The Complexity of Optimal Reduction and Sharing Graphs

Andrea Laretto

Università di Pisa
Dipartimento di Informatica

23 luglio 2021

Introduction

- The problems treated in this seminar:
  - *Is there an optimal way of reducing λ-terms? If so, how?*
  - *What is the complexity of optimal β-reduction?*
- The main topics presented:
  - **Theoretical concepts**
    - Orders of reduction
    - Optimal reduction and Lévy-optimality
    - Wadsworth's technique
    - Sharing graphs
  - **Complexity and results for optimal reduction**
    - Asperti and Mairson (1998)
    - Asperti, Coppola, Martini (2004)
  - **Practical aspects**
    - Examples of Higher-Order reduction
    - The Bologna Optimal Higher-order Machine (BOHM)
    - BOHM: Benchmarks and results
  - **Conclusion and related work**

Theoretical concepts

Preliminary Definitions

- Identity: $I \triangleq \lambda x.x$
- Duplication: $\Delta \triangleq \lambda x.x\ x$

Definition (Redex = Reducible expression)

An application where the left side is a λ-abstraction.

Definition (Innermost reduction)

A reduction strategy where we first apply redexes which contain no other redex inside of them (i.e.: fully reduce the arguments, then substitute).

Definition (Outermost reduction)

A reduction strategy where we first apply redexes which are contained by no other redex (i.e.: substitute the arguments in the body as they are).

Reduction Orders in λ-terms

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

- **Innermost reduction**: fully evaluate the arguments, then apply

$$\begin{aligned} & (\lambda x.\lambda y.x)\ z\ \underline{((\lambda x.x\ x)\ k)} \\ \Longrightarrow_\beta\ & \underline{(\lambda x.\lambda y.x)\ z}\ (k\ k) \\ \Longrightarrow_\beta\ & \underline{(\lambda y.z)\ (k\ k)} \\ \Longrightarrow_\beta\ & z \end{aligned}$$

- **Outermost reduction**: substitute arguments immediately as they are

$$\begin{aligned} & \underline{(\lambda x.\lambda y.x)\ z}\ ((\lambda x.x\ x)\ k) \\ \Longrightarrow_\beta\ & \underline{(\lambda y.z)\ ((\lambda x.x\ x)\ k)} \\ \Longrightarrow_\beta\ & z \end{aligned}$$

- The issue: *doing useless work by applying unneeded reductions*

Reduction Orders in λ-terms

$$(\lambda f.(f\ a)\ (f\ b))\ ((\lambda y.\lambda x.x)\ k)$$

- **Innermost reduction**: fully evaluate the arguments, then apply

$$\begin{aligned}
& (\lambda f.(f\ a)\ (f\ b))\ \underline{((\lambda y.\lambda x.x)\ k)} \\
\Longrightarrow_\beta\ & \underline{(\lambda f.(f\ a)\ (f\ b))\ (\lambda x.x)} \\
\Longrightarrow_\beta\ & \underline{((\lambda x.x)\ a)}\ ((\lambda x.x)\ b) \\
\Longrightarrow_\beta\ & a\ \underline{((\lambda x.x)\ b)} \\
\Longrightarrow_\beta\ & a\ b
\end{aligned}$$

- **Outermost reduction**: substitute arguments immediately as they are

$$\begin{aligned}
& \underline{(\lambda f.(f\ a)\ (f\ b))\ ((\lambda y.\lambda x.x)\ k)} \\
\Longrightarrow_\beta\ & (\underline{((\lambda y.\lambda x.x)\ k)}\ a)\ (((\lambda y.\lambda x.x)\ k)\ b) \\
\Longrightarrow_\beta\ & \underline{((\lambda x.x)\ a)}\ (((\lambda y.\lambda x.x)\ k)\ b) \\
\Longrightarrow_\beta\ & a\ (\underline{((\lambda y.\lambda x.x)\ k)}\ b) \\
\Longrightarrow_\beta\ & a\ \underline{((\lambda x.x)\ b)} \\
\Longrightarrow_\beta\ & a\ b
\end{aligned}$$

- The issue: *duplicating work by copying redexes*

Reduction Orders in λ-terms – Results

- As we have seen, *none of the two reduction orders presented* can reach the normal form in the minimum number of β-reductions

Property (Outermost applies only required redexes)

If a redex is not applied by the outermost reduction, it is not necessary to obtain the normal form of the term.

Theorem (Standardization theorem) [Curry & Feys, 1958]

If a normal form exists, there exists a (standard) outermost reduction to it.

Theorem (Uncomputability of optimal reduction strategies) [Barendregt et al. 1976]

Any *reduction strategy* that always selects the minimal length β-reduction (one redex at a time) for all λ-terms is necessarily uncomputable.

Virtual Redexes, Redex Families

- Inefficiency is hard to avoid! Using **innermost reduction**:

$$
\begin{array}{ll}
 & (\lambda x.x\ I)\ (\lambda y.\underline{\Delta\ (y\ z)}) \\
\Longrightarrow_\beta & \underline{(\lambda x.x\ I)\ (\lambda y.(y\ z)\ (y\ z))} \\
\Longrightarrow_\beta & \underline{(\lambda y.(y\ z)\ (y\ z))\ I} \\
\Longrightarrow_\beta & \underline{(I\ z)}\ (I\ z) \\
\Longrightarrow_\beta & z\ \underline{(I\ z)} \\
\Longrightarrow_\beta & z\ z
\end{array}
$$

- Duplication can also occur in more *subtle* ways: $(y\ z)$ is not *yet* a redex!

Definition (Virtual redex)

An application where the left side could be *a λ-abstraction in the future.*

- Issue: we *individually* reduce terms deriving from a *same (virtual) redex*

Definition (Redex family) [Lévy 1978]

A set of redexes with a common origin (i.e.: residuals of the same redex)

Lévy-Optimality

- No optimal strategy; but a optimal *parallel β-reduction* strategy exists!
- The problem of duplication: *separately* reducing redex of a family

Theorem (Lévy-optimality – parallel β-reduction) [Lévy 1978]

Any relation contracting a whole family of redexes in one step is optimal.

Corollary (Lévy-optimality – no-duplication property) [Lévy 1978]

No redex, explicit or virtual, is ever duplicated during such reduction.

- *How can we achieve this?* By uniquely representing all the redexes in a same redex family in a *collective and* ***shared*** way
- A first efficient idea for the sharing of redexes: ***Wadsworth's technique***

Wadsworth's Technique

Wadsworth's Technique

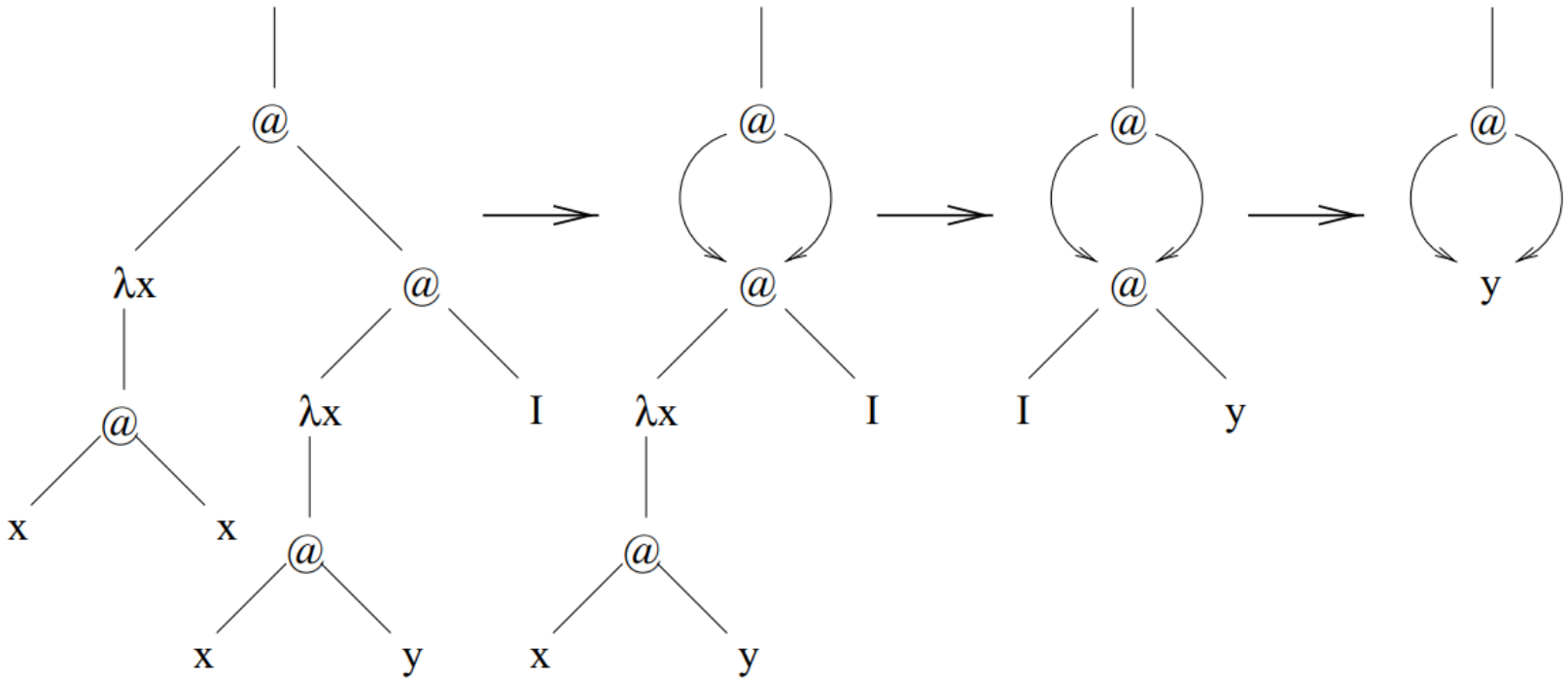
- Introduced in Wadsworth's Ph.D. thesis (1971) as a practical technique for efficiently reducing λ-terms:

  Avoid duplicating work by sharing function arguments

- Implemented in modern lazy functional programming languages with call-by-need (Haskell GHC)
- λ-terms $\rightarrow$ abstract syntax trees + pointers $\approx$ directed acyclic graphs
- Variables in a term are simply pointers to the abstract syntax tree of the argument given to the function
- *Each reduction in a subtree represents one or more reductions in the corresponding term*
- (An **outermost reduction** order is still required to avoid *useless work*)

Wadsworth's Technique – Example

- $M \triangleq (\lambda x.x\ x)\ ((\lambda x.x\ y)\ I)$



$$
\begin{array}{ll}
 & \underline{(\lambda x.x\ x)}\ \underline{((\lambda x.x\ y)\ I)} \\
\Longrightarrow_\beta & \underline{((\lambda x.x\ y)\ I)}\ \underline{((\lambda x.x\ y)\ I)} \\
\Longrightarrow_\beta^* & \underline{(I\ y)}\ \underline{(I\ y)} \\
\Longrightarrow_\beta^* & y\ y
\end{array}
$$

Wadsworth's Technique – Non-optimality

- Unfortunately, Wadsworth's technique is *non-optimal*
- Why: virtual redexes inside functions still need to be duplicated
- For example:

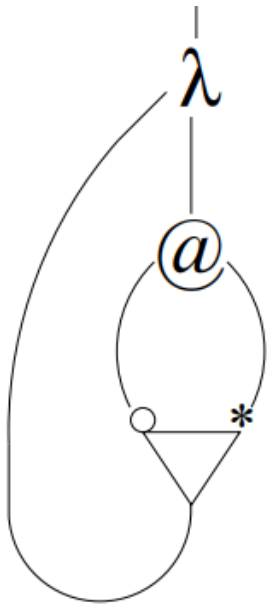
$$\Longrightarrow_\beta \quad \frac{(\lambda f.(f\ a)\ (f\ b))\ (\lambda y.\ \ldots)}{(\underline{(\lambda y.\ \ldots)\ a})\ (\underline{(\lambda y.\ \ldots)\ b})}$$

- Now the function body will inevitably have to be *unshared!*
- This duplicates any internal virtual redexes *inside the function*
- *Observation*: the copy of the function will only differ by the argument, the rest of the body remains the same
- *A more complex sharing and unsharing mechanism is required to maintain optimality, using full (cyclic) graphs:* ***Sharing graphs***

