



fête
pour

Georges Gonthier

23 / 06 / 25

Inria Paris

jean-jacques.levy@inria.fr
jeanjacqueslevy.net/talks



Le Monde



Consulter
le journal

Actualités ▾

Économie ▾

Vidéos ▾

Débats ▾

Culture ▾

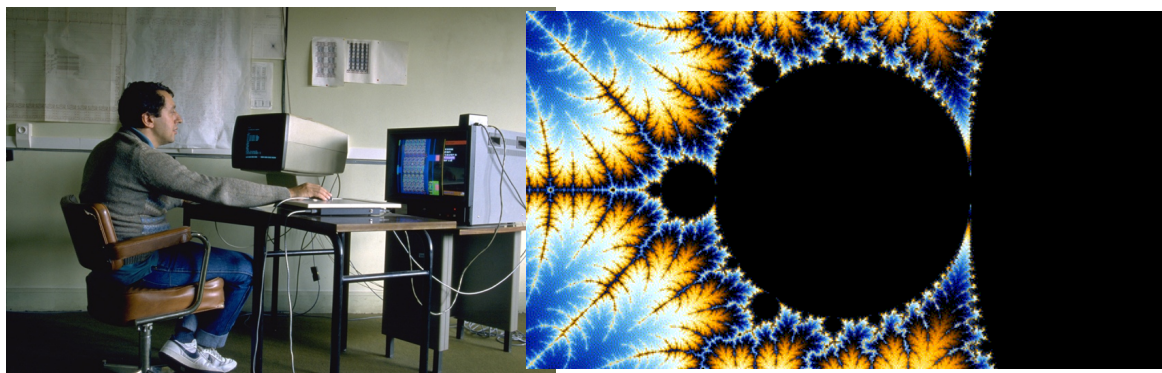
Le Goût du Monde ▾

SCIENCES

Portrait Georges Gonthier, « hacker » de théorèmes

L'informaticien a développé des programmes de vérifications des preuves mathématiques

Plan



colorix '86

sophia

phD '88

Sémantiques et modèles d'exécution des langages réactifs synchrones :
application à Esterel / Georges Gonthier ;
[sous la direction de Gérard Berry]

bell labs

lambda opt '91



rocq

join calculus

The join-calculus is a process calculus developed at INRIA.
with Cédric Fournet

gc tlp '95

**Verifying the Safety of a Practical Concurrent
Garbage Collector**
Georges Gonthier
INRIA Rocquencourt
78153 LE CHESNAY CEDEX
FRANCE

**Portable, unobtrusive garbage collection
for multiprocessor systems (POPL'94)**
Damien Doligez, Georges Gonthier

cambridge

4 couleurs '04

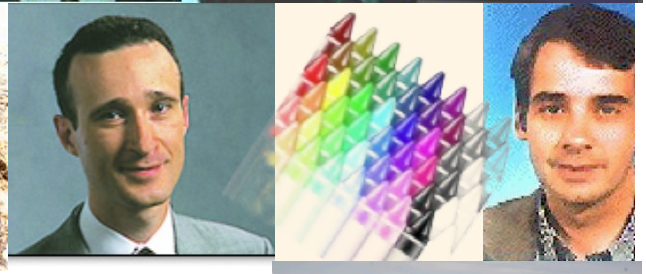


feit thompson '12

math comp
[ssr, bigops, .20 ans]



saclay



a501



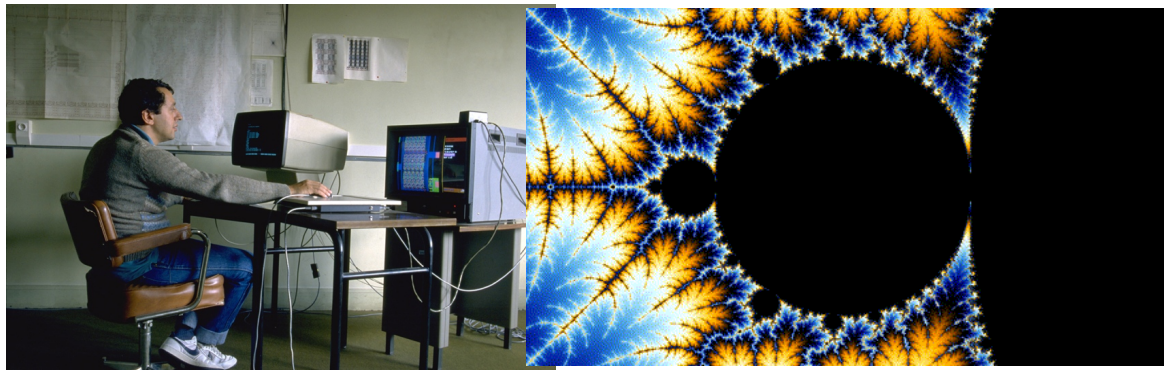
X- pi

X- cc

X-concours



Plan



colorix '86

sophia

phD '88

Sémantiques et modèles d'exécution des langages réactifs synchrones :
application à Esterel / Georges Gonthier ;
[sous la direction de Gérard Berry]

bell labs

lambda opt '91



join calculus

The join-calculus is a process calculus developed at INRIA.
with Cédric Fournet

gc tlp '95

**Verifying the Safety of a Practical Concurrent
Garbage Collector**
Georges Gonthier
INRIA Rocquencourt
78153 LE CHESNAY CEDEX
FRANCE

**Portable, unobtrusive garbage collection
for multiprocessor systems (POPL'94)**
Damien Doligez, Georges Gonthier

4 couleurs '04

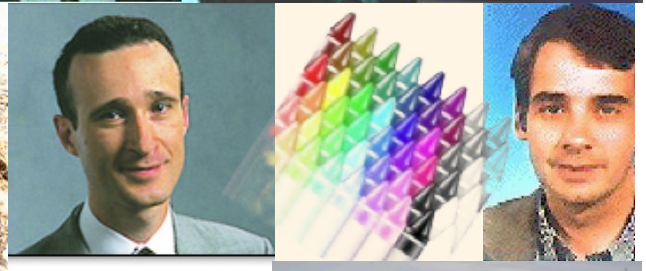


feit thompson '12

math comp
[ssr, bigops, .20 ans]



saclay



a501



X- pi

X- cc

X-concours



joint centre

Microsoft Research - Inria
JOINT CENTRE

cambridge

Plan

sophia

bell labs

rocq

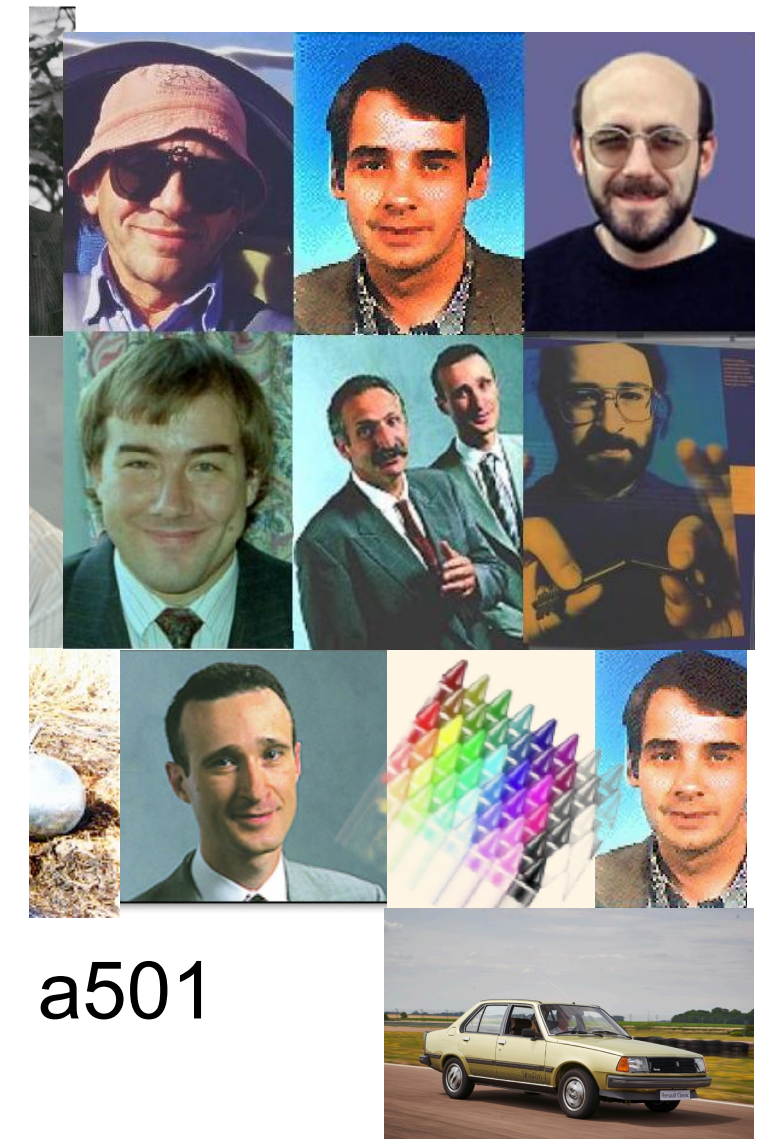
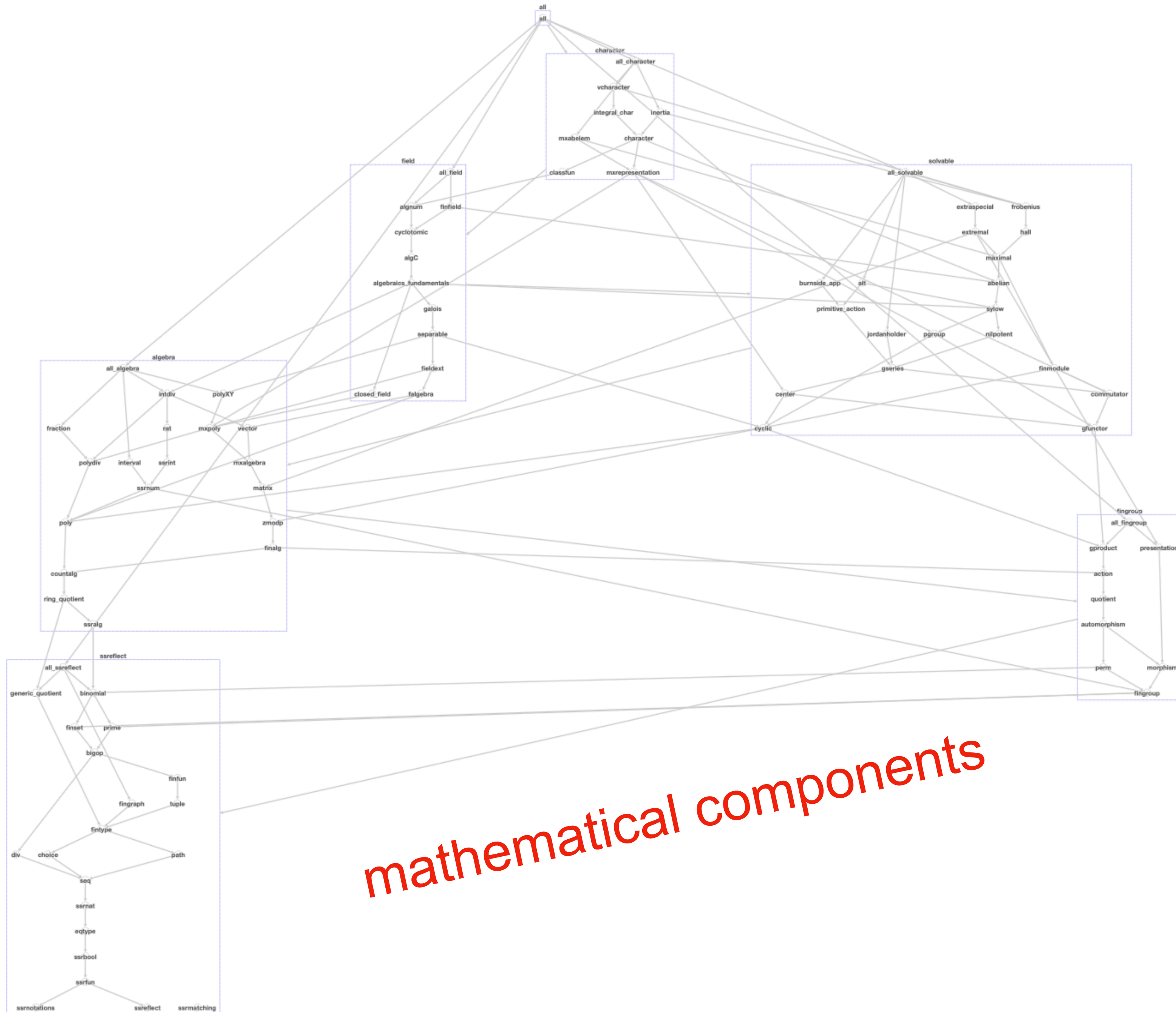
cambridge

