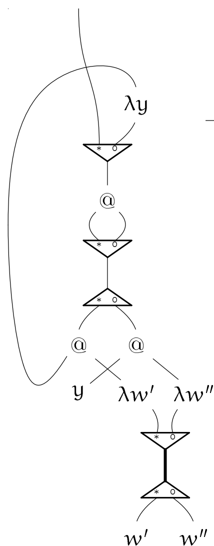


→



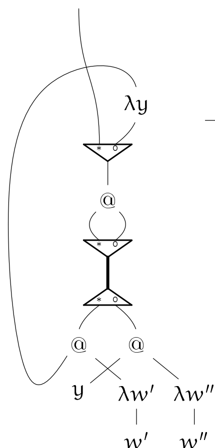
(5) (6) (7)

Example (cont.)



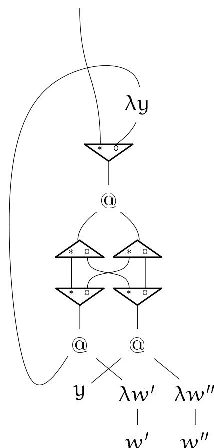
(7)

$\to_{\text{an}}$



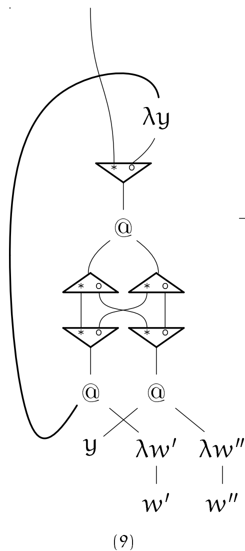
(8)

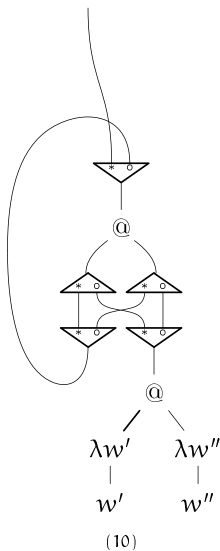
$\to_{\text{sw}}$

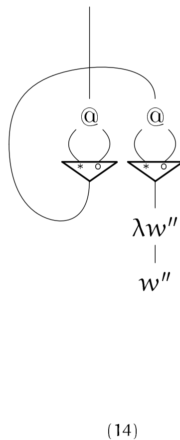


(9)

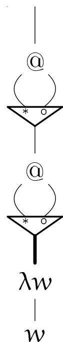
Example (cont.)

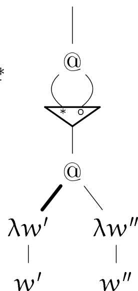


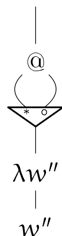


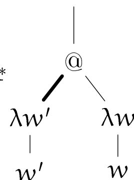


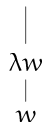
Example (cont.)











How to name muxes?

We need a way to name muxes s.t.:

- if two muxes face with the same name, they annihilate;
- if two muxes face with different names, they swap.

Because of the higher-order nature of λ-calculus, we cannot use a *static naming* of muxes

- assign a new name to a mux at its creation (when firing the corresponding β-redex);
- keep the name on a mux on its copies.

Static naming does not work, as there are cases (even in typed λ-calculus) in which two copies of the same mux must swap.
However, static naming of muxes works in some systems with bound complexity as, for instance, the so-called elementary and light calculi.

Boxes

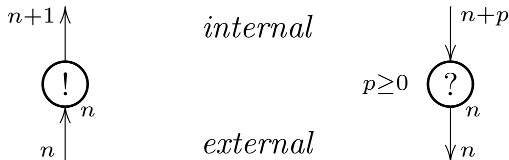
According to the MELL translation of λ-calculus

$$A \to B = !A \multimap B$$

the argument of an application is always enclosed into a box.

A **box** is a subgraph that has a unique **principal door** and many **auxiliary doors**.

Let us introduce two new links for the principal and auxiliary doors of boxes.



Remark. The boxes that we consider are always connected.

The box nesting property

The key property of boxes is that they properly nest.

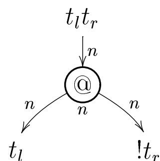
Two boxes B_1 and B_2 cannot partially overlap,

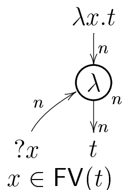
- either B_1 and B_2 are disjoint,
- or B_1 is in B_2, or B_2 is in B_1.