Sharing graphs

Sharing Graphs

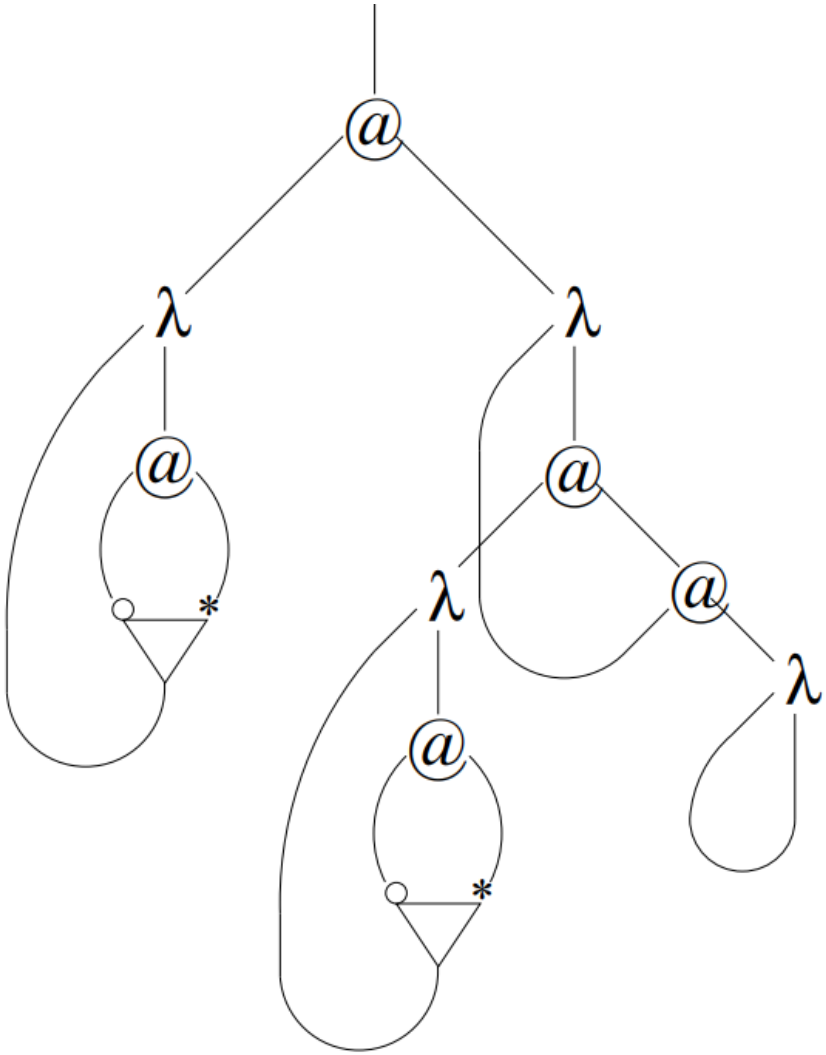
- First proposed by Lamping in 1989: *an implementation based on graph reduction of Lévy's optimal parallel β-reduction*
- One explicit node for *sharing* (fan-in) and *unsharing* (fan-out)
- (Note: no operational distinction between fan-in and fan-out)
- The core idea: *progressively and lazily unshare* parts of the term
- Additionally, variables are connected to their binders on the left:



Sharing graph for $\Delta \triangleq \lambda x.x\ x$

Sharing graph for $I \triangleq \lambda x.x$

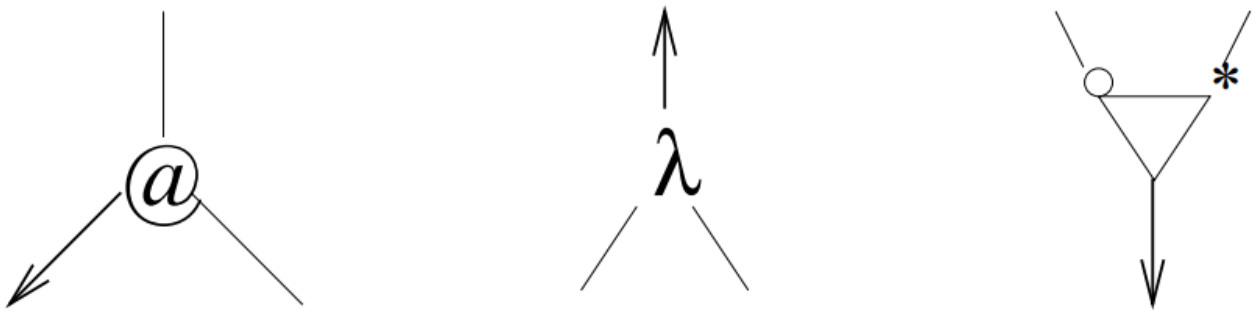
Sharing Graphs – Example



Sharing graph for $M \triangleq \Delta\ (\lambda f.\Delta(f\ I))$

Sharing Graphs – Reduction Rules

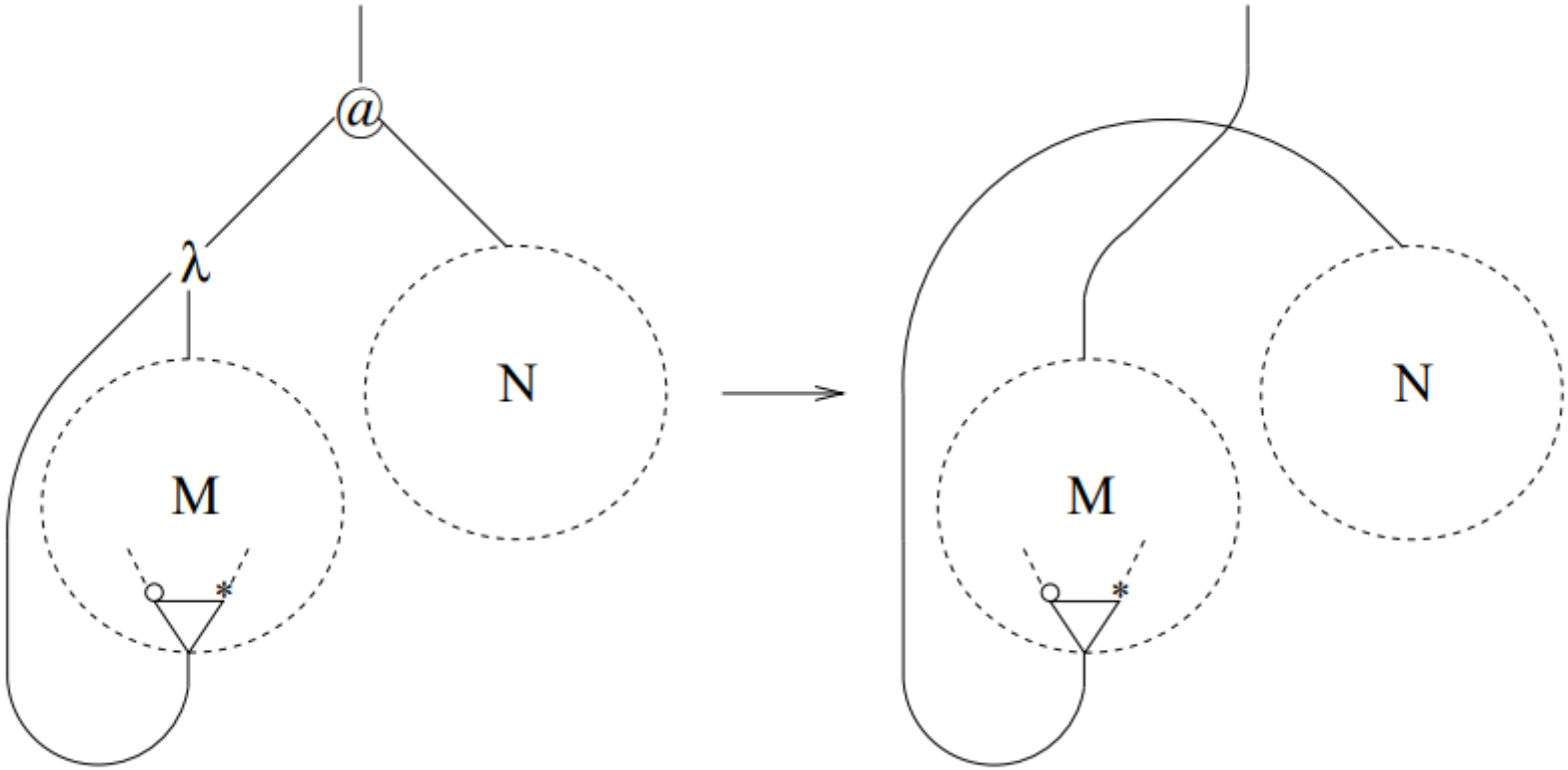
- We have a representation for λ-terms with explicit sharing, but:
- *How is β-reduction implemented?*
- *How do the nodes interact with each other and "reduce the graph"?*
- Each node has one **principal port** and zero or many **secondary ports**:



Principal ports of the three nodes

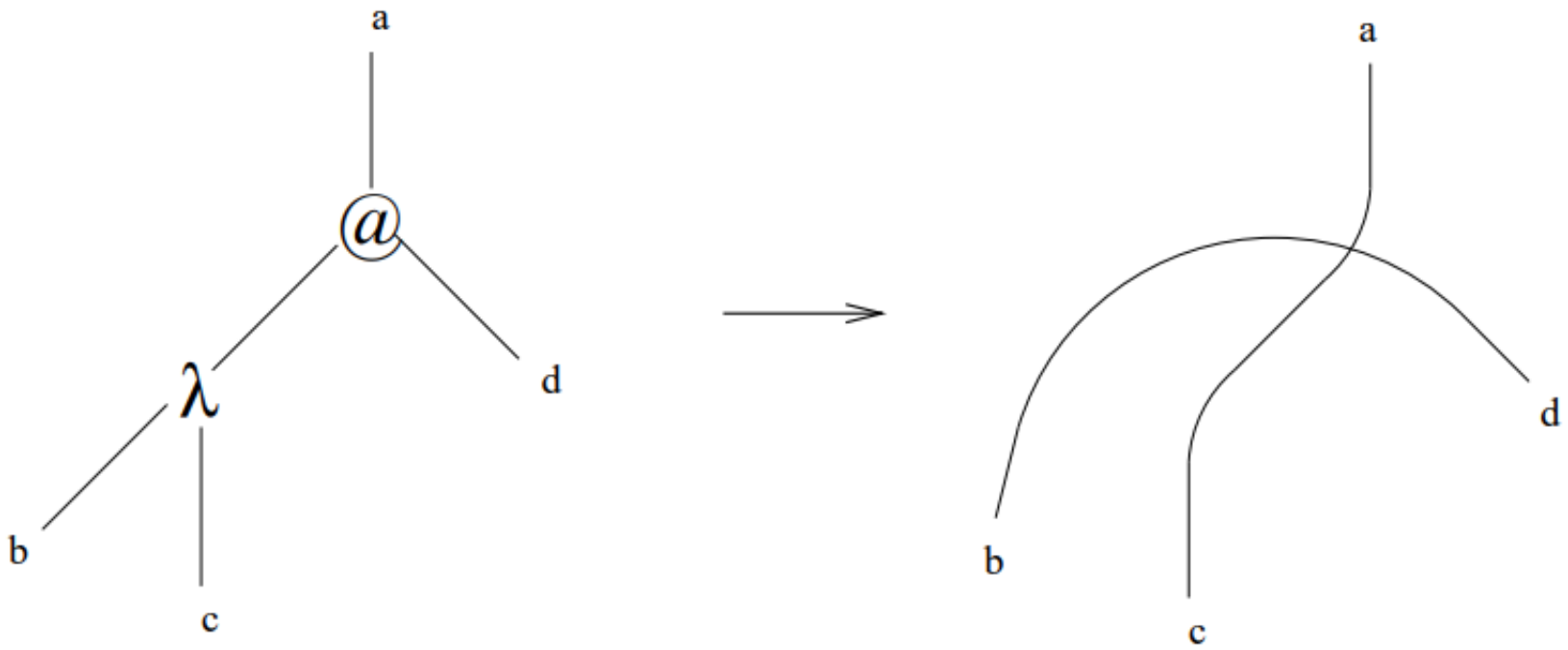
- *Reductions occur when the **principal ports** of two nodes are connected with each other (interaction between nodes)*
- A graph rewriting system introduced by Lafont in 1990: *Interaction Nets*

Sharing Graphs – Reduction Rules (β)



β-reduction in sharing graphs (λ-@ interaction)

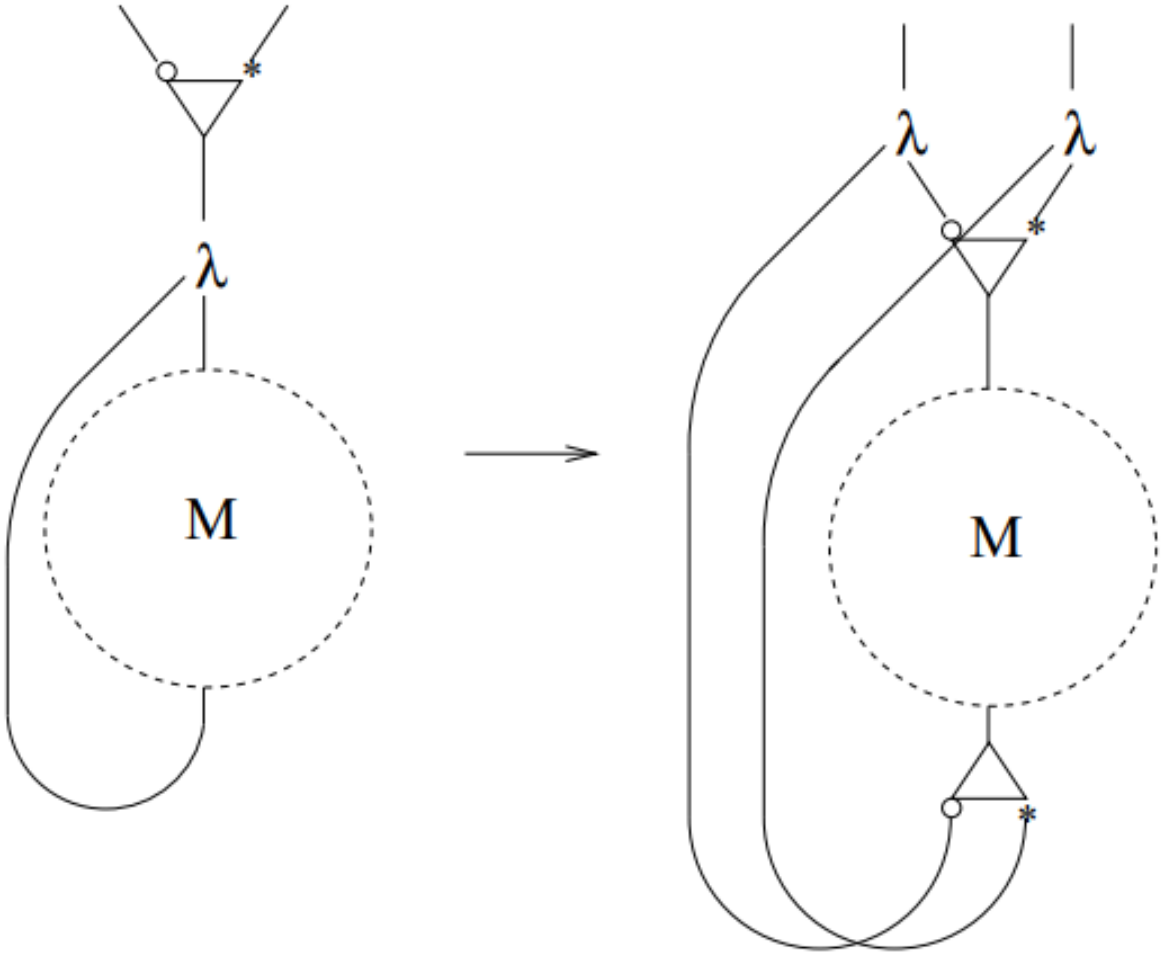
Sharing Graphs – Reduction Rules (β)