joint centre

saclay



Microsoft Research - Inria
JOINT CENTRE



a501

X- pi

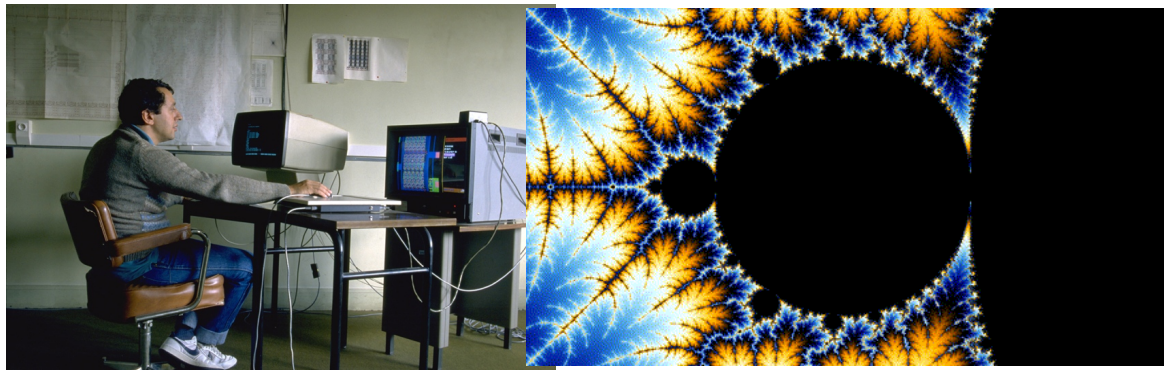
X- cc

X-concours



mathematical components

Plan



colorix '86

sophia

phD '88

Sémantiques et modèles d'exécution des langages réactifs synchrones :
application à Esterel / Georges Gonthier ;
[sous la direction de Gérard Berry]

bell labs

lambda opt '91



join calculus

The join-calculus is a process calculus developed at INRIA.
with Cédric Fournet

gc tlp '95

**Verifying the Safety of a Practical Concurrent
Garbage Collector**
Georges Gonthier
INRIA Rocquencourt
78153 LE CHESNAY CEDEX
FRANCE

**Portable, unobtrusive garbage collection
for multiprocessor systems (POPL'94)**
Damien Doligez, Georges Gonthier

4 couleurs '04



feit thompson '12

math comp
[ssr, bigops, .20 ans]



saclay



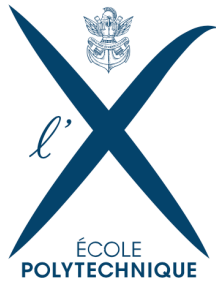
a501



X- pi

X- cc

X-concours



joint centre

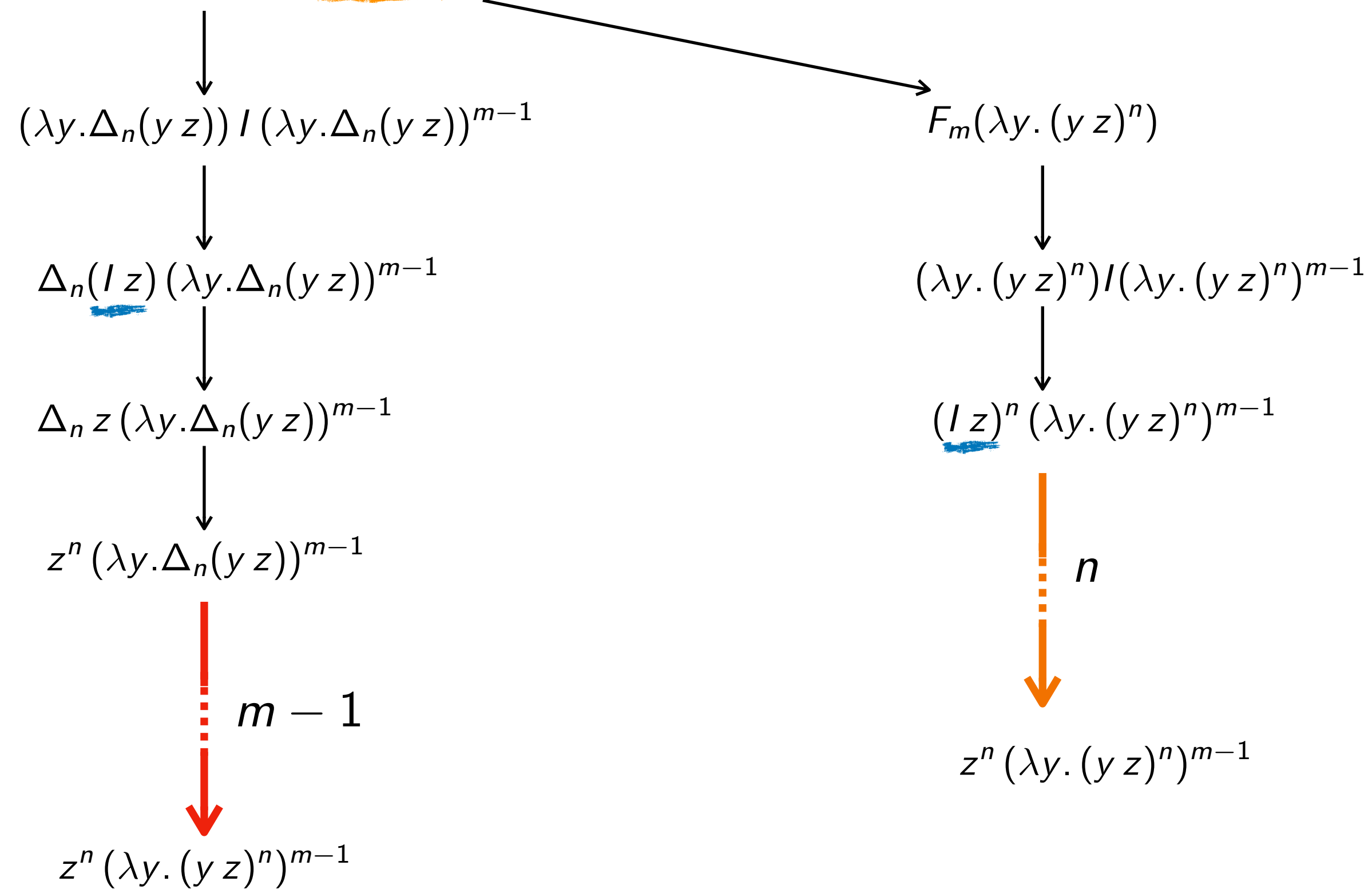
Microsoft Research - Inria
JOINT CENTRE

duplications
in
the λ -calculus

Initial motivation (side remark)