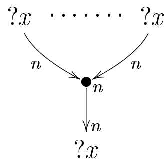
Two boxes may have some auxiliary doors in common, but they always have distinct principal doors.

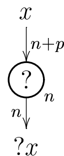
Links with levels

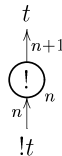
Because of the box nesting property we can assign a level to each link: its **box nesting depth**.







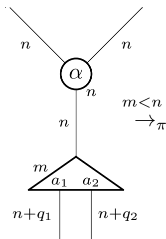




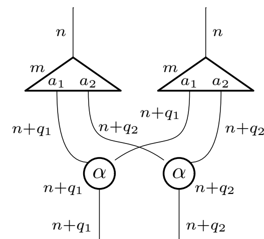
Consequences of the introduction of levels

- The introduction of levels implies that, after a β-rule, the new copies of the argument must be properly reindexed.
- Therefore, muxes must duplicate and reindex the link that cross.
- The level of a mux is a **threshold**: a mux duplicates and reindex every link that it reaches and that has a level lower than its threshold.
- We can recognize when a mux must propagate at the binding port of a λ-node or must be absorbed by it.

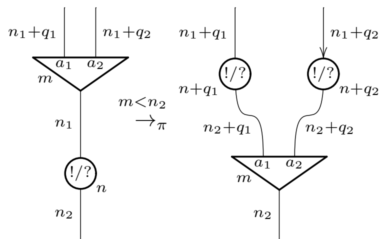
Propagation rules



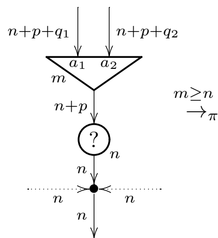
$\overset{m<n}{\rightarrow_\pi}$



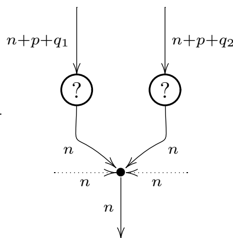
α-propagation



!/?-propagation



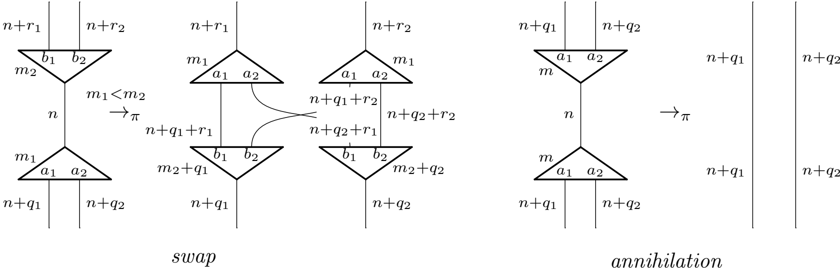
$\overset{m\geq n}{\rightarrow_\pi}$



absorption

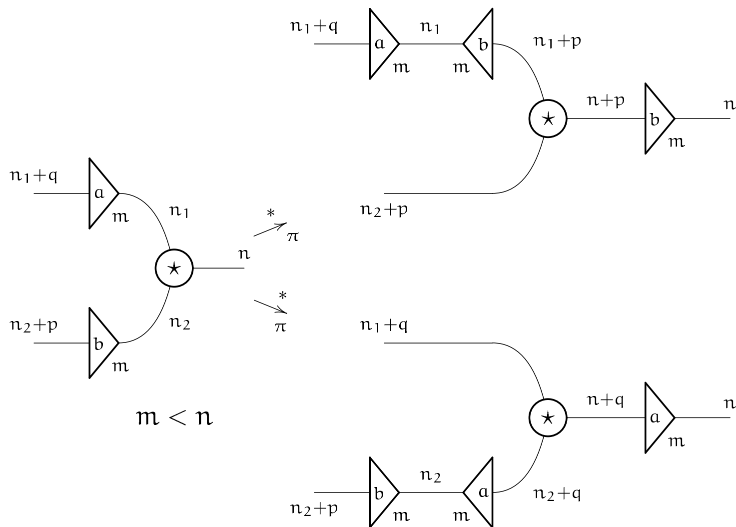
Mux rules

Two muxes annihilate if they meet with the same threshold, or swap otherwise.

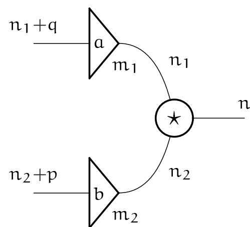


Let us note that in the swap rule, the mux with the threshold $m_1 < m_2$ duplicates and reindex the mux with the threshold m_2.