β-reduction in sharing graphs, locally (λ-@ interaction)

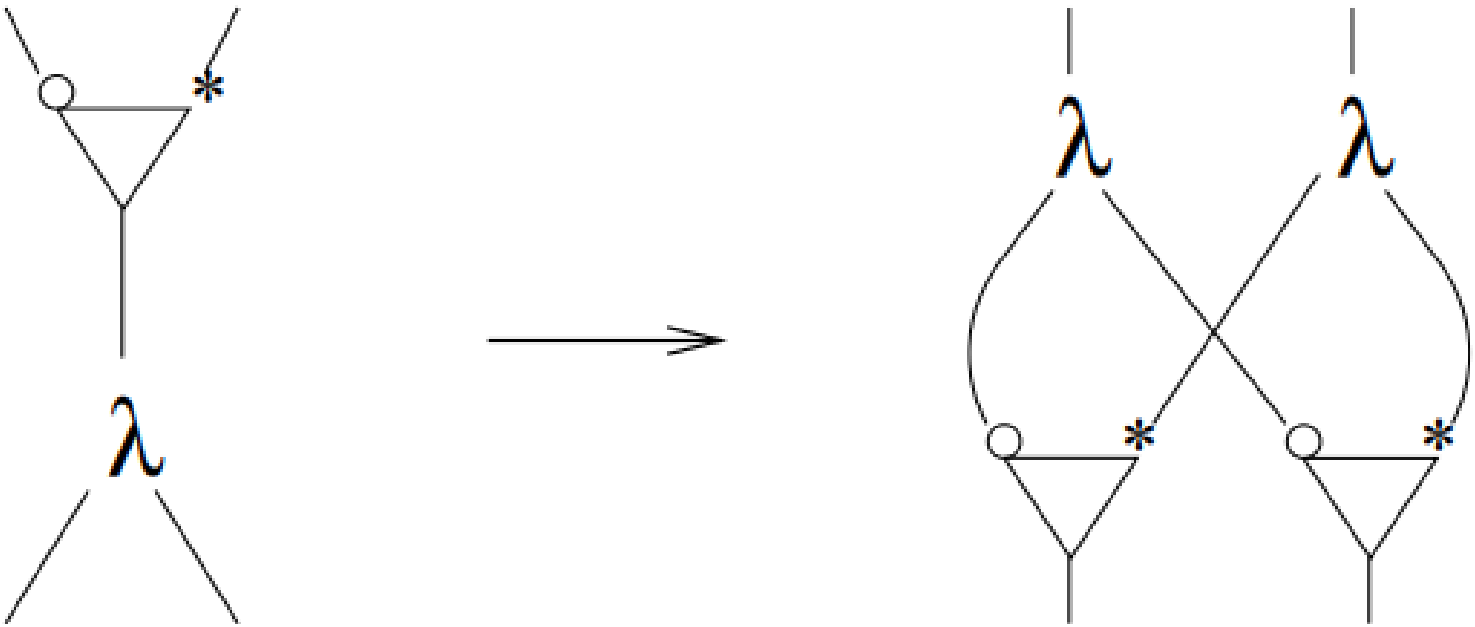
- *Reductions are local*: only two nodes interact, the graph is unaltered
- This allows for *highly-parallel implementations* of graph reduction
- The reduction can be performed in $O(1)$ time by a simple rewiring

Sharing Graphs – Reduction Rules (λ-fan)



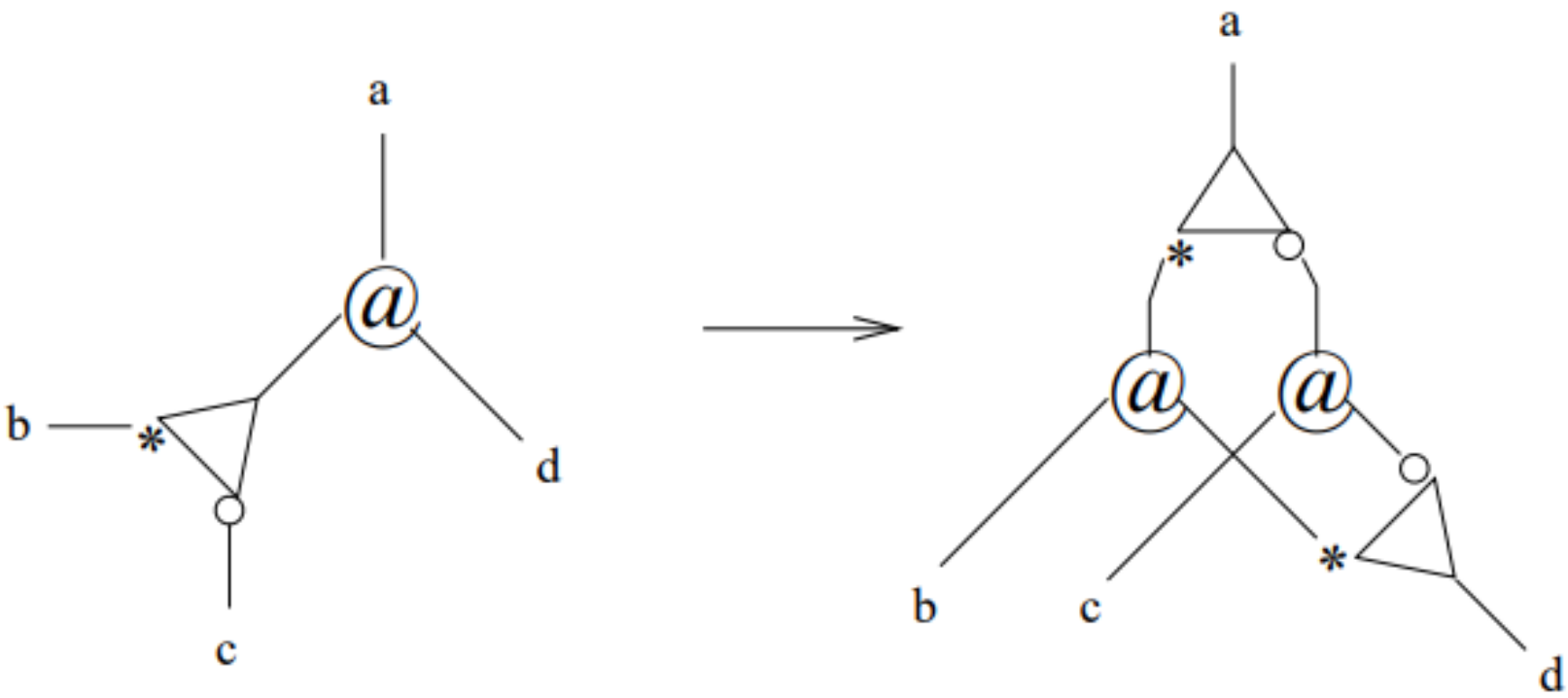
Unsharing of λ-abstractions
(note the matching fan-out at the end)

Sharing Graphs – Reduction Rules (λ-fan)



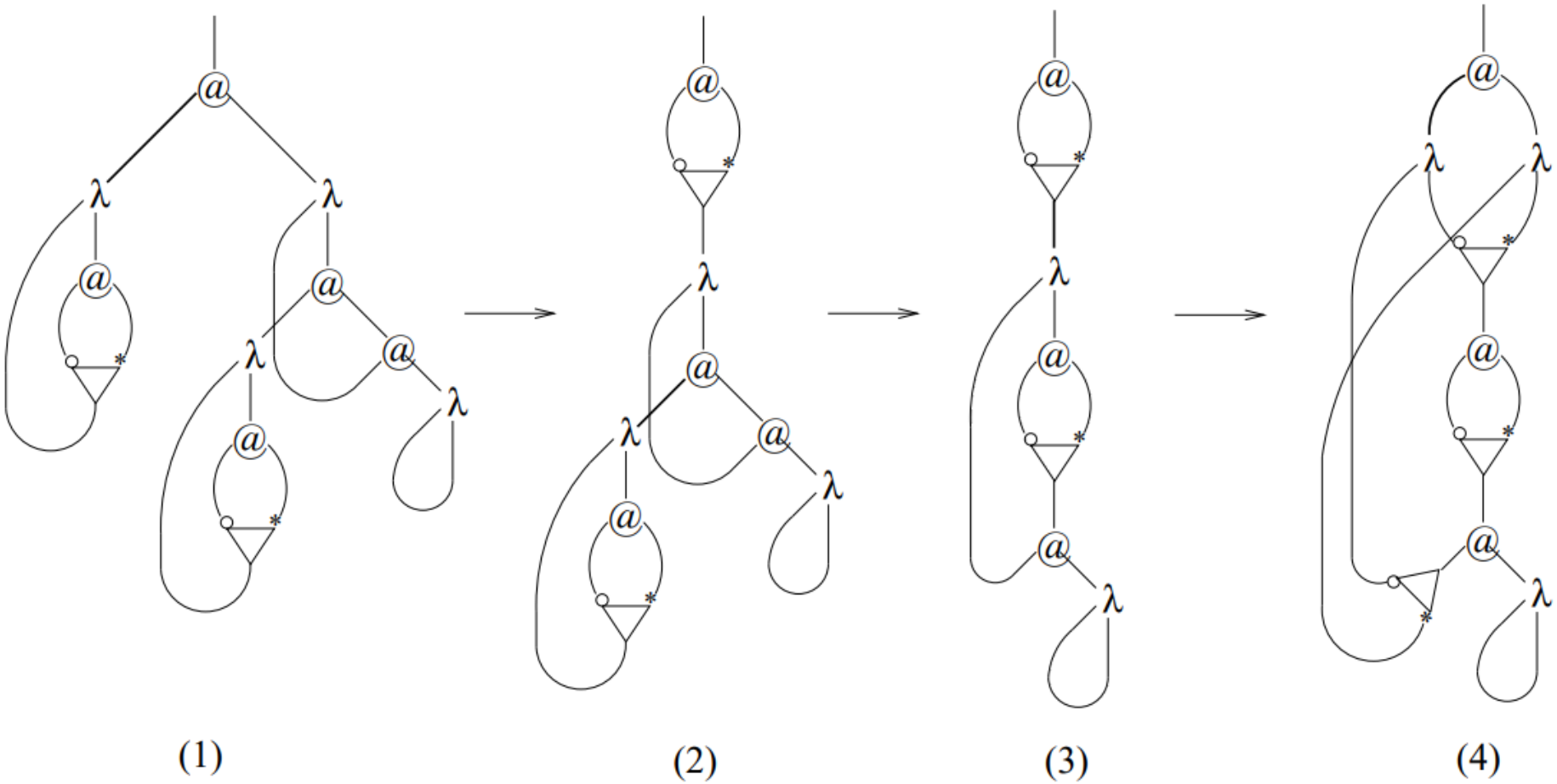
Unsharing of λ-abstractions, locally (λ-fan interaction)

Sharing Graphs – Reduction Rules (@-fan)



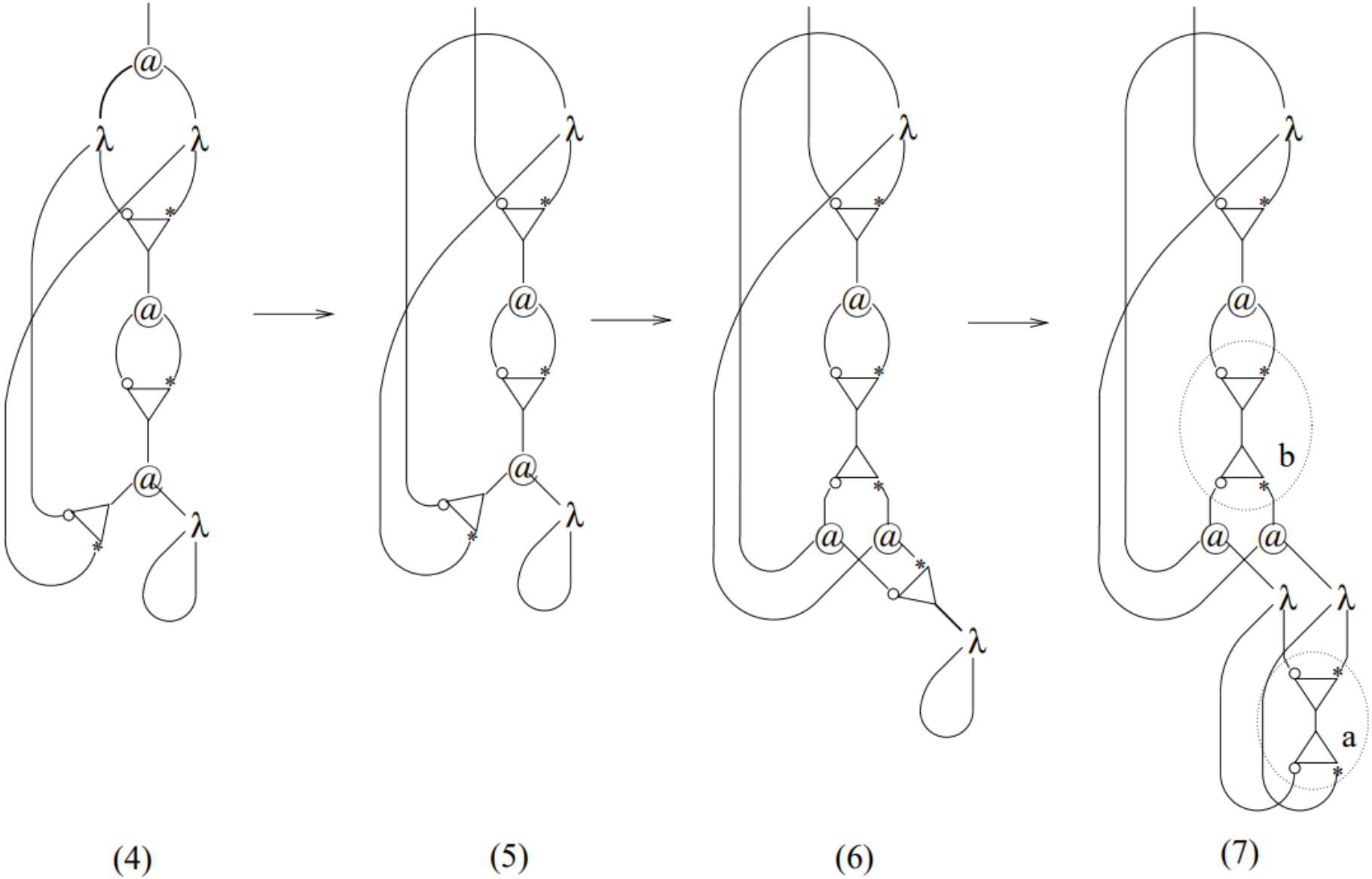
Unsharing of applications (@-fan interaction)

Sharing Graphs – Reduction Example (1-4)



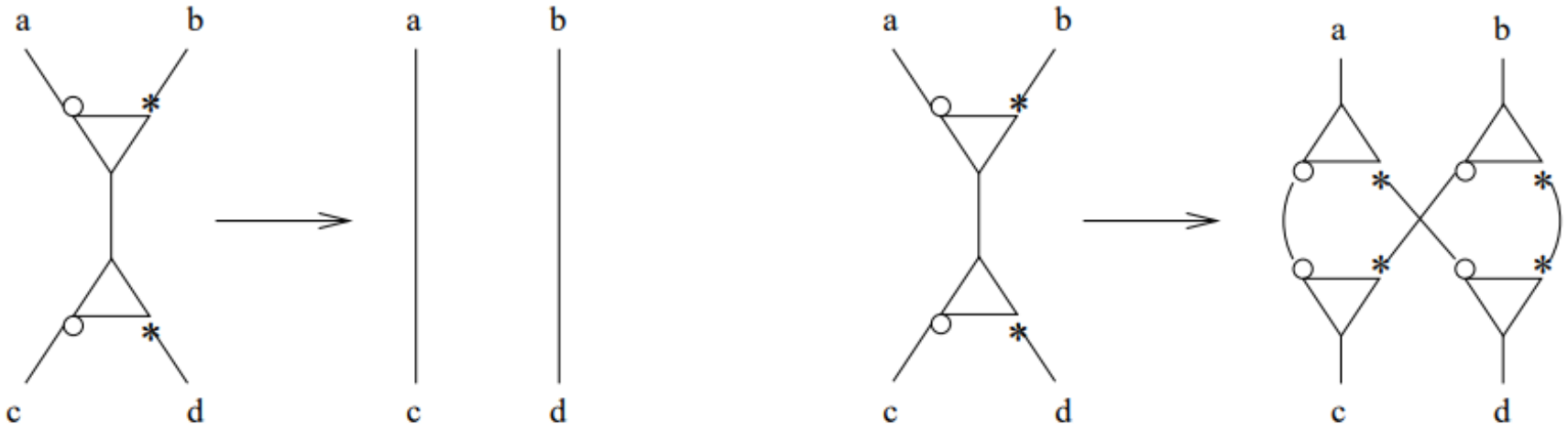
Sharing graph reduction for $M \triangleq \Delta\ (\lambda f.\Delta(f\ I))$ (eventually, $M \Longrightarrow_\beta^* I$)

Sharing Graphs – Reduction Example (4-7)



How should the fan pairs a and b interact?

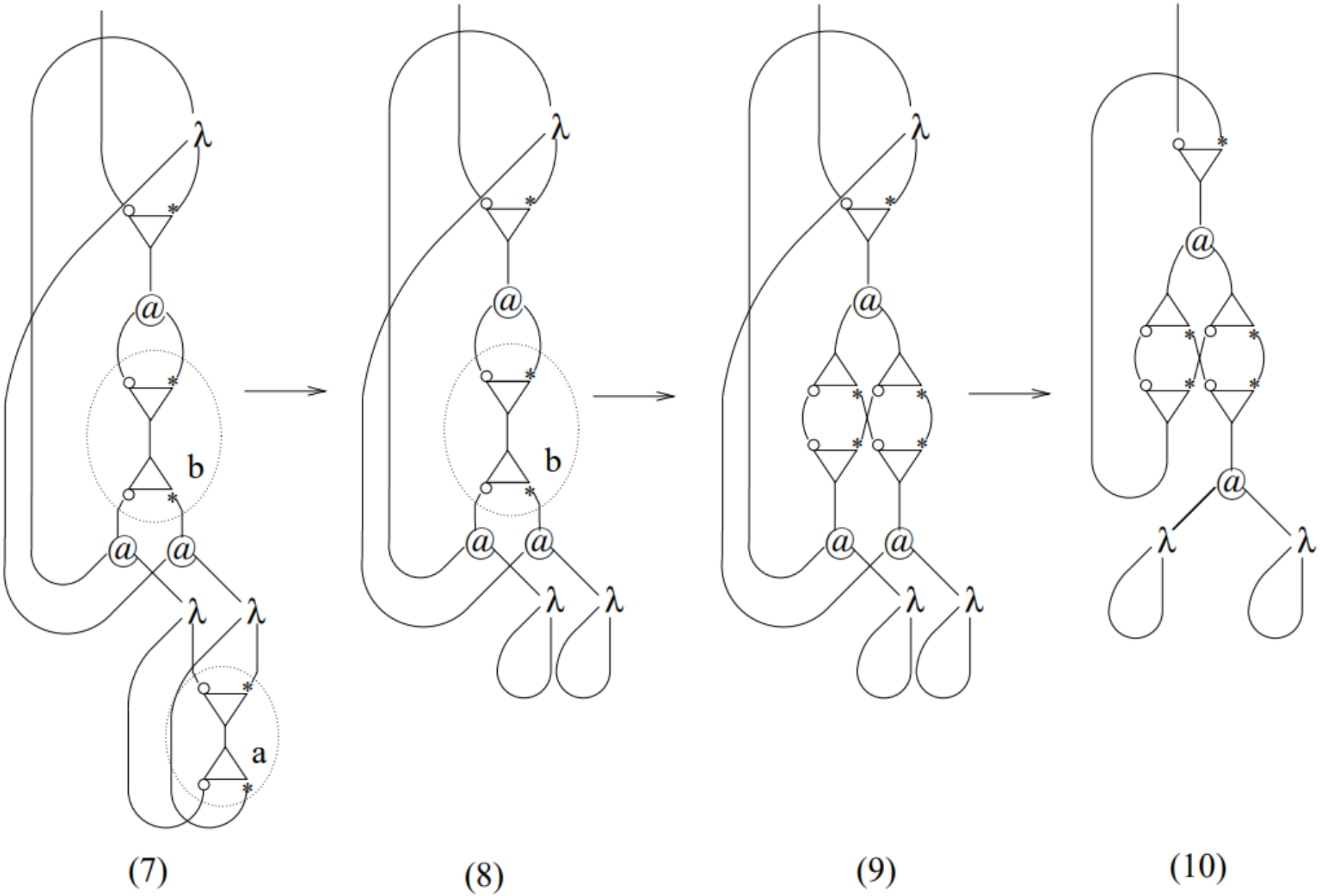
Sharing Graphs – Reduction Rules (fan-fan)



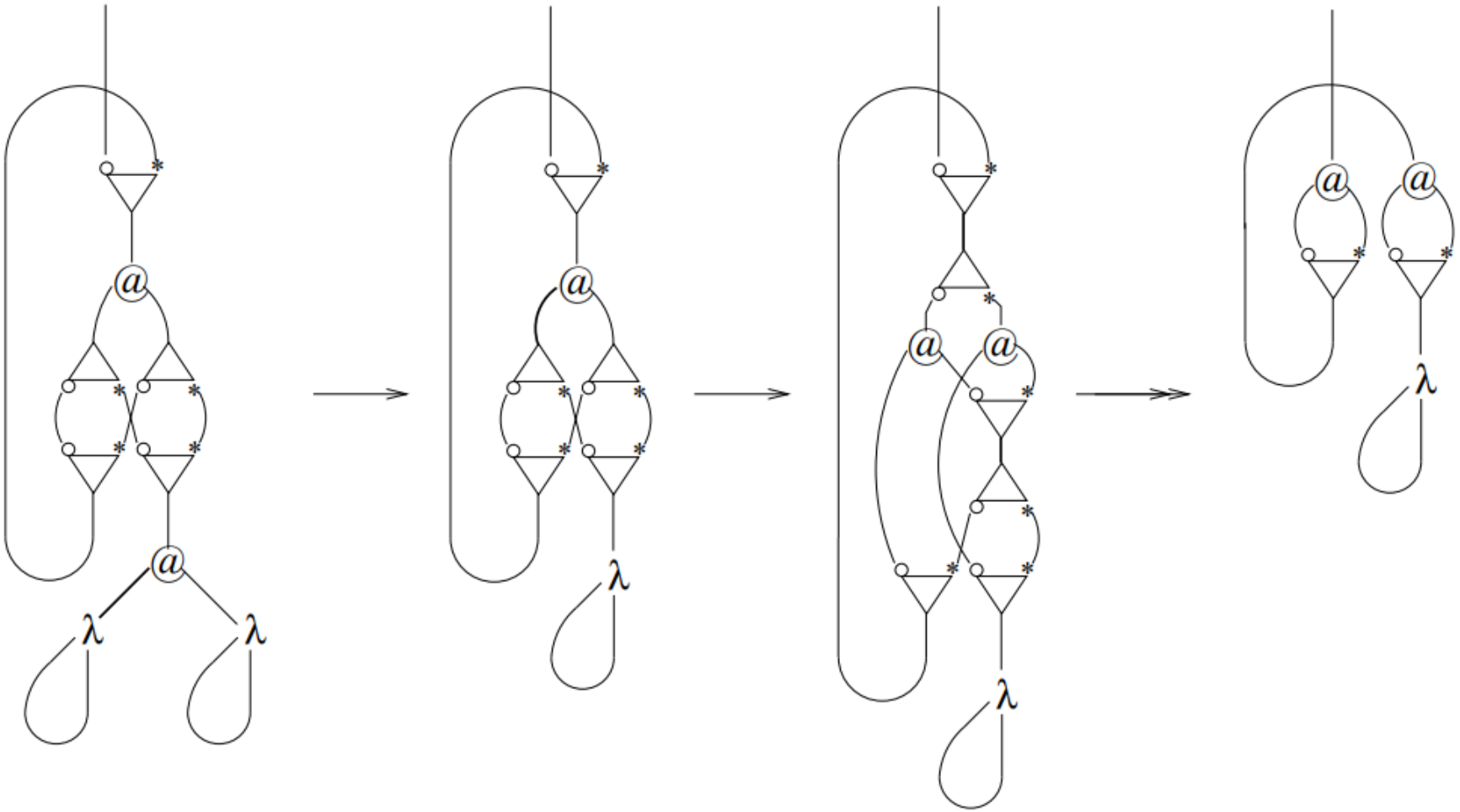
Annihilation and duplication rules for fans (fan-fan interaction)

- It is *non-trivial to determine* which rule should be applied
- Intuition: if the two fans were originated from the same "duplication process", they *annihilate* and cancel each other out (sharing $\rightarrow$ unsharing)
- Otherwise, *exchange* them by duplicating the two fans

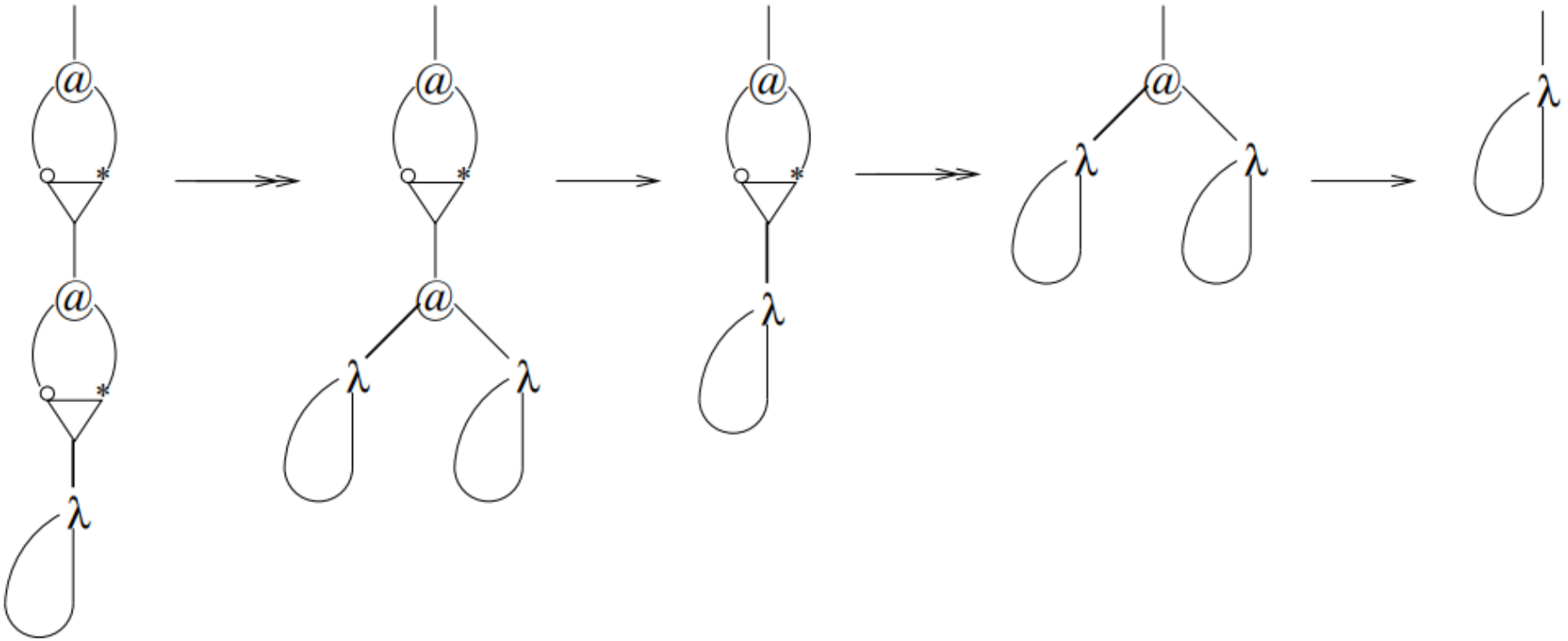
Sharing Graphs – Reduction Example (7-10)



Sharing Graphs – Reduction Example (10-14)



Sharing Graphs – Reduction Example (14-20)



(14) (15-16) (17) (18-19) (20)

No more rules can be applied, we reached the normal form $I \triangleq \lambda x.x$

Sharing Graphs – Bookkeeping

- In practice, how is the pairing of fans determined?
- Presented so far: **abstract algorithm = β-reduction + duplication**
- The additional structure required to determine the correct pairing of duplication fans is called **bookkeeping** [Lamping 1990]
- The structure captured by fans is actually quite complex:

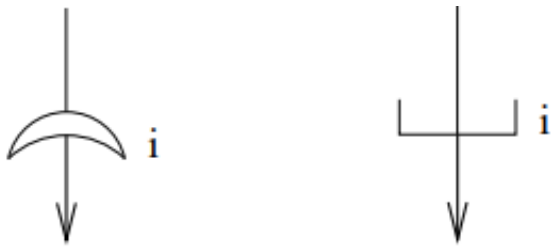
  Would a simple unique labeling work? **No**

  Would giving a level number to fans work? **Almost, but no**

- The main presentations in the literature:
  - Lamping (1990): *"An Algorithm for Optimal Lambda-Calculus Reduction"*
  - Gonthier, Abadi, Lévy (1992): *"The Geometry of Optimal Lambda-Reduction"*
  - Asperti (1994): *"Linear Logic, Comonads and Optimal Reductions"*

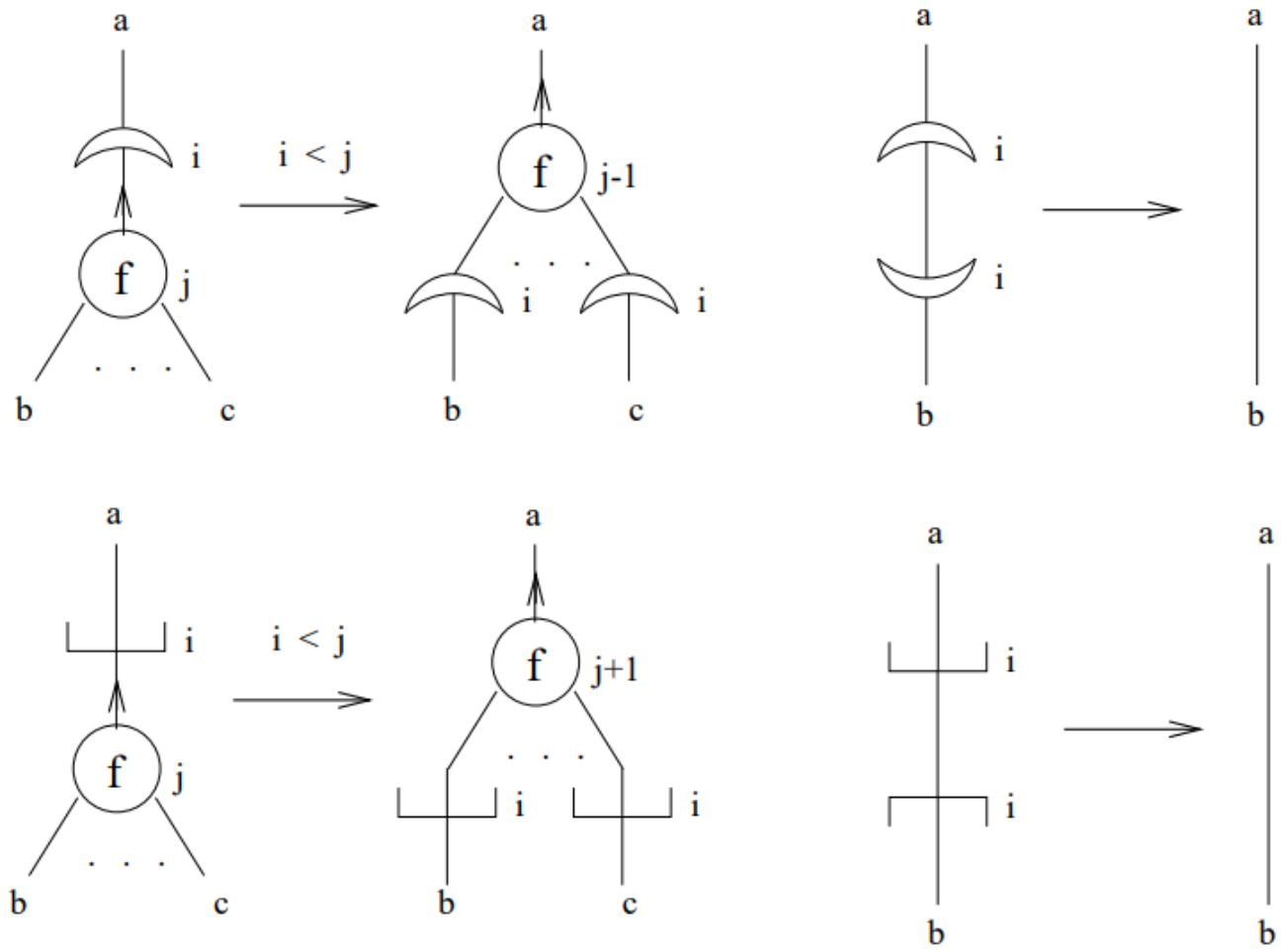
Sharing Graphs – Bookkeeping

- Each node is given a *level*, which can change during the computation
- If two fans pairs are at the same level, they *annihilate*, otherwise they *exchange* as before
- Interaction between two nodes now occurs when both: the principal ports are connected **and** *their level is the same*
- Non-trivial idea: *a notion of enclosure to delimit the interaction of fans*
- The level of a node can be *dynamically* incremented or decremented by two new nodes: **open brackets** (croissants) and **close brackets** (square brackets), respectively



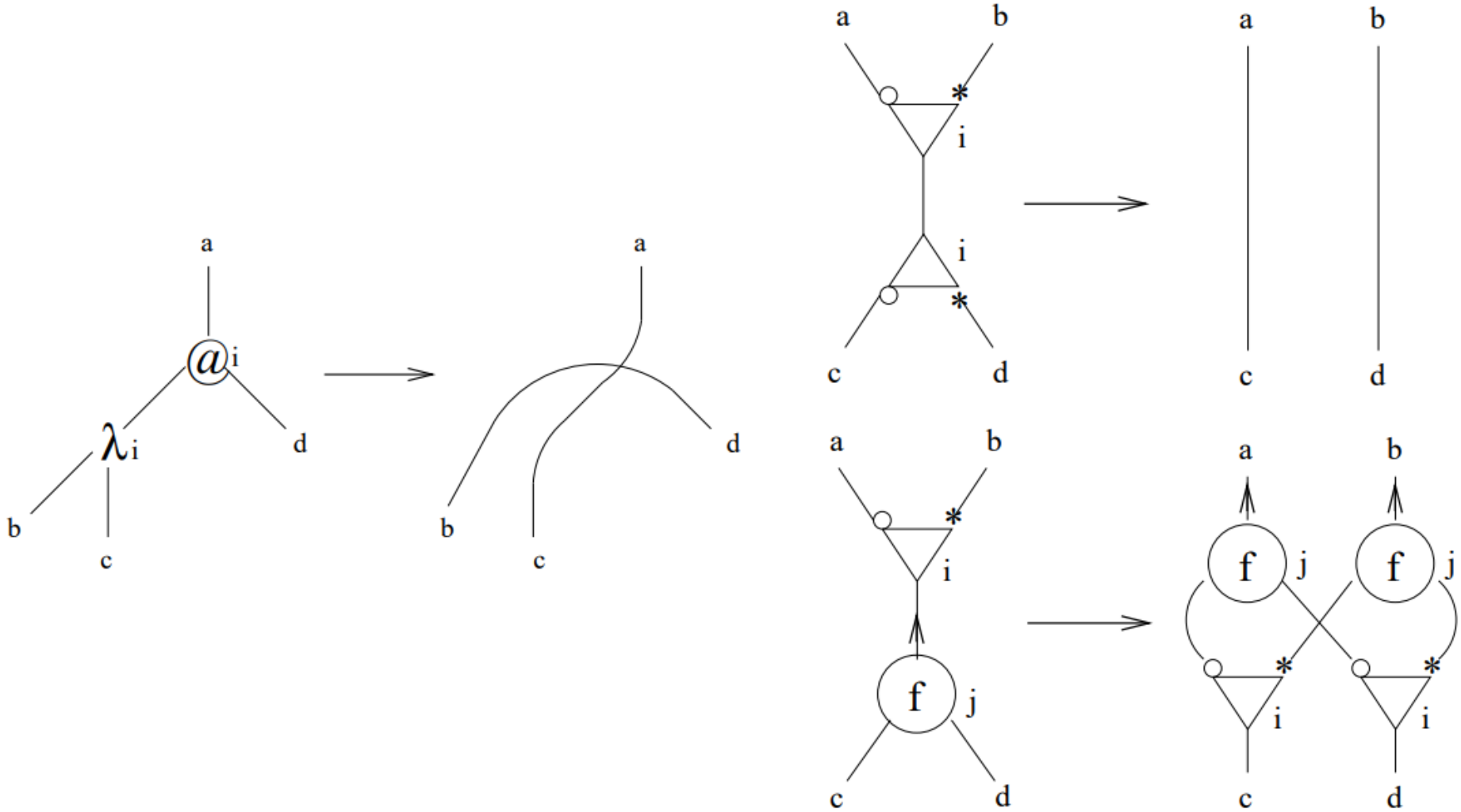
Bookkeeping nodes: open bracket and close bracket