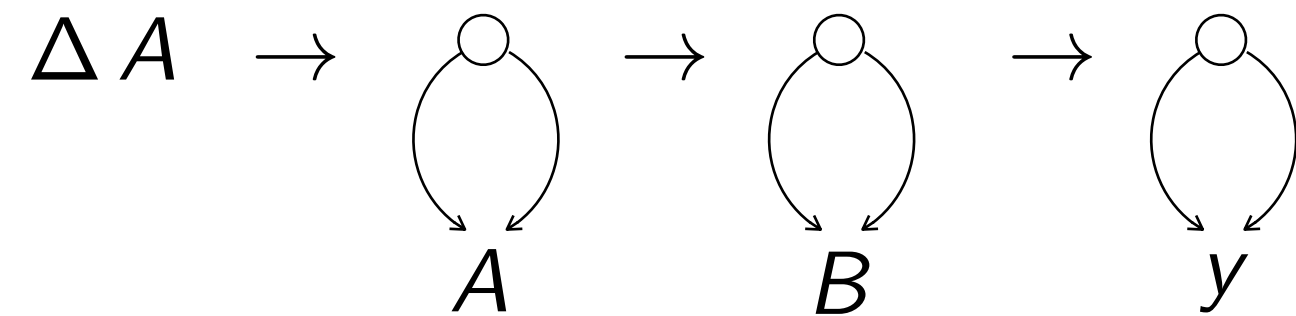
- no single-redex reduction strategy can be optimal

example $F_m(\lambda y. \Delta_n(y z))$ where $F_m = \lambda x. x I x x \dots x$ and $\Delta_n = \lambda x. x x \dots x$



Duplication of redexes

- call-by-need is **call-by-name** with **sharing** to avoid duplications

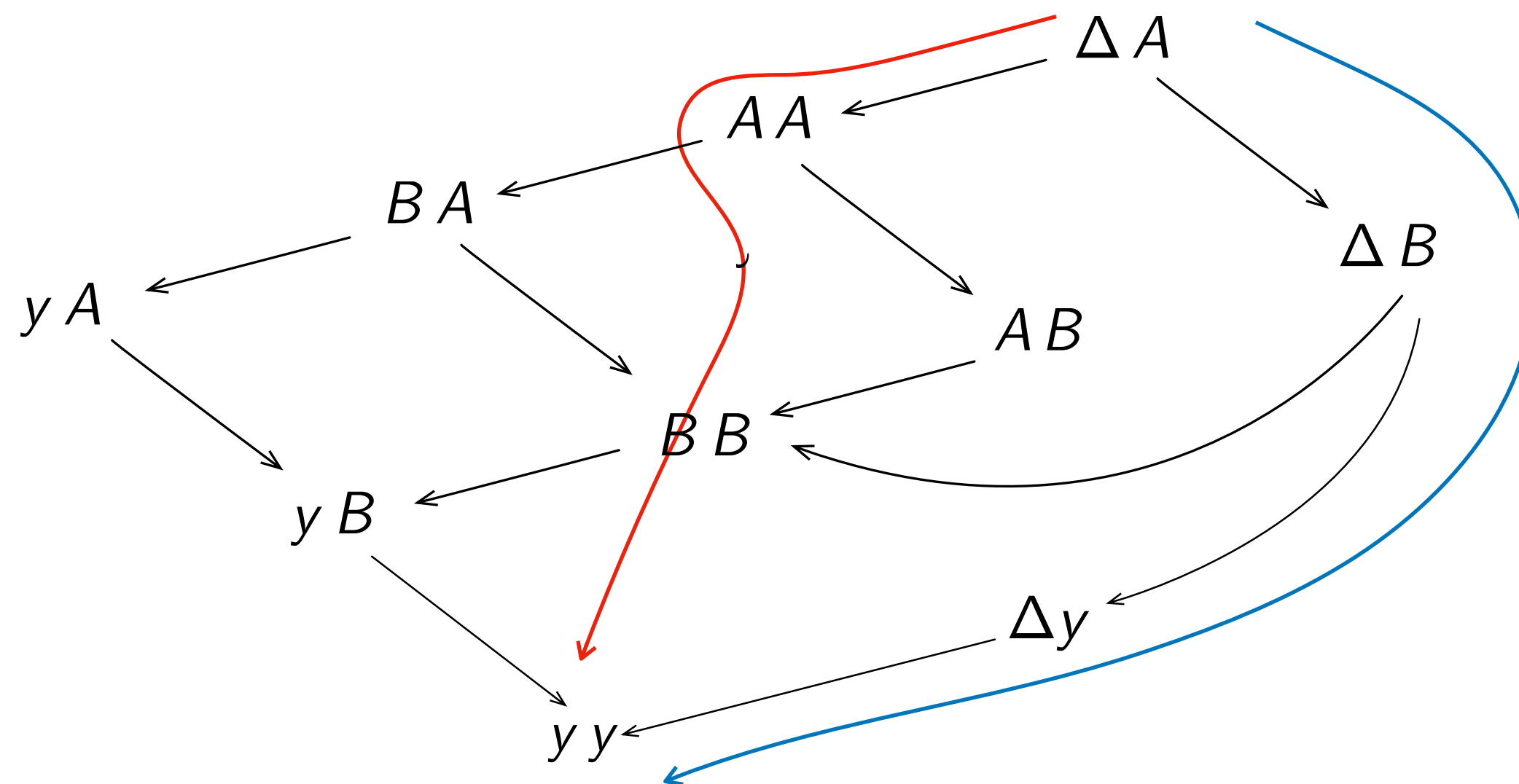


$$\Delta = \lambda x. x x$$

$$F = \lambda f. f y$$

$$I = \lambda x. x$$

$$A = F I \rightarrow I y = B \rightarrow y$$

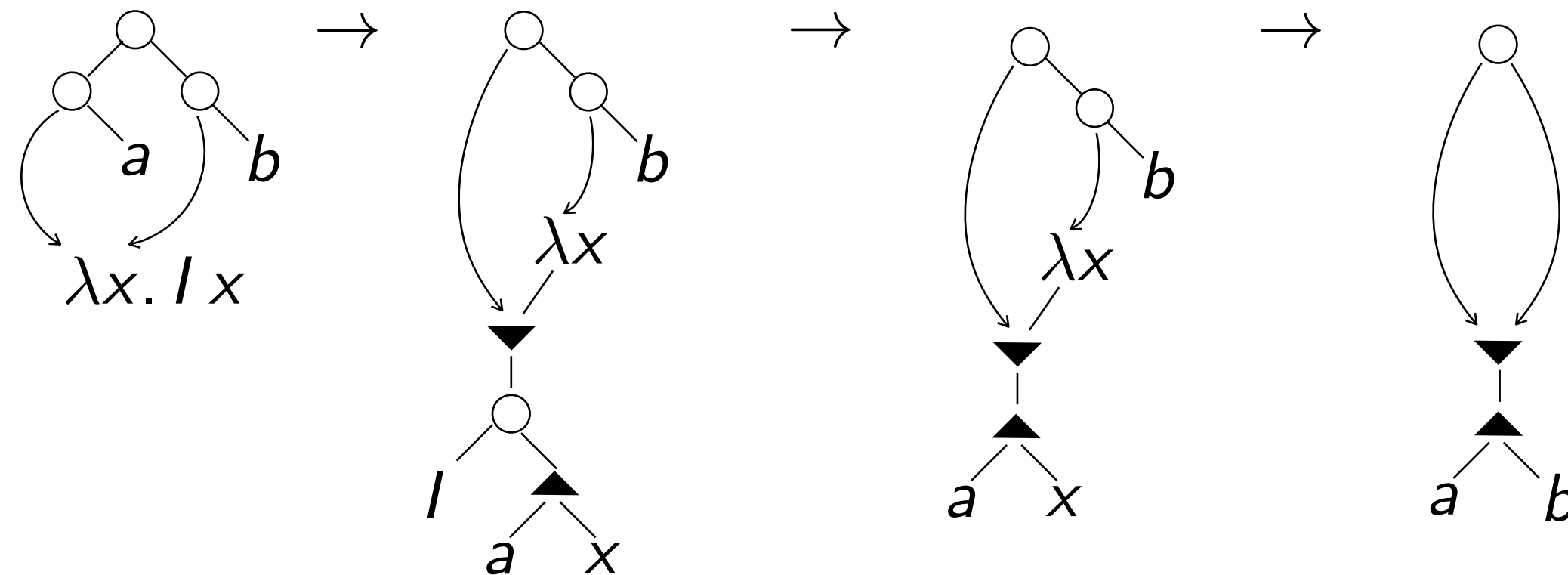


- optimal reductions has to deal with **parallel reductions**

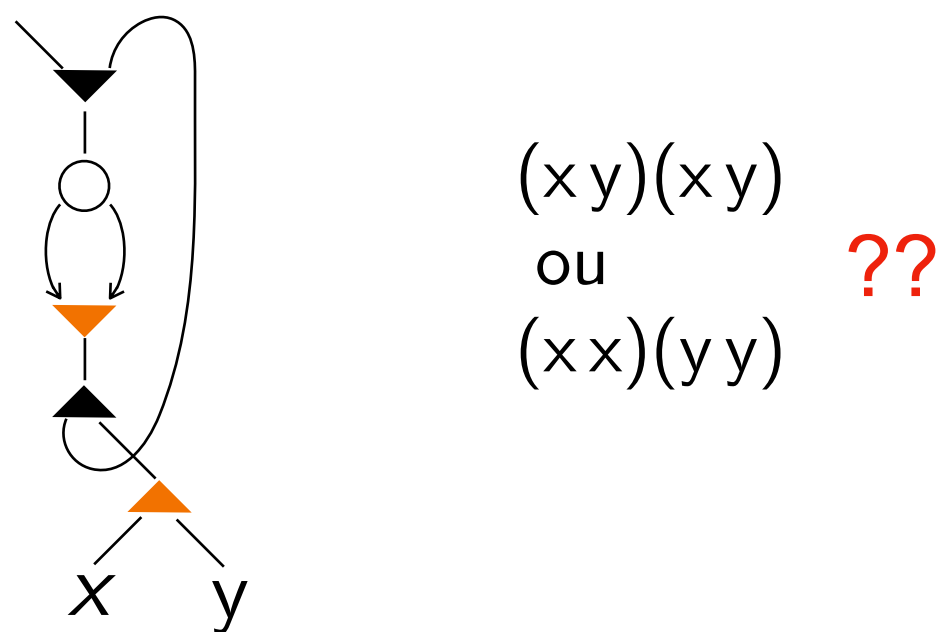
Duplication of redexes

- sharing λ -abstractions ? $I = \lambda x. x$

$(\lambda f. f a (f b))(\lambda x. I x) \rightarrow$



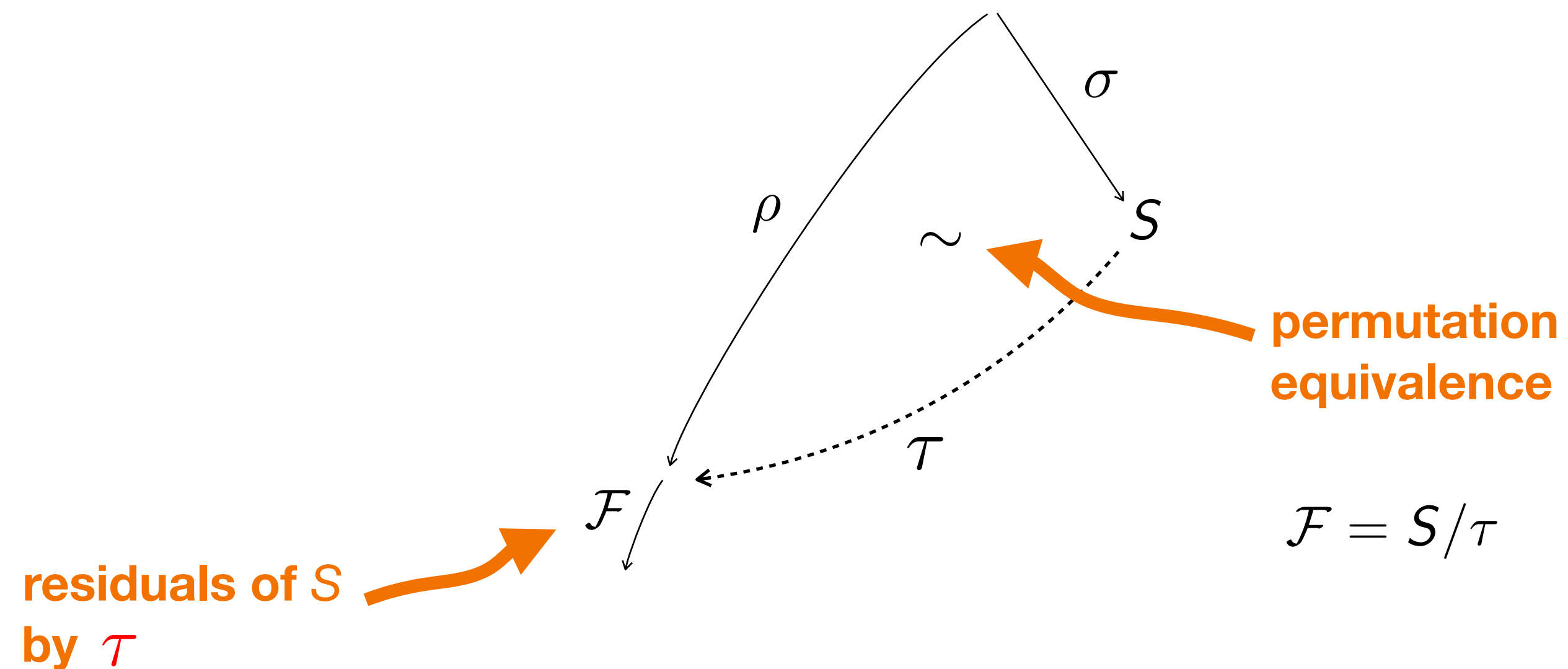
- possible sharing !!



- sharing contexts ? with boxes and tags ??
- back to easier theory ...

Duplication complete reductions

- a reduction step $\xrightarrow{\mathcal{F}}$ is duplication complete if $\mathcal{F}$ is a maximal set of redexes residuals of a single redex modulo permutations

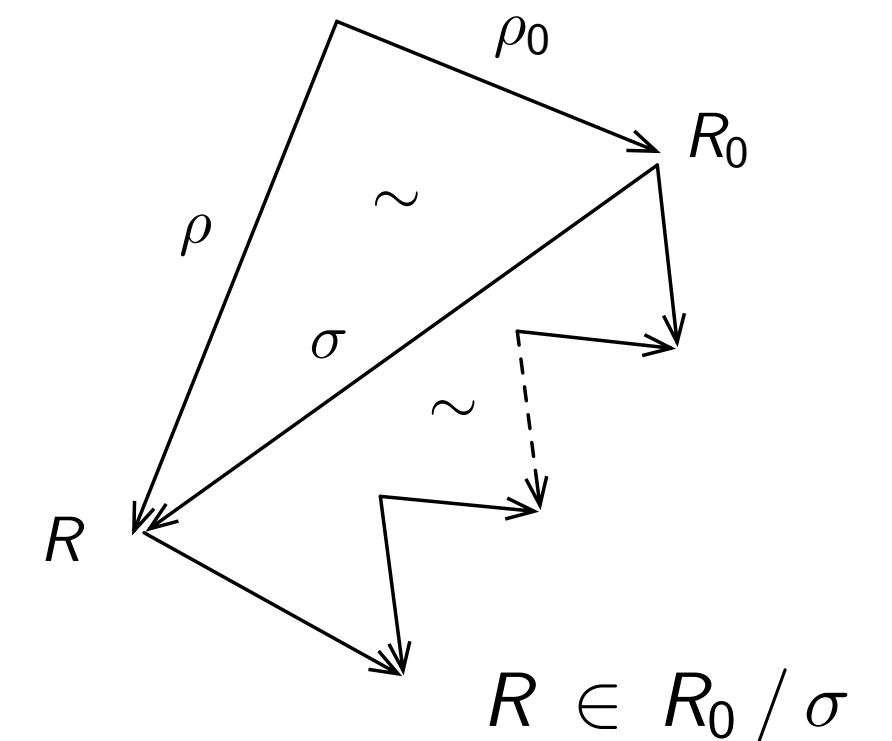
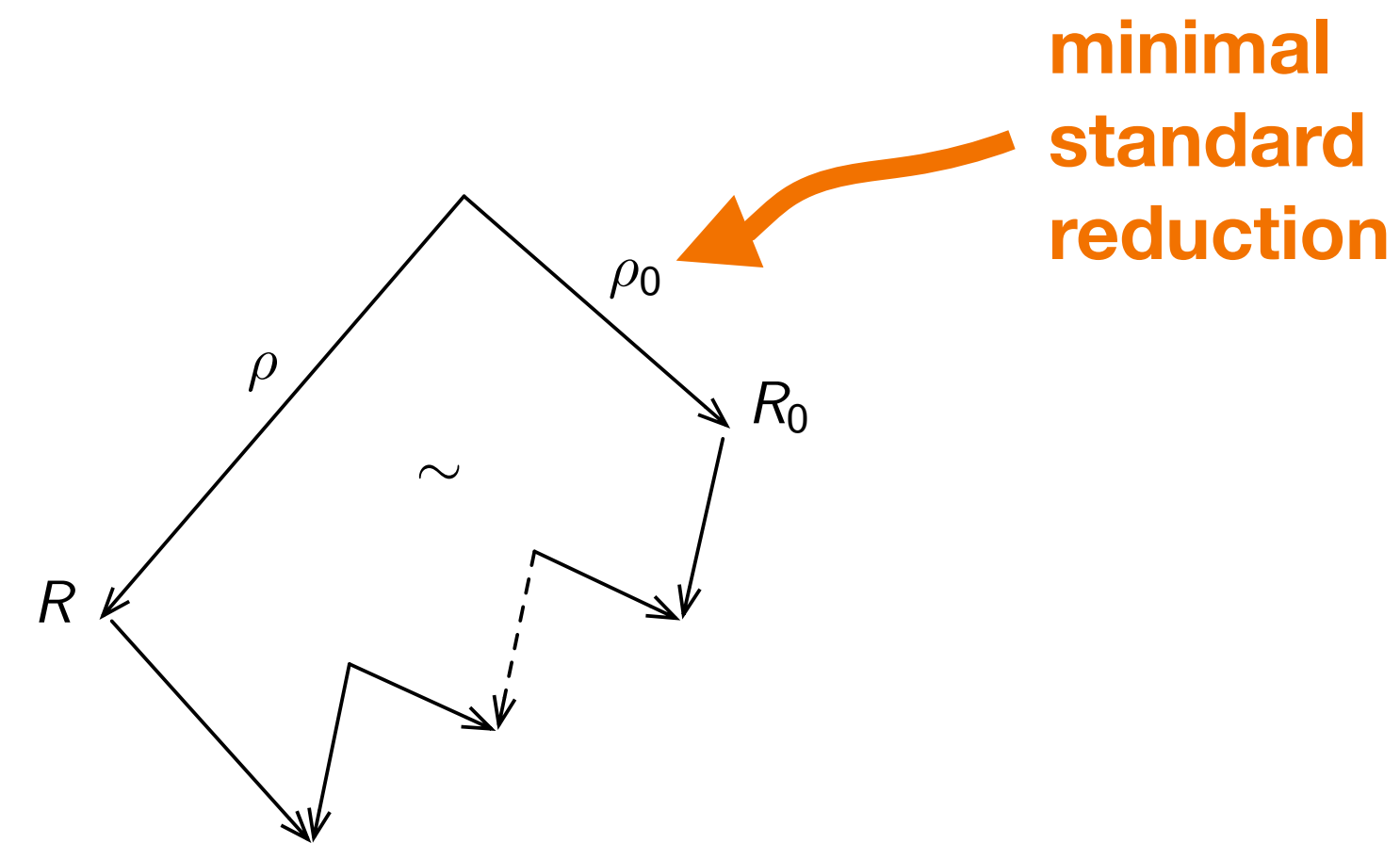
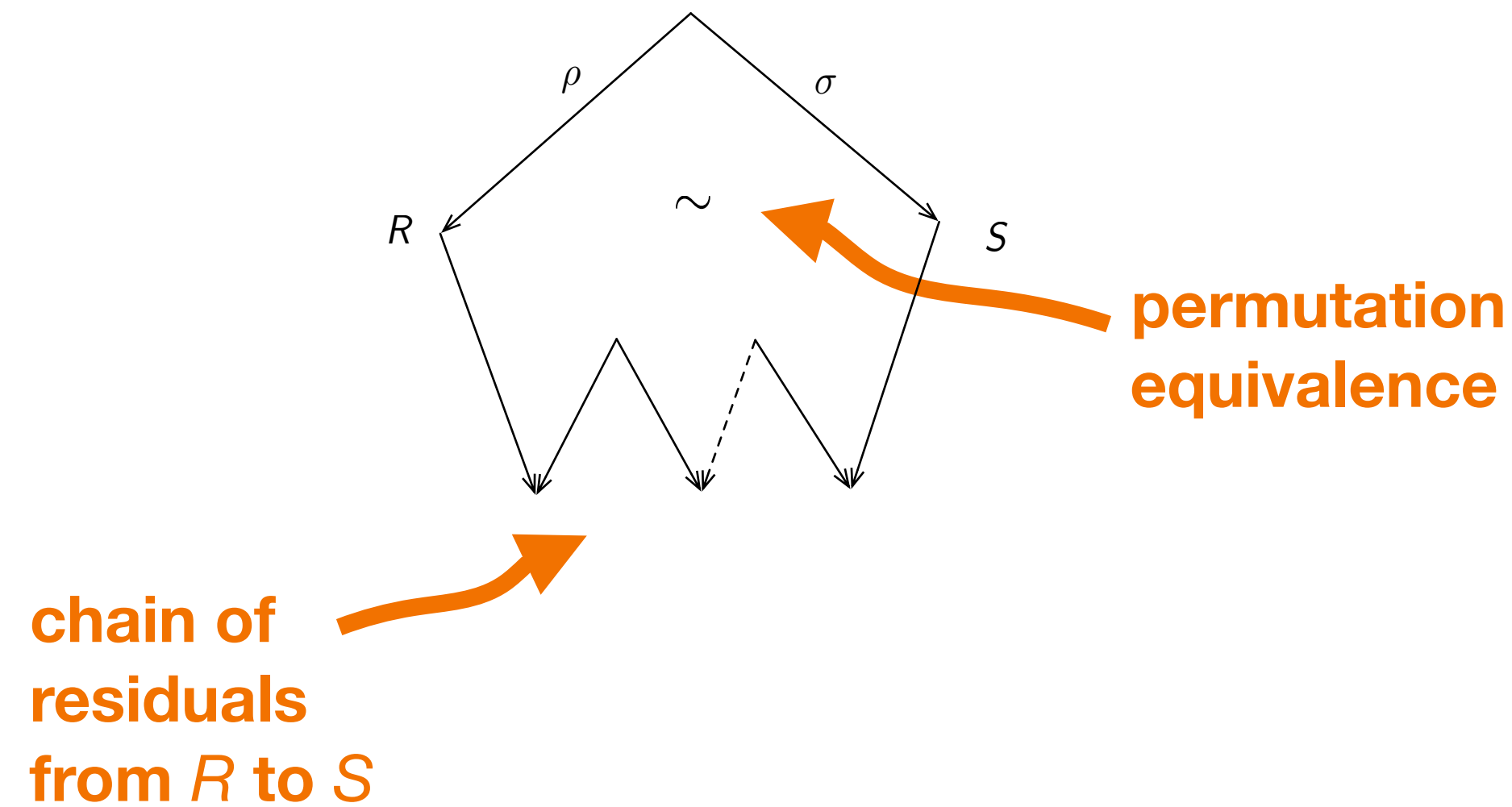


Goal [optimality thm] leftmost-outermost d-complete reductions are optimal in their number of parallel reduction steps

- but how to find $\langle \sigma, S \rangle$?

Redex families

- family of redexes is chain of residuals modulo permutation equivalence

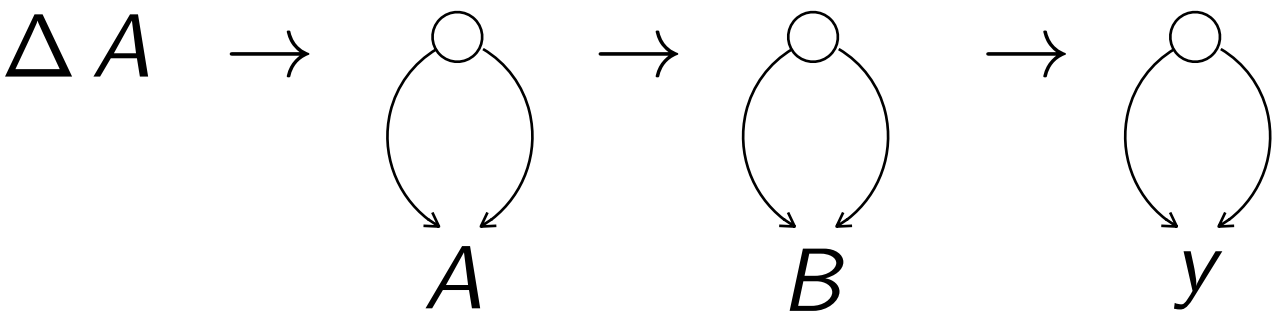


- in each family, canonical representative is unique with standard reduction of minimal length (*stability property*)
- below the canonical representative reduction, family is same as duplication of the canonical representative

Fact [for optimality thm] d-complete reductions are same as family-complete reductions

Duplication of redexes

- call-by-need is call-by-name with sharing to avoid duplications

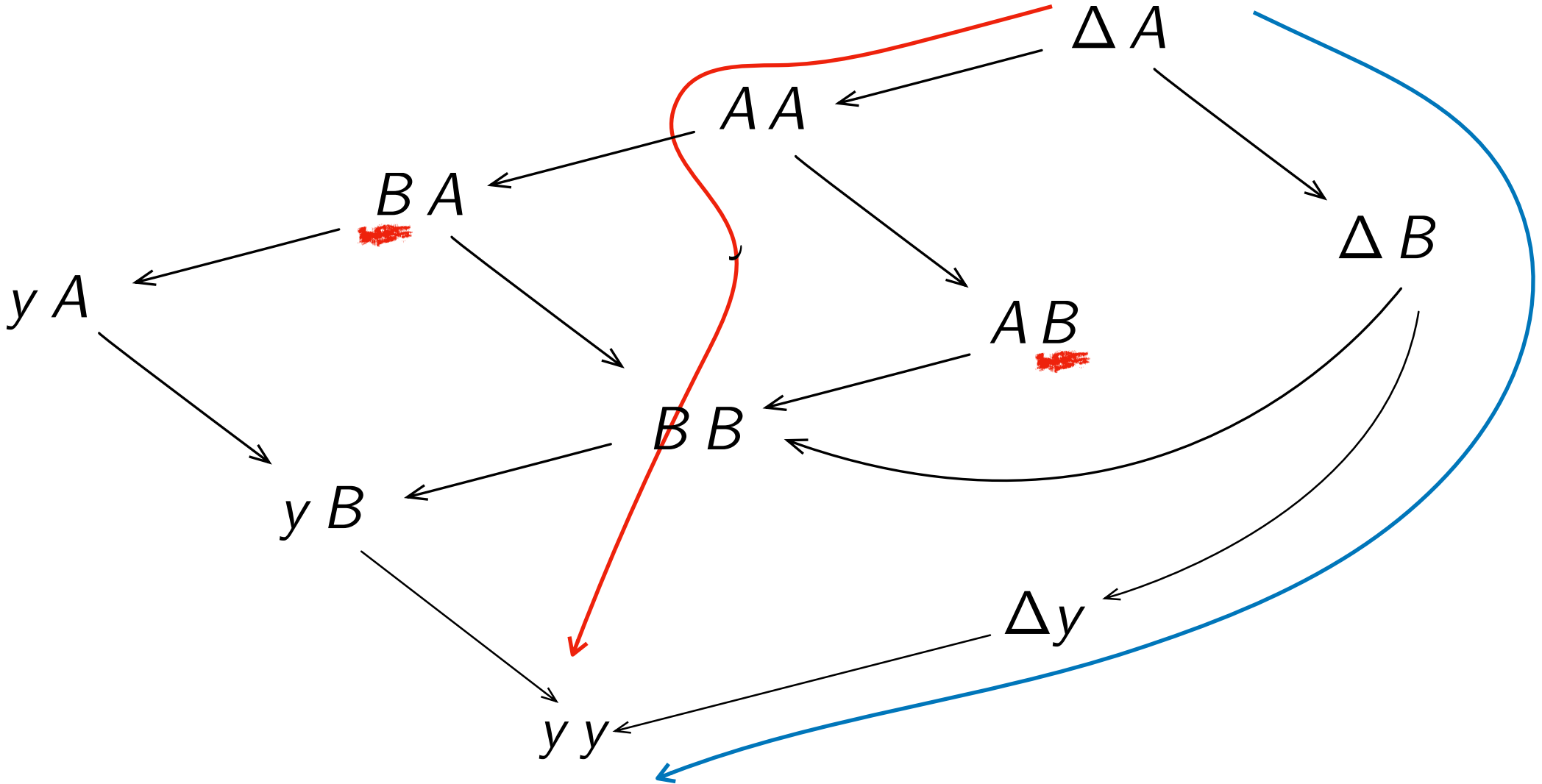


$$\Delta = \lambda x. x x$$

$$F = \lambda f. f y$$

$$I = \lambda x. x$$

$$A = F I \rightarrow I y = B \rightarrow y$$

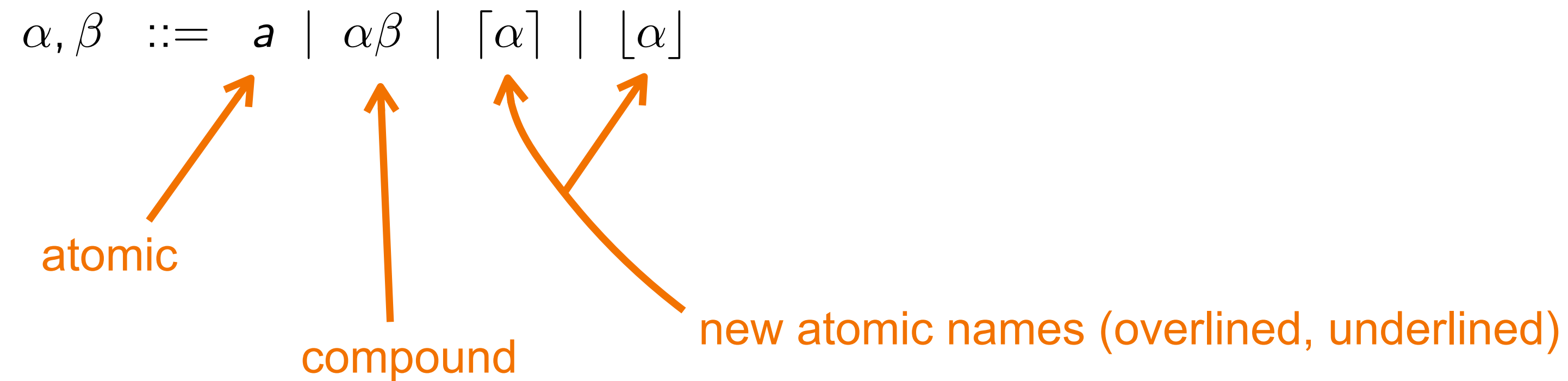


- optimal reductions has to deal with parallel reductions

labeled
calculus

The labeled λ -calculus

- labels over alphabet $\mathcal{A} = \{a, b, c, \dots\}$



- labeled λ -calculus

$$M, N, \dots ::= x \mid MN \mid \lambda x.M \mid M^\alpha$$

$$(\lambda x.M)^\alpha N \rightarrow M\{x := N^{\underline{\alpha}}\}^{\overline{\alpha}}$$

$$M^\alpha\{x := N\} = M\{x := N\}^\alpha$$

$$(M^\alpha)^\beta = M^{\alpha\beta}$$

The labeled λ -calculus

- again an alphabet of atomic labels $\mathcal{A} = \{a, b, c, \dots\}$
- their labeled λ -calculus [Asperti-Laneve]

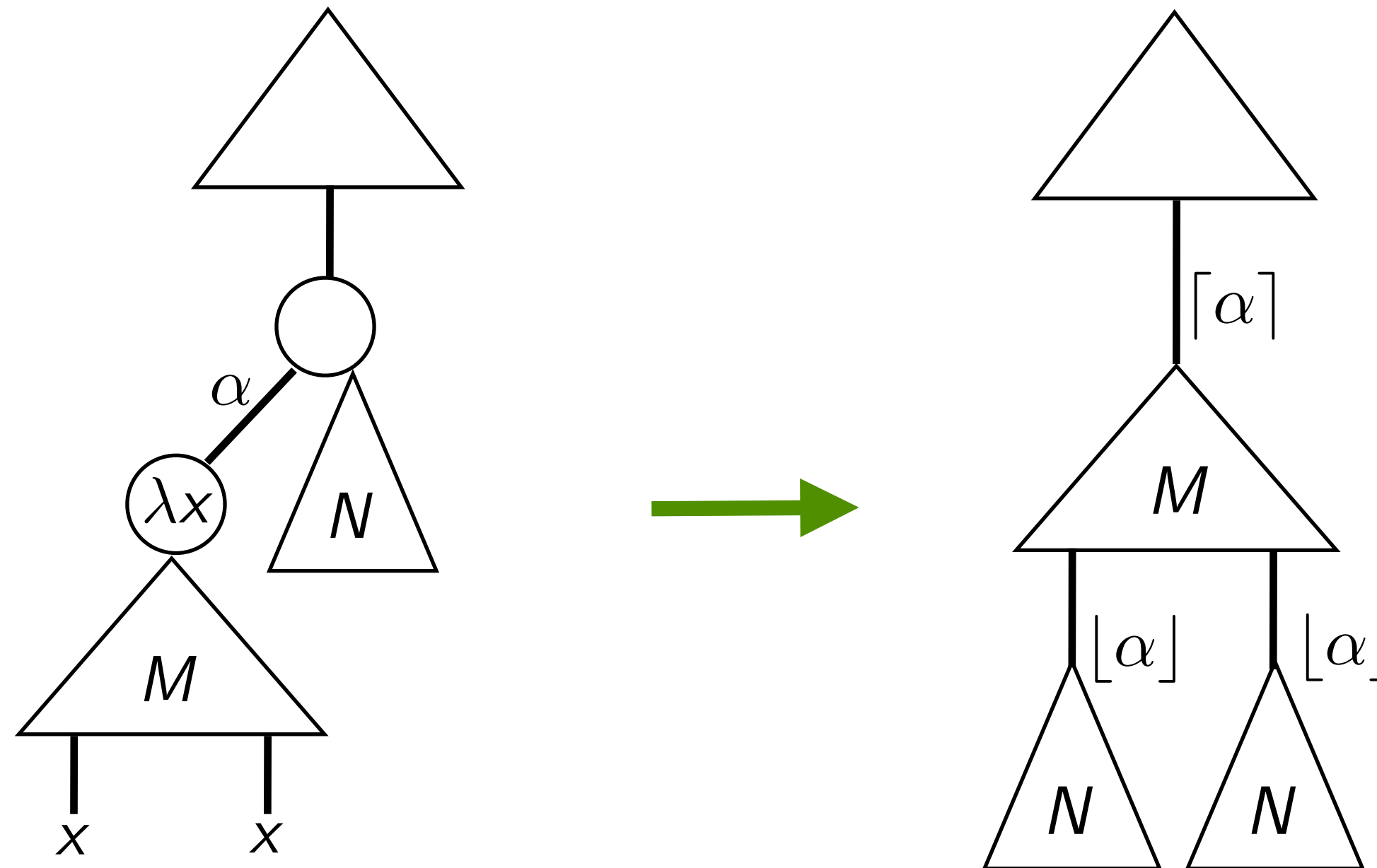
$$M, N, \dots ::= x \mid MN \mid \lambda x.M \mid a:M$$

$$(a_1 : a_2 : \dots a_n : \lambda x.M)N \rightarrow a_1 : a_2 : \dots a_n : M\{x := a_n : a_{n-1} : \dots a_1 : N\}$$

$$(a : M)\{x := N\} = a : M\{x := N\}$$

- correspondence with paths in initial term

The labeled λ -calculus



Theorem [optimality] leftmost-outermost **labeled-complete** reductions are optimal in their number of parallel reduction steps

Theorem [labeled family] redexes have same labeled names iff they are in same family

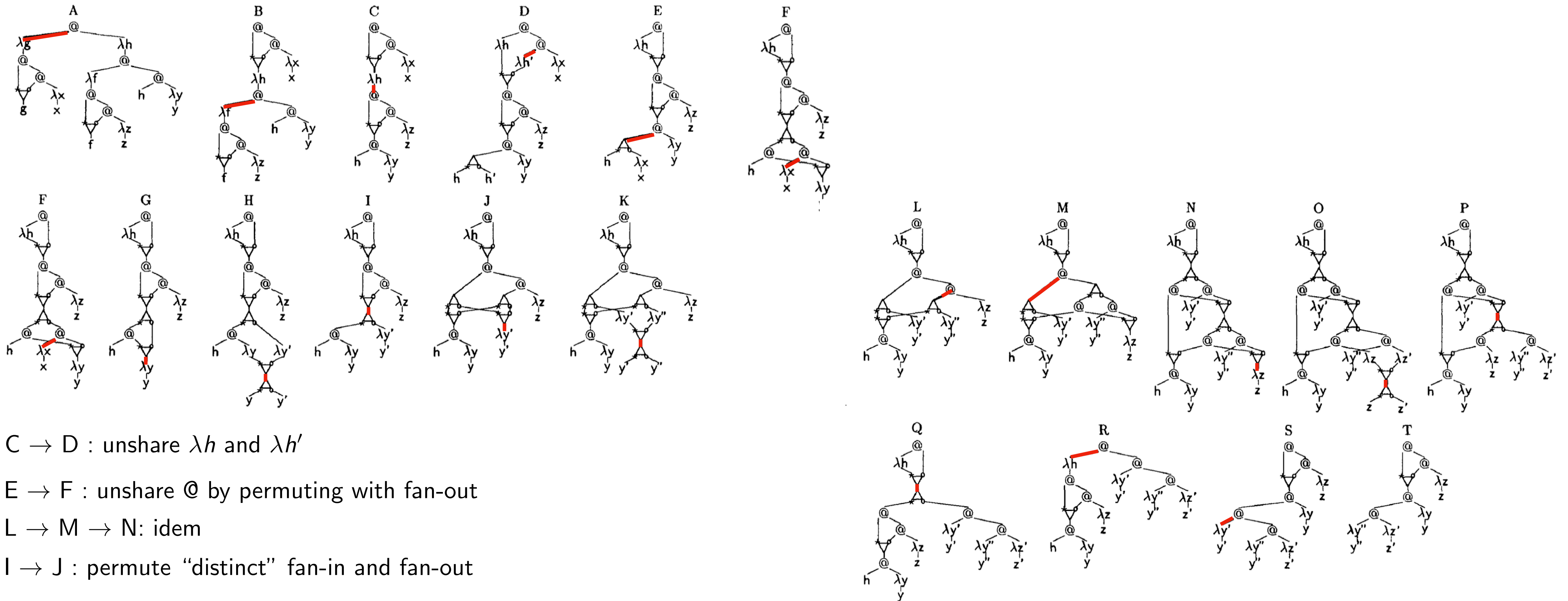
Lamping algorithm

Lamping algorithm

- implements optimal reductions [Lamping 90]