Critical pairs



Critical pairs (cont.)



$m_1 < m_2 < n, n_1$

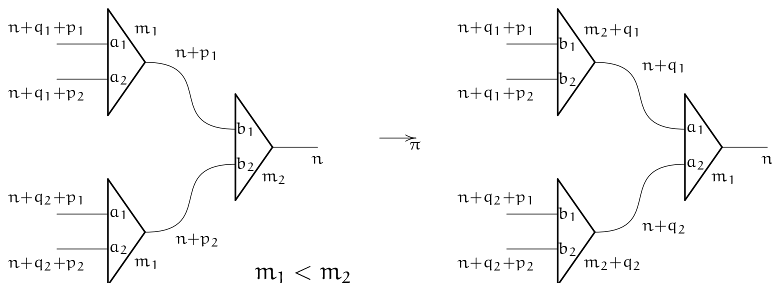
$\overset{*}{\nearrow}_{\pi}$

m2+q
b
n1+q
n1+p+q
n+q+p
a
n+p
b
n
m1
m2
m1
a
n2+p
n2+p+q

$\overset{*}{\searrow}_{\pi}$

m2+q
b
n1+q
n1+p+q
n+q+p
b
n+q
a
n
m2+q
m1
m1
a
n2+p
n2+p+q

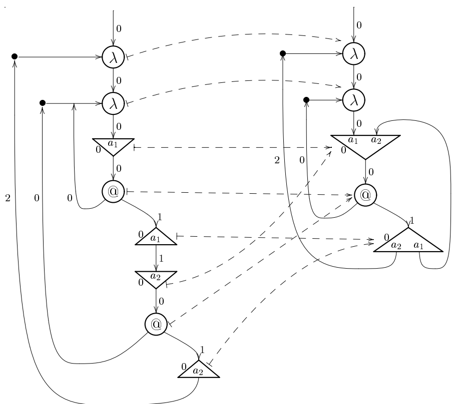
Permutation equivalence



Remark

Permutation equivalence is not part of the implementation, but it is useful to prove the properties of the rewriting system.

Unfolding sharing graphs

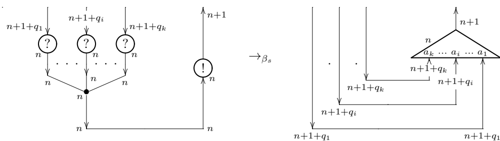


Definition (Sharing morphism)

A *sharing morphism* is a surjective homomorphism of nets which is injective when restricted to the conclusions of the sharing graph (root and free variables) and that preserves the type of the links, the levels of the edges, and the names of the ports to which the edges are connected.

N.B. A sharing mux maps a k-ary mux into a mux with $h \geq k$ ports. However, the ports of the source mux must be a subset of its image.

Unshared β-rule



n

@

n $n+1$

$n+p_i$

λ

n

T_λ $T_@$

$n+p_k$ $n+p_1$

$\longrightarrow_{\beta_u}$

n

T_λ

$n+p_1$ ... ... $n+p_k$

$n+p_i$

a_1 a_i a_k

n n n

$n+1$ $n+1$ $n+1$

$T^1_@$ ... $T^i_@$ ... $T^k_@$

Unshared graphs

Remark

In the following we shall restrict to the case weakening/erasing free, i.e., we shall consider terms of the λI-calculus only.
The proof techniques that we shall show adapts to the case with weakening, but requires to define a reduction that postpone weakenings to the end.

Definition

The sharing graph G_1 is a less shared instance of the sharing graph G when there exists a sharing morphism $M : G_1 \to G$.

- A *lift* is a multiplexer with only one auxiliary port.
- An *unshared graph* is a shared graph in which all the muxes are lifts.
- Let U be an unshared graph. If U is a less shared instance of the sharing graph G, we shall say that U is a *completely unshared instance* of G.

Unshared reductions

- A λ-*tree* is the graph representation of a λ-term.
- A sharing graph G is *proper* when $N \to^* G$, for some λ-tree N.
- An unshared graph is a sharing graph s.t. every mux is unary,
- An unshared graph is proper when $N \to^* G$, for some λ-tree N a sequence of π and unshared β-rule.
- Let U be an unshared graph. $\mathcal{R}_u(U)$ is the graph obtained by replacing lifts every lift by a wire and by properly recomputing the levels assigned to the edges and the nodes of the graph.