Sharing Graphs – Bookkeeping Rules



Bookkeeping interactions and annihilations

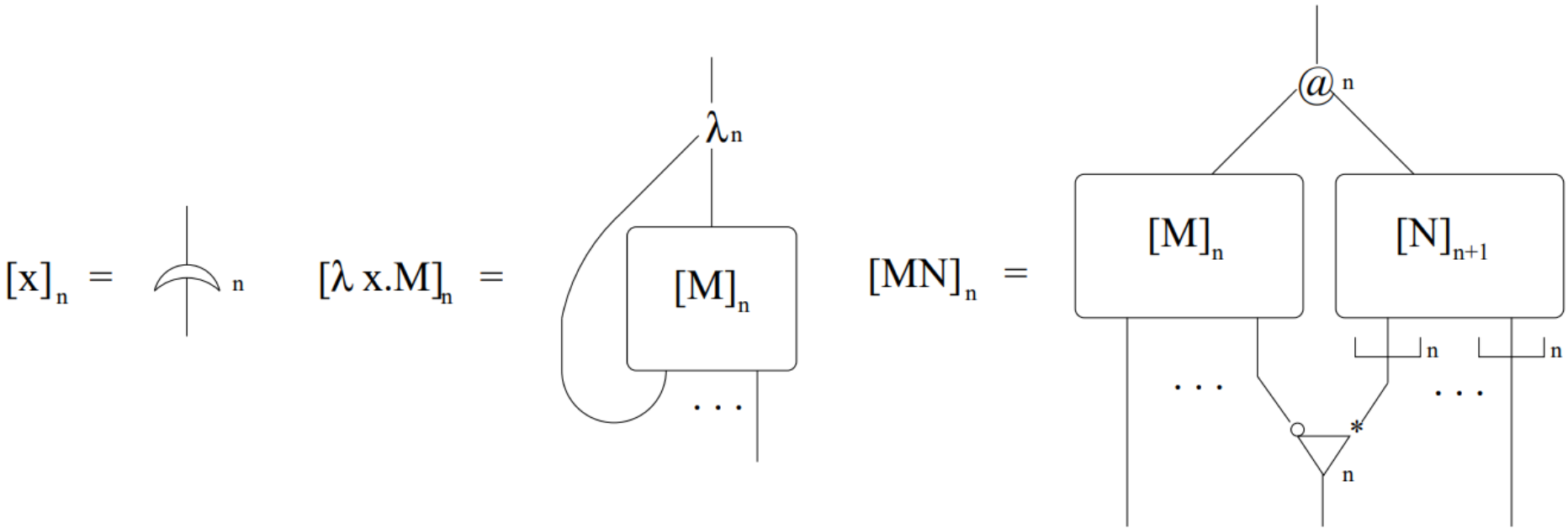
Sharing Graphs – Reduction Rules with Levels



β-rule, fan annihilation, fan duplication

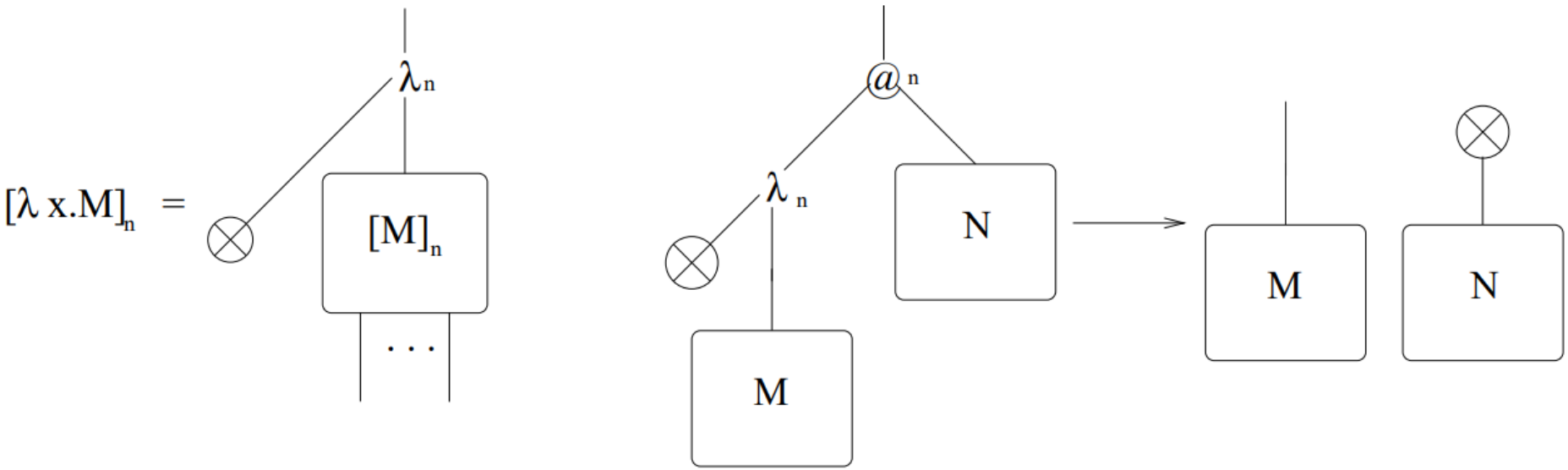
Sharing Graphs – Initial Encoding

- Final issue: how do we assign levels to λ-term nodes?
- Encoding defined by structural induction, with $n = 0$ at the start



- The graph has m free edges at the bottom, one for each free variable
- The sharing level of a term is **increased** when it is an *argument of an application*, and **decreased** when accessed by a variable in a function
- Note: in the application, fans are required for each *shared* free variable

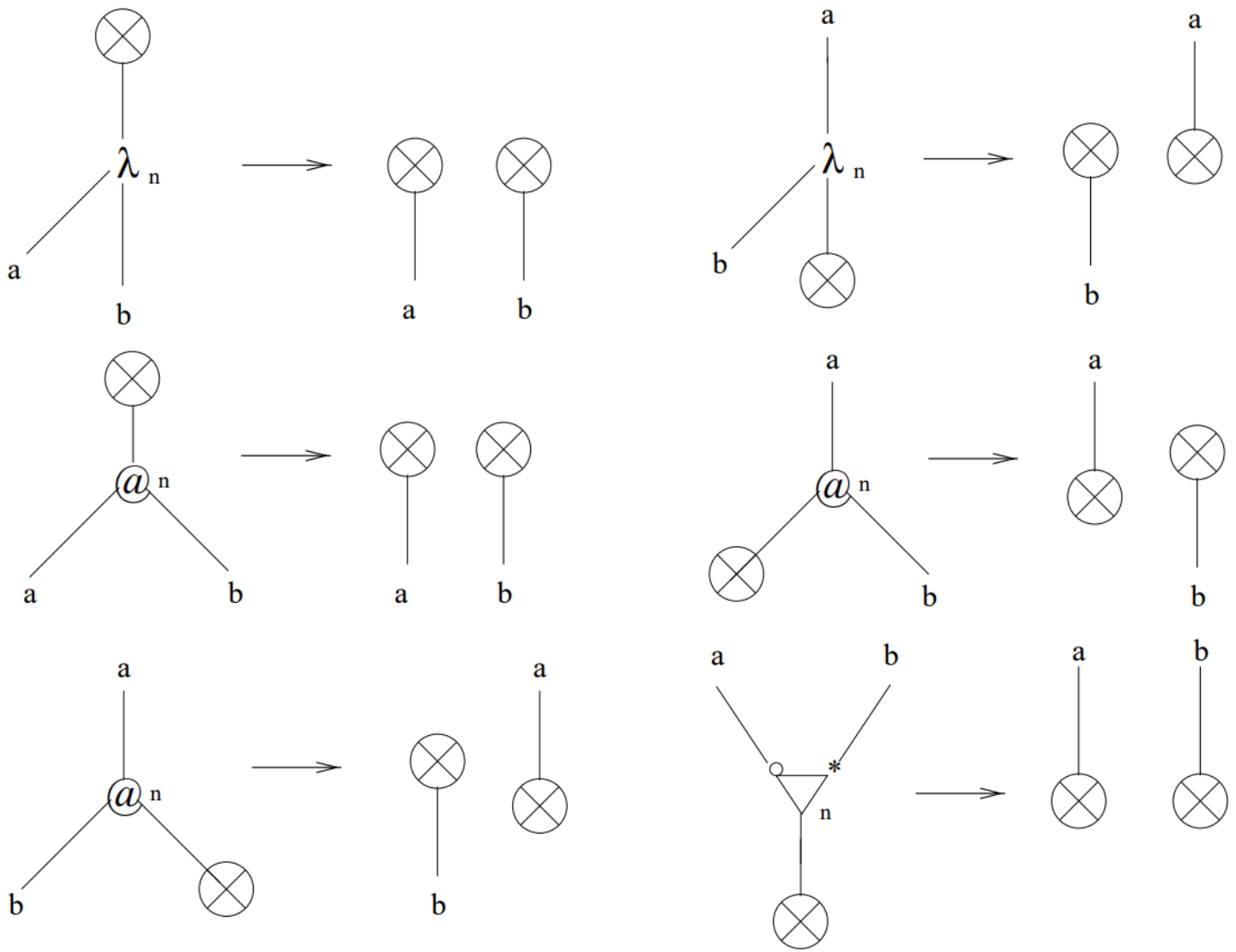
Sharing Graphs – Initial Encoding with Garbage nodes



Additional encoding rule with garbage nodes when $x \notin \mathrm{FV}(M)$

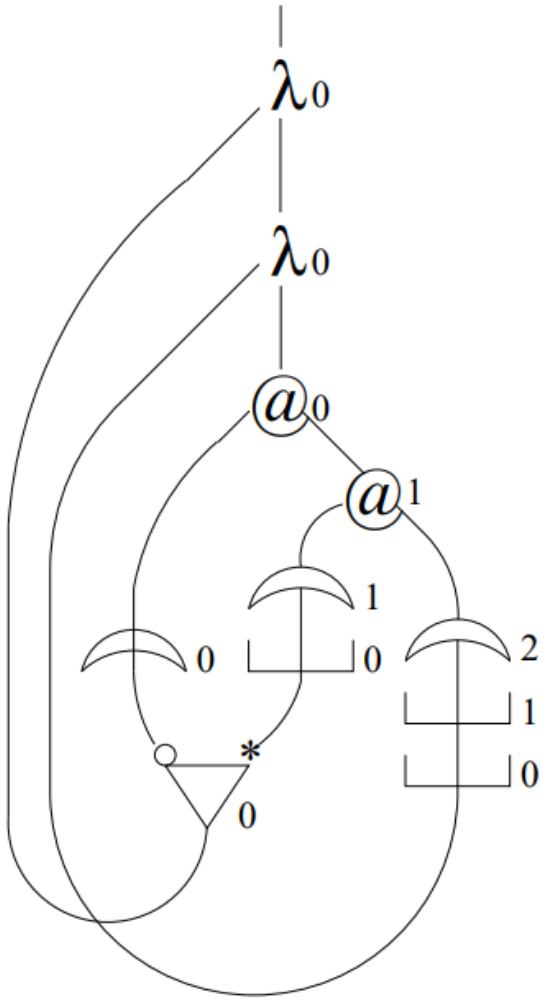
In the β*-reduction, the argument* N *ends up "discarded"*

Sharing Graphs – Garbage Collection Rules



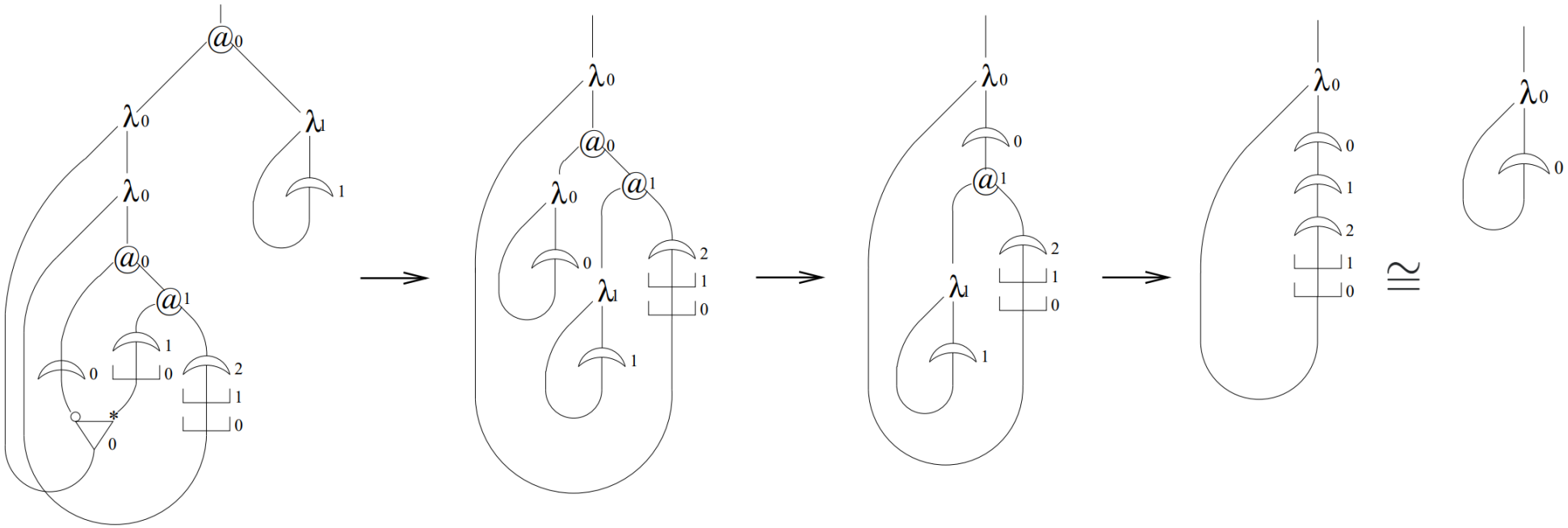
Garbage collection reduction rules

Sharing Graphs – Example with Levels



Sharing graph for $\overline{two} \triangleq \lambda f.\lambda x.f\ (f\ x)$

Sharing Graphs – Reduction with Levels



Reduction for $\overline{two}\ I$, isomorphic to the identity

Complexity results of Optimal Reduction

Complexity Results – Asperti and Mairson (1998)

- *How costly is it to reduce a redex family inside a λ-term?*
- Is there a bound on the amount of graph reductions done in terms of the parallel β-reduction steps (i.e.: redex families) of a λ-term?

Theorem (Parallel β-reduction is not Kalmár-elementary recursive) [Asperti and Mairson 1998]

The time complexity of n steps of parallel β-reduction steps (reducing n redex families) is not bounded by $O(2^n)$, nor $O(2^{2^n})$, nor $O(2^{2^{2^n}})$, etc.

- This lower bound applies to *any* technology that can implement Lévy-optimal reduction.
- *The number of redex families is not a good cost indicator of a λ-term*
- Previous results by [Lawall, Mairson, 1996] and [Asperti, 1996] gave $O(2^n)$

Complexity Results – The Proof Ideas

Statman's Theorem [Statman 1979]

The time required to normalize *simply-typed λ-terms* is not bounded by any elementary recursive function in the size of the term.

Simply-typed λ-terms are linear in redex families [Asperti, Mairson 1998]

Any *simply-typed λ-term* can be normalized in a number of parallel β-reduction steps *linear* in the size of the term.

- The number of parallel β-reduction steps required is surprisingly *low*
- Hence, most part of the work (which has to be done by Statman's theorem) must be devolved to *duplication and bookkeeping*
- The complexity of a term is hardly related to the number of required parallel β-steps, but to the *sharing machinery needed to implement them*
- (Statman's theorem is re-derived using the techniques of [Mairson 1992], which uses λ-calculus terms to implement *quantifier elimination for higher-order logic over a finite base type*)

Complexity Results – Asperti, Coppola, Martini (2004)

- Which is to blame? The *abstract algorithm* or the *bookkeeping*?
- Surprisingly, the issue is *not the bookkeeping*, but the **intrinsic duplication** and sharing of λ-terms:

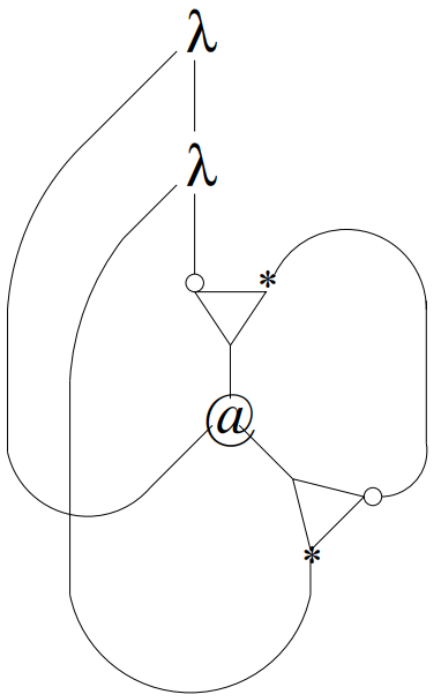
Theorem (Duplications are not Kalmár-elementary recursive) [Asperti, Coppola, Martini 2004]

There exist λ-terms normalizable in n steps of parallel β-reduction with number of *duplication interactions* not bounded by $O(2^n)$, nor $O(2^{2^n})$, etc.

- Main technical tool: revisiting the proof terms used in [Asperti, Mairson 1998] by typing them in *Elementary Affine Logic (EAL)*
- If a term can be given a type in EAL (for which they give a type-inference algorithm), *bookkeeping is not necessary* and a simple labelling of fan nodes suffices.
- *λ-terms have an inherent sharing, which can explode in size even with an implementation of optimal duplication.*

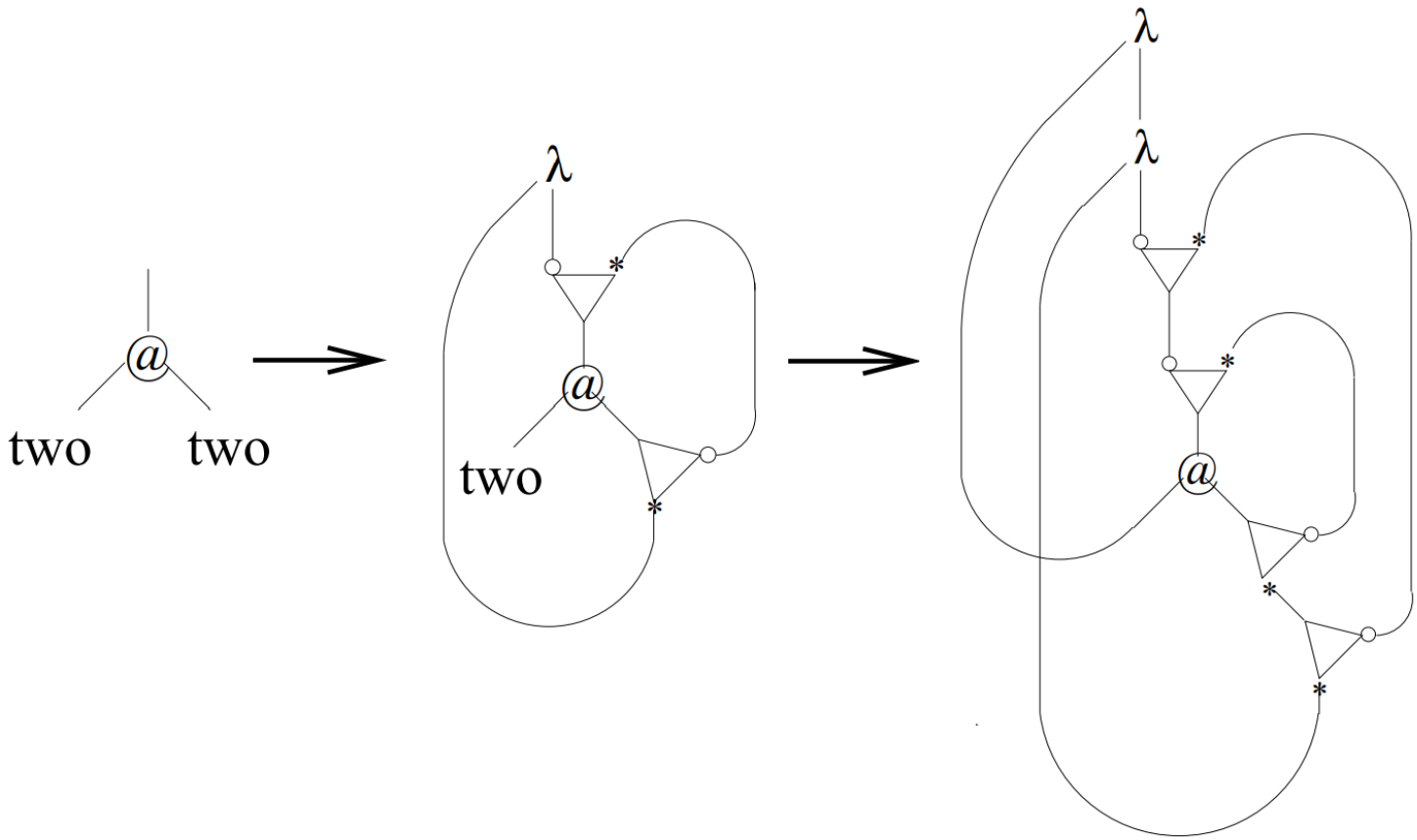
An Example of Higher-Order Sharing

- Sharing graphs take sharing to the extreme, sharing not only the basic structures of λ-calculus, but *sharing nodes themselves*
- This allows for exponential terms to be coded in *linear space* graphs with implicit *sharing nets*:



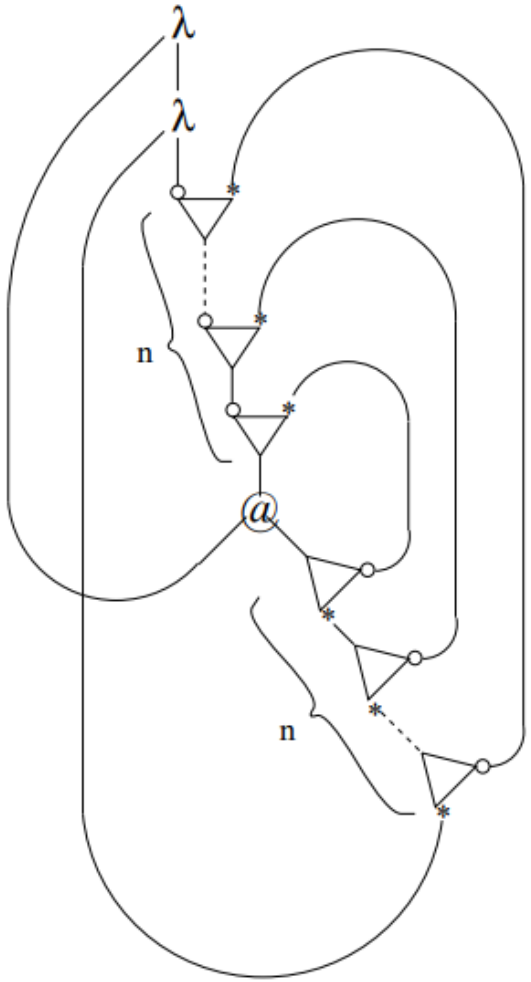
An isomorphic sharing graph representation for $\overline{two} \triangleq \lambda f.\lambda x.f\ (f\ x))$

An Example of Higher-Order Sharing



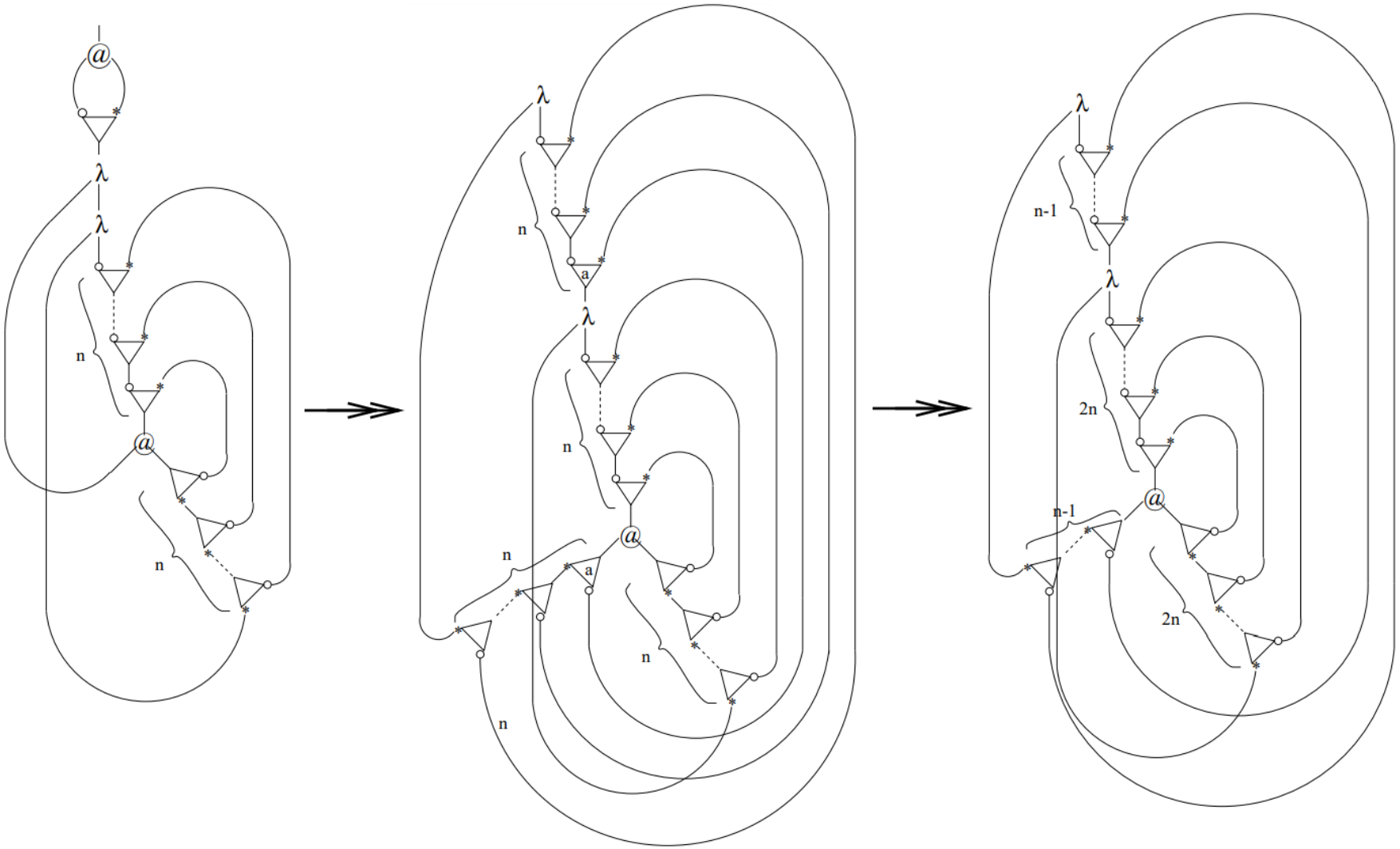
Sharing graph representation for $\overline{two}\ \overline{two}$

An Example of Higher-Order Sharing



Sharing graph representation for $\overline{n}\ \overline{two} \cong 2^n$

An Example of Higher-Order Sharing



Sharing graph representation for 2^n 2^n (in n family reductions)

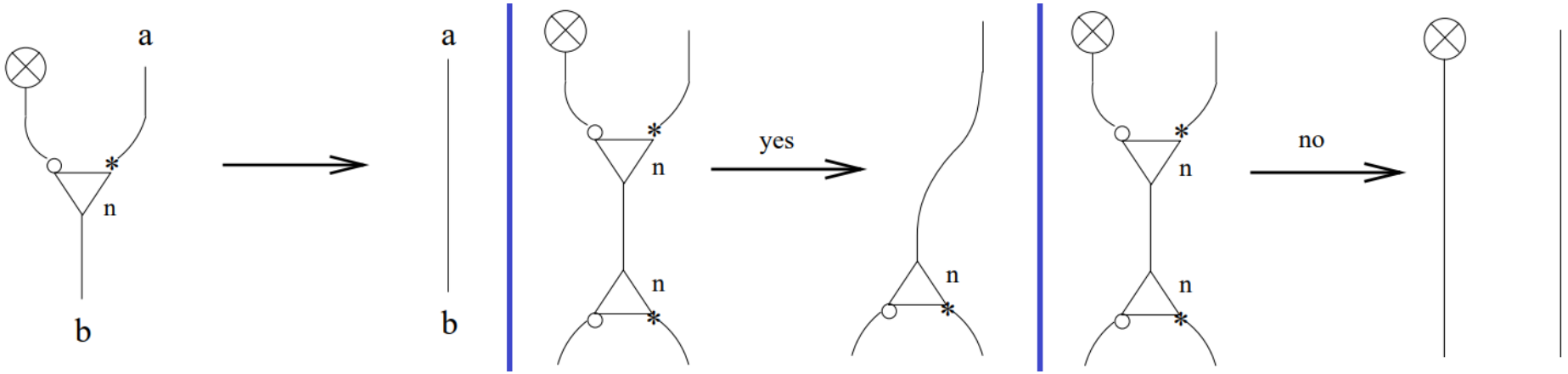
Practical aspects of Optimal Reduction

The Bologna Higher-Order Machine (BOHM)

- An efficient implementation of a (lazy) functional programming language using sharing graphs, written in C [Asperti, Giovannetti, Naletto 1996]
- The source language is a sugared λ-calculus with integer, lists, booleans
- **For pure λ-calculus terms**, BOHM greatly *outperforms* the competition (SML, Caml Light, Yale Haskell), giving polynomial reductions for many exponential cases
- **For real world functional programs**, BOHM is *one order of magnitude slower* than call-by-value languages (SML, Caml Light), and sometimes only slightly worse than Yale Haskell
- The fundamental issue: real world programming rarely uses *higher-order functionals* and *functions-as-data* (usually just for parametricity)
- However, higher-order programming is *precisely* what optimal reduction exploits, by taking advantage of the *inherent sharing* of λ-terms

The Bologna Higher-Order Machine (BOHM)

- On *total* absence of sharing: BOHM 1.0 is 10x slower on λ-terms, 50x slower on numerical computations compared with Caml Light
- Experimentally, the slowdown compared to Caml Light is *constant*, but on many cases the speedup is *exponential*
- *Essential* optimization: garbage collection *must* be performed *frequently* because garbage can be reduced and *duplicated* along the graph
- In many cases, *exponential* reduction times can be reduced to *linear*
- BOHM 1.1: fans that can never pair with other fans (*safe*) are allowed to be collected earlier (e.g.: in the initial graph all operators are safe)



Critical pair fan-garbage

BOHM – Benchmarks (Factorial and Fibonacci)

```
def I     = \x.x;;
def zero  = \x.\y.y;;
def one   = \x.\y.(x y);;
def two   = \x.\y.(x (x y));;
def three = \x.\y.(x (x (x y)));;
...
def Pair = \x.\y.\z.(z x y);;
def Fst  = \x.\y.x;;
def Snd  = \x.\y.y;;
def Succ = \n.\x.\y.(x (n x y));;
def Add  = \n.\m.\x.\y.(n x (m x y));;
def Mult = \n.\m.\x.(n (m x));;
```

```
def nextfact =
  \p.let n1 = (p Fst) in
     let n2 = (Succ (p Snd)) in
       (Pair (Mult n1 n2) n2);;
def nextfibo =
  \p.let n1 = (p Fst) in
     let n2 = (p Snd) in
       (Pair (Add n1 n2) n1);;
def fact = \n.
  (n nextfact (Pair one zero) Fst);;
def fibo = \n.
  (n nextfibo (Pair zero one) Fst);;
```

Factorial and Fibonacci implemented with Church numerals.

BOHM – Benchmarks (fact)

Input	BOHM 0.0	BOHM 1.1	Caml Light	Haskell
(fact one I I)	0.00 s. 561 interactions 71 fan int. 20 family red.	0.00 s. 133 interactions 16 fan int. 30 family red.	0.00 s.	0.00 s.
(fact three I I)	0.04 s. 3375 interactions 251 fan int. 45 family red.	0.00 s. 386 interactions 154 fan int. 55 family red.	0.00 s.	0.01 s.
(fact five I I)	0.23 s. 17268 interactions 412 fan int. 74 family red.	0.00 s. 922 interactions 341 fan int. 84 family red.	0.02 s.	0.05 s.
(fact seven I I)	explodes	0.02 s. 1952 interactions 608 fan int. 117 family red.	0.17 s.	0.84 s.
(fact nine I I)		0.03 s. 3820 interactions 979 fan int. 154 family red.	10.60 s.	explodes
(fact ten I I)		0.05 s. 5202 interactions 1211 fan int. 174 family red.	explodes	
(fact twenty I I)		0.33 s. 53605 interactions 5983 fan int. 435 family red.		

BOHM – Benchmarks (fibo)

Input	BOHM 0.0	BOHM 1.1	Caml Light	Haskell
(fibo one I I)	0.01 s. 513 interactions 72 fan int. 20 family red.	0.00 s. 121 interactions 14 fan int. 28 family red.	0.00 s.	0.00 s.
(fibo four I I)	0.06 s. 3152 interactions 244 fan int. 55 family red.	0.00 s. 462 interactions 174 fan int. 63 family red.	0.00 s.	0.01 s.
(fibo seven I I)	0.19 s. 12749 interactions 513an int. 98 family red.	0.00s. 1260 interactions 440 fan int. 106 family red.	0.01 s.	0.01 s.
(fibo ten I I)	0.74 s. 52654 interactions 1260 fan int. 173 family red.	0.03 s. 3898 interactions 1178 fan int. 181 family red.	0.02 s.	0.04 s.
(fibo thirteen I I)	3.08 s. 230548 interactions 4023 fan int. 390 family red.	0.12 s. 14357 interactions 3916 fan int. 398 family red.	0.04 s.	0.15 s.
(fibo sixteen I I)	explodes	0.38 s. 57844 interactions 15126 fan int. 1185 family red.	0.12 s.	0.85 s.
(fibo nineteen I I)		1.57 s. 241291 interactions 62224 fan int. 4412 family red.	0.44 s.	3.69 s.

BOHM – Benchmarks (Garbage Collection)

n	**Term**	**GC off**	**GC on**
1	fact one $I\ I$	1188	1178
2	fact two $I\ I$	1242	1219
3	fact three $I\ I$	1303	1276
5	fact five $I\ I$	1515	1383
10	fact ten $I\ I$	4544	1797
15	fact fifteen $I\ I$	20029	2452
20	fact twenty $I\ I$	73661	3226
1	fibo one $I\ I$	1220	1209
2	fibo two $I\ I$	1247	1234
3	fibo three $I\ I$	1296	1283
5	fibo five $I\ I$	1418	1381
10	fibo ten $I\ I$	2439	1763
15	fibo fifteen $I\ I$	26984	6228
20	fibo twenty $I\ I$	288003	57269

Maximum number of nodes allocated with and without GC.

BOHM – Benchmarks ($g \triangleq \lambda n.n\ \overline{two}\ I\ 0$)

Input	BOHM 0.0	BOHM 1.1	Caml Light	Haskell
(g one)	0.00 s. 124 interactions 23 fan int. 7 family red.	0.00 s. 30 interactions 4 fan int. 7 family red.	0.00 s.	0.00 s.
(g four)	0.00 s. 608 interactions 62 fan int. 16 family red.	0.00 s. 79 interactions 34 fan int. 16 family red.	0.00 s.	0.01 s.
(g seven)	0.04 s. 3052 interactions 101 fan int. 25 family red.	0.00 s. 136 interactions 64 fan int. 25 family red.	0.00 s.	0.09 s.
(g ten)	0.43 21176 interactions 140 fan int. 34 family red.	0.00 s. 193 interactions 94 fan int. 34 family red.	0.03 s.	0.70 s.
(g thirteen)	explodes	0.00 s. 280 interactions 126 fan int. 49 family red.	0.20 s.	6.12 s.
(g sixteen)		0.00 s. 346 interactions 156 fan int. 58 family red.	1.60 s.	explodes
(g nineteen)		0.00 s. 412 interactions 186 fan int. 67 family red.	12.65 s.	

BOHM – Benchmarks $(f \triangleq \lambda n.(n\ \overline{two})\ \overline{two}\ I\ 0)$

Input	BOHM 0.0	BOHM 1.1	Caml Light	Haskell
(f one)	0.00 s. 282 interactions 42 fan int. 12 family red.	0.00 s. 50 interactions 22 fan int. 12 family red.	0.00 s.	0.00 s.
(f two)	0.00 s. 765 interactions 67 fan int. 17 family red.	0.00 s. 94 interactions 44 fan int. 17 family red.	0.00 s.	0.02 s.
(f three)	0.04 s. 7001 interactions 100 fan int. 22 family red.	0.00 s. 170 interactions 74 fan int. 22 family red.	0.00 s.	0.18 s.
(f four)	explodes	0.00 s. 346 interactions 120 fan int. 27 family red.	1.02 s.	53.01 s.
(f five)		0.00 s. 866 interactions 198 fan int. 32 family red.	explodes	explodes
(f six)		0.00 s. 2650 interactions 340 fan int. 37 family red.		
(f ten)		3.58 s. 531706 interactions 4236 fan int. 57 family red.		

Conclusion and Related Work

Sharing graphs provide a theoretically-relevant and elegant model for the optimal and asymptotically more efficient (parallel) evaluation of λ-terms.

- Many other interesting connections:
  - the relation with linear logic [Gonthier, Abadi, Lévy 1997],
  - Lamping's paths, semantic equivalence [Asperti, Laneve 1997],
  - read-back, safe nodes, garbage collection [Asperti and Guerrini 1998],
  - interaction nets, algorithm optimizations [Mackie 2004],
  - parallel GPU-based implementations [Pedicini, Pellitta 2010]
- Other techiques for optimal reduction: Lambdascope [van Oostrom 2010]
- The cost of bookkeeping is still not yet exactly known [Asperti 2017]
- More recently: the literature is concentrating on abstract machines and their complexity [Accattoli, Dal Lago 2016], [Accattoli et al. 2019]

Thank you for your attention!

Bibliography I


Andrea Asperti and Harry G. Mairson.
Parallel beta reduction is not elementary recursive.
In *POPL '98, The 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 303–315. ACM, 1998.


Andrea Asperti, Paolo Coppola, and Simone Martini.
(optimal) duplication is not elementary recursive.
Inf. Comput., 193(1):21–56, 2004.


Andrea Asperti and Stefano Guerrini.
The optimal implementation of functional programming languages, volume 45 of *Cambridge tracts in theoretical computer science*.
Cambridge University Press, 1998.


Andrea Asperti, Cecilia Giovanetti, and Andrea Naletto.
The bologna optimal higher-order machine.
J. Funct. Program., 6(6):763–810, 1996.

Bibliography II


Andrea Asperti.
About the efficient reduction of lambda terms.
CoRR, abs/1701.04240, 2017.


Andrea Asperti.
On the complexity of beta-reduction.
In *POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 110–118. ACM, 1996.


Christopher P. Wadsworth.
Semantics and pragmatics of the lambda-calculus.
PhD thesis, Oxford, 1971.


Jean-Jacques Lévy.
Réductions correctes et optimales dans le lambda-calcul.
PhD thesis, January, 1978.

Bibliography III


Julia L. Lawall and Harry G. Mairson.
Optimality and inefficiency: What isn't a cost model of the lambda calculus?
In *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming, ICFP 1996*, pages 92–101. ACM, 1996.


Andrea Asperti and Cosimo Laneve.
Interaction systems I: the theory of optimal reductions.
Math. Struct. Comput. Sci., 4(4):457–504, 1994.

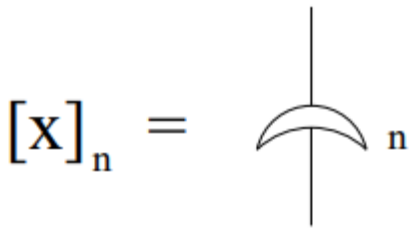

Andrea Asperti and Cosimo Laneve.
Interaction systems II: the practice of optimal reductions.
Theor. Comput. Sci., 159(2):191–244, 1996.

Appendix: Sharing Graphs – An Intuition for Levels

- The brackets enclose *"shareable data"*, i.e.: application arguments
- The level of a term represents *the number of ways it can be shared*:

$$A\ (B\ C)$$

- $(B\ C)$ can be shared inside A: to avoid conflicts between the fans of the two terms, one must be put at a (conventionally) higher level
- Similarly, C can be shared both in A and B, thus its level $= 2$
- The level is increased when we pass in the *argument of an application*
- When does a term "lose" a level of sharing? *When it is accessed by a variable inside a function (which will be at a lower level)*



Appendix: BOHM Rules for Lists, Booleans, Conditionals