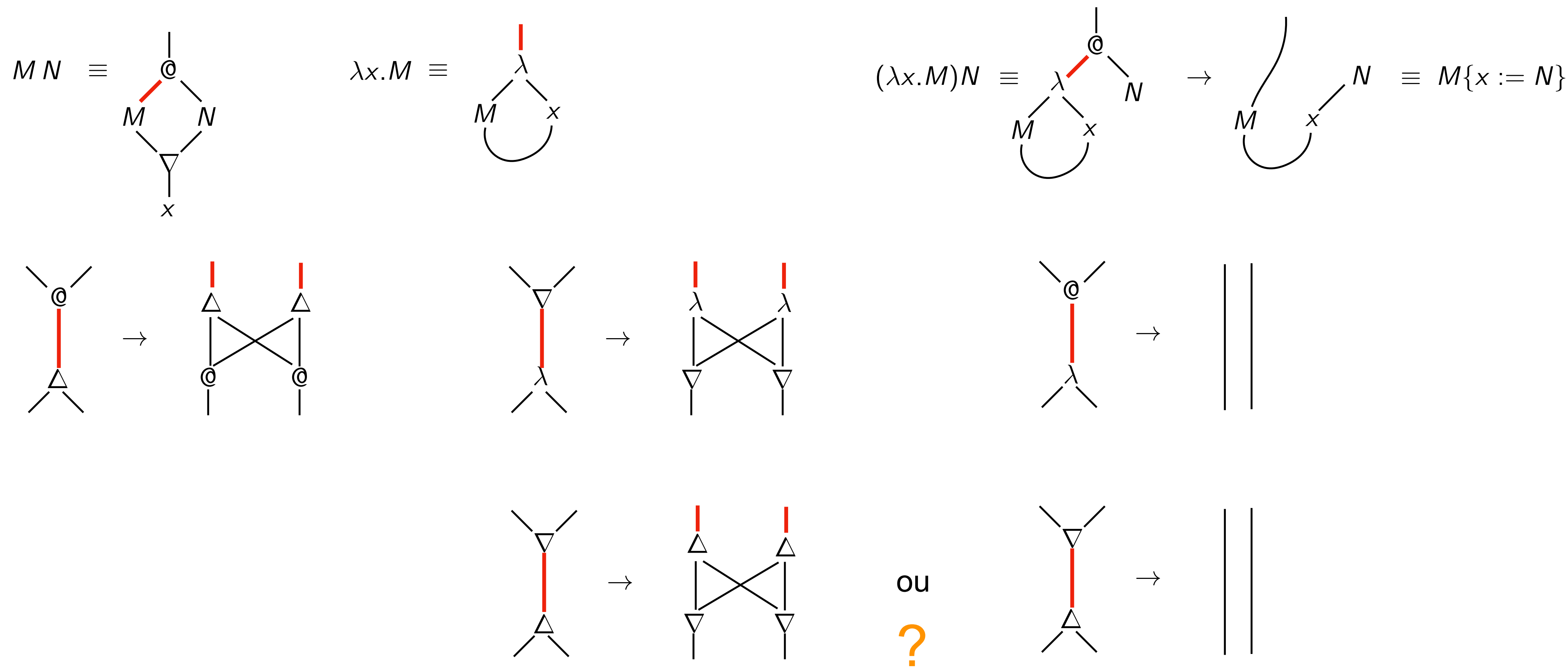
$$I = \lambda x.x = \lambda y.y = \lambda z.z$$

$$A = (\lambda g.g(g I)) (\lambda h. (\lambda f.f(f I)) (h I))$$



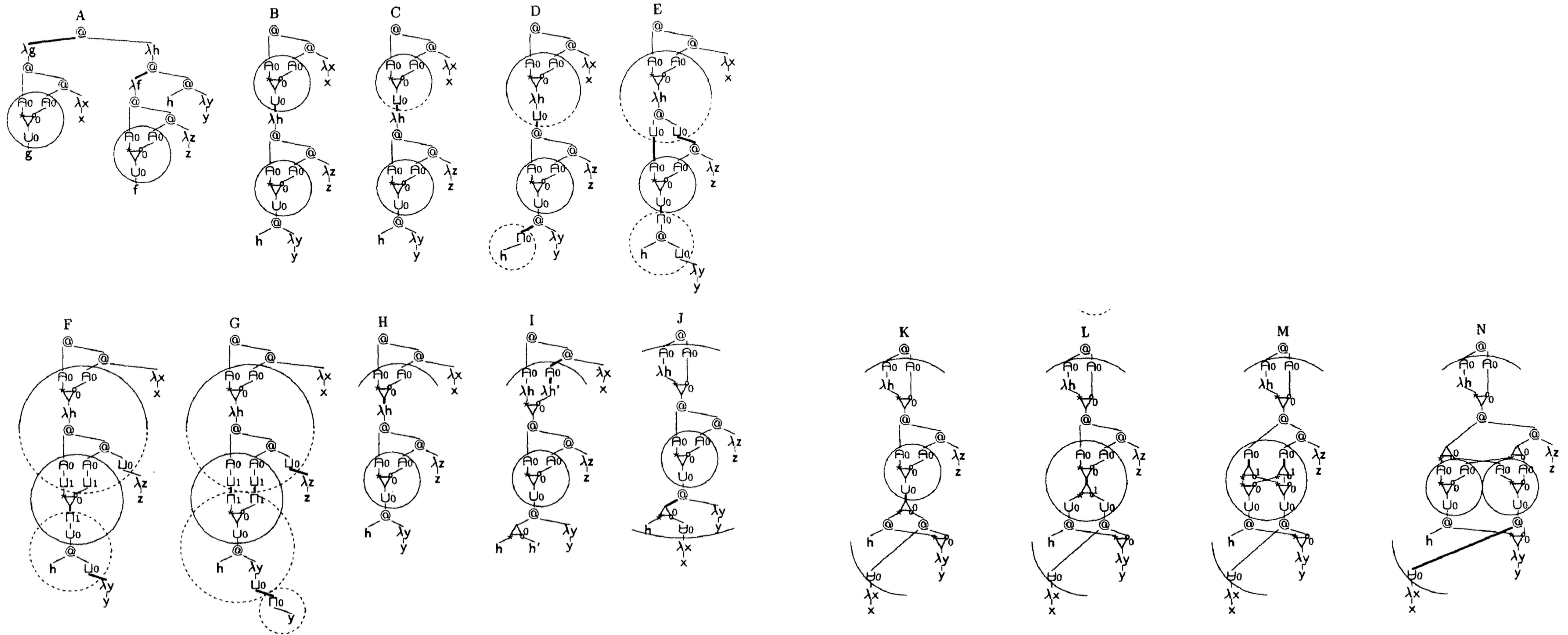
Lamping algorithm

- execution rules [Lamping 90] with interaction nets [Lafont 90]



Lamping algorithm

- implements optimal reductions [Lamping 90] with control operators ' $\mathbb{C}$ ' and ' $\mathbb{J}$ ' for fanin's and fanout's enclosures



B $\rightarrow$ C : unconditional $\sqsubset$ becomes conditional $\supset$ allowing non-contiguous enclosures

Lamping algorithm

- execution rules [Lamping 90]

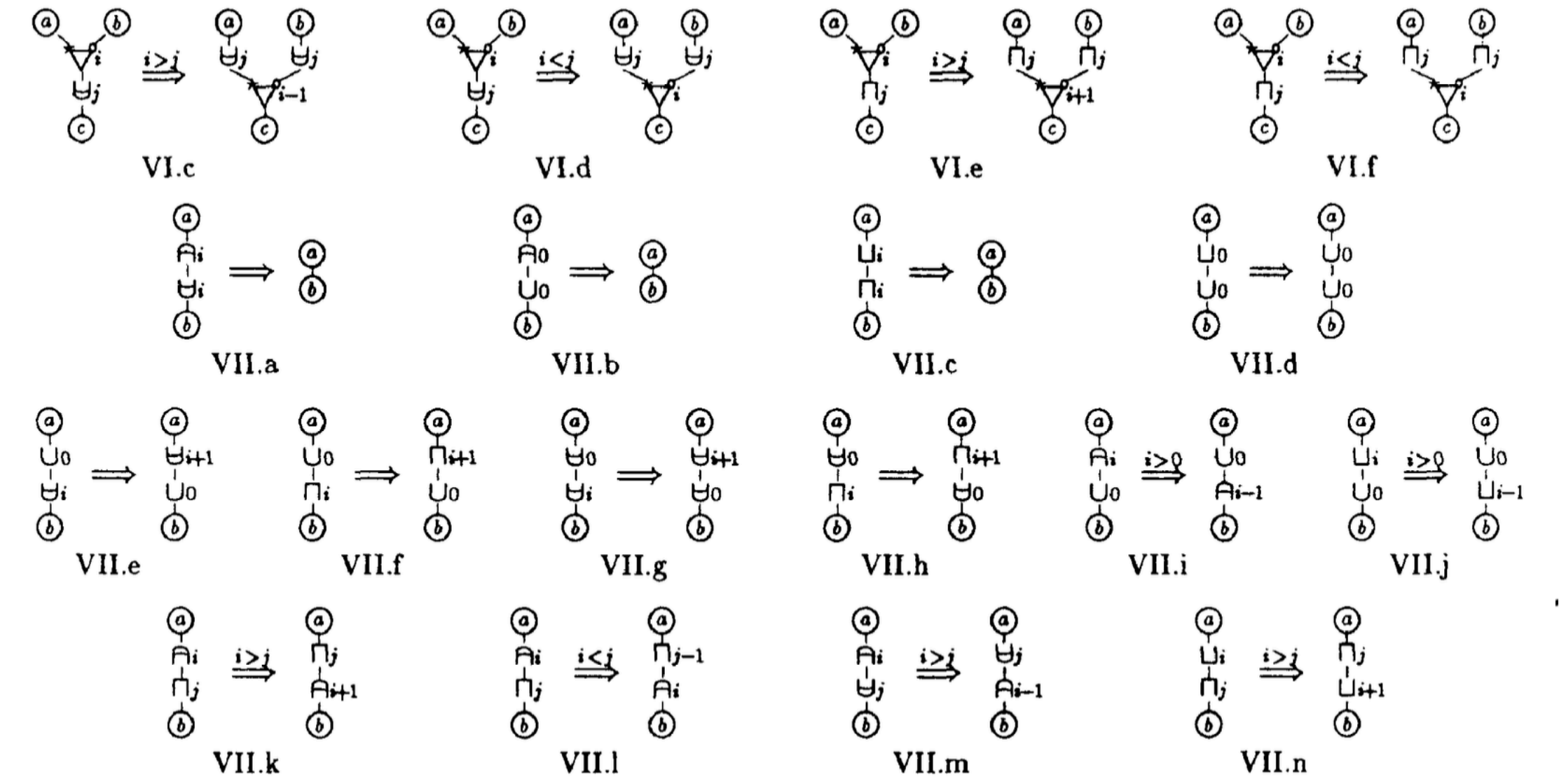
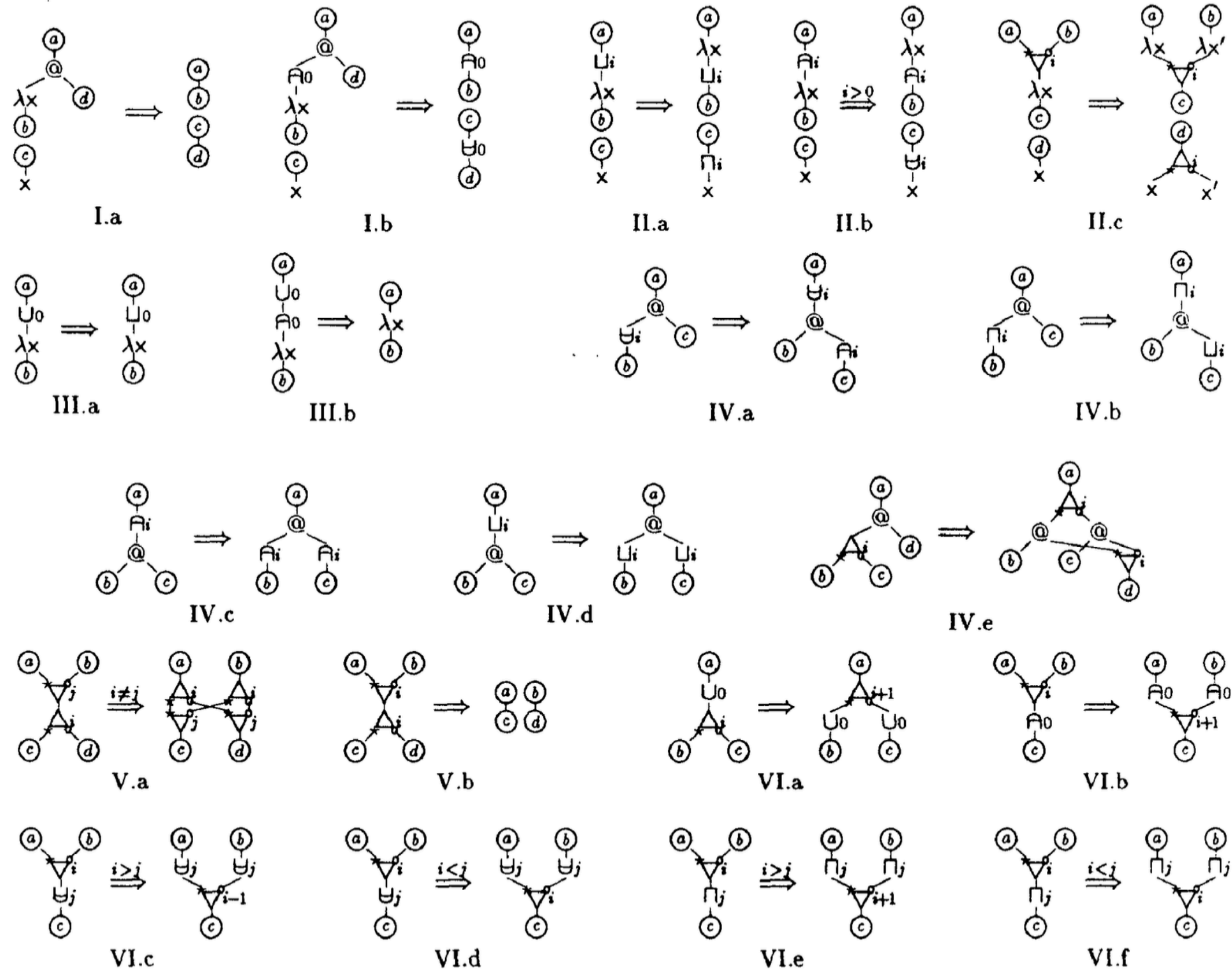
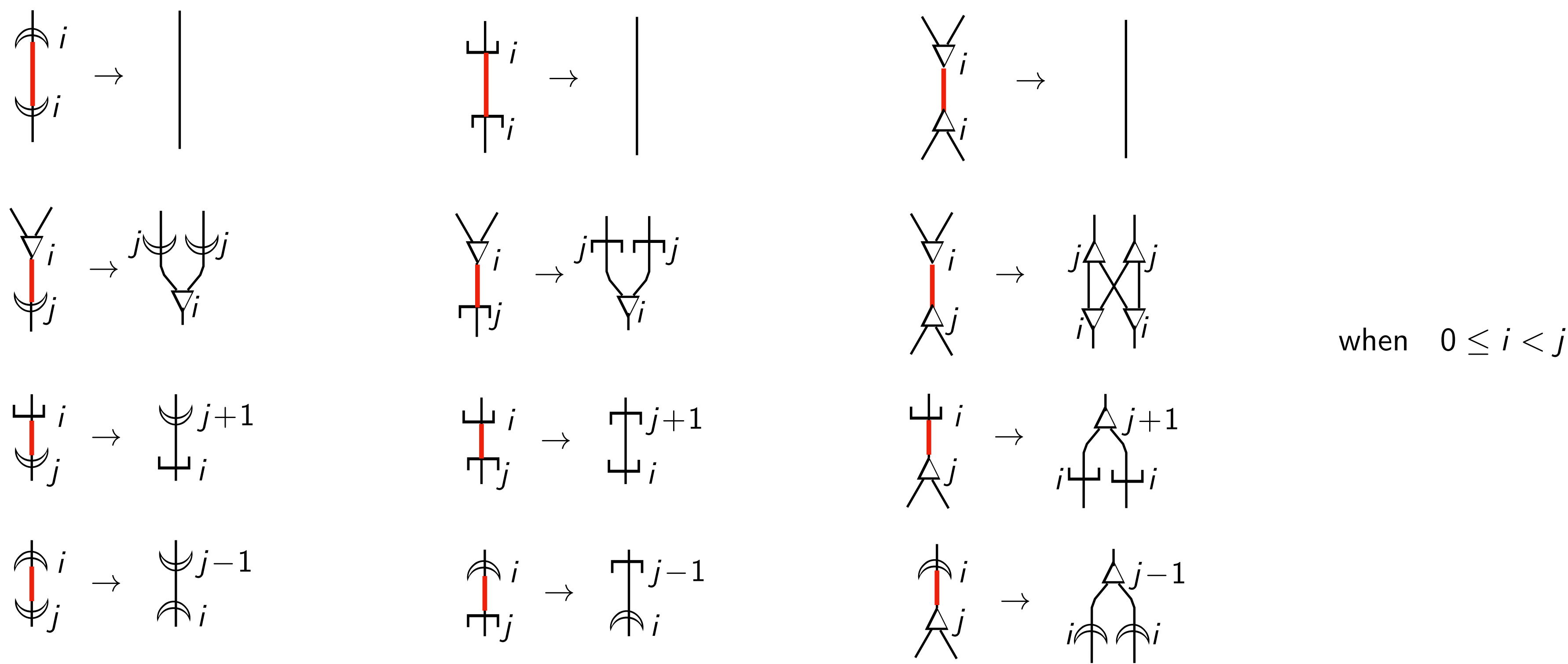


Figure 3: Rules