Lemma

Let T be a λ-tree and U a proper unshared graph s.t. $T \to_u^* U$, then $\mathcal{R}_u(U)$ is a λ-tree and $T \to_\beta^* \mathcal{R}_u(U)$ by the standard β-reduction (extended to the case with levels associated to the nodes).

$$\mathcal{R}_u(U) = \mathcal{R}_u(U') \quad \text{when} \quad U \to_\pi U'$$
$$\mathcal{R}_u(U) \to_\beta \mathcal{R}_u(U') \quad \text{when} \quad U \to_{\beta_u} U'$$

Readback of unshared graphs

- We aim at proving the following relevant properties of unshared reductions:

Lemma

Let U be a proper unshared graph.

1. *There is no infinite π reduction of U.*
2. *The λ-tree $\mathcal{R}_u(U)$ is the unique π normal form of U.*

- The previous results can be proved by an algebraic semantics of sharing graphs inspired by the so-called geometry of interaction.
- The algebraic semantics allows to define proper unshared graphs directly, without assuming that they are the reducts of some λ-tree.

Simulation property

- Let us write $U \lessdot G$ when
  1. U is a proper unshared graph;
  2. there is a sharing morphism $M : U \to G$.

Lemma (Simulation)

$$\begin{array}{ccc} U & \lessdot & G \\ \rho \Big\downarrow {}_u^+ & & \Big\downarrow r \\ U' & \lessdot & G' \end{array}$$

Corollary

Let G be a proper sharing graph and T a λ-tree s.t. $T \to_s^* G$. Then, there is a proper unshared graph $U \lessdot G$ s.t. $T \to_u^* U$.

- The unshared graph U is said the *least-shared-instance* of G.

Main results

Theorem

Let G be a proper sharing graph.

1. *The π rules are strongly normalizing and confluent on G. The π normal form of G is a λ-tree $\mathcal{R}(G)$.*
2. *The $\beta + \pi$ rewriting rules are confluent on G.*
3. *Let $G \to_s^* G'$, then $\mathcal{R}(G) \to_\beta^* \mathcal{R}(G')$.*

Theorem

Let G be a proper sharing graph and T be a λ-tree s.t. $T \to^* G$. Then $T \to_\beta^* \mathcal{R}(G)$.

- In particular, when the λ-tree T has a β-normal form N, the λ-tree N is the unique $\beta + \pi$ normal form of G.

Summary of the main results

$$\begin{array}{ccc} G & \xrightarrow[\beta_s]{} & G' \\ \downarrow & & \downarrow \\ \mathcal{R}(G) & \xrightarrow[\beta]{+} & \mathcal{R}(G') \end{array} \qquad \begin{array}{ccc} G & \xrightarrow[\pi]{} & G' \\ \downarrow & & \downarrow \\ \mathcal{R}(G) & = & \mathcal{R}(G') \end{array}$$

$$G \begin{array}{l} \xrightarrow[\pi]{*} \mathsf{NF}_\pi(G) \\ \qquad\quad = \\ \mapsto \mathcal{R}(G) \end{array}$$

Sharing graphs optimal reductions

Optimal reductions are just a particular reduction strategy of the previous rewriting system.

It is the reduction that minimize the number of β-reduction without blocking the creation and execution of them.

The optimal rules are the shared β-rule, λ-up, @-left and the mux rules (plus the absorption rule).
The optimal system is an indexed Interaction Net.

$$\begin{array}{ccc} G & \xrightarrow[o]{}^{*} & \mathsf{NF_o}(G) \\ \big\downarrow & & \big\downarrow \\ \mathcal{R}(G) & \xrightarrow[\beta]{}^{*} & \mathsf{NF}(\mathcal{R}(G)) \end{array}$$

The optimal rules

π_{opt} is the subset of π which is an interaction net.
In π_{opt}, a mux interacts with a link only when it points to its principal port.

- The principal port of a linear logic link is its conclusion.
- The principal port of an @-link is its left port, of a λ-link is its up port.

In order to avoid that the computation might get stuck, it is enough to restrict to the π_{opt} rules.

Theorem

The $\beta + \pi_{\text{opt}}$ rewriting rules are optimal.

Theorem

Let G be a proper sharing graph and N be its $\beta + \pi$ normal form. Let G' be a $\beta + \pi_{\text{opt}}$ normal form of G, then $\mathcal{R}(G') = N$, that is, $\mathcal{R}(G') \to_{\pi}^{*} N$.

The optimal rules implement Lévy's optimal reductions, that is every optimal reduction of a λ-tree T corresponds to some reduction by (Lévy's) families of the corresponding λ-term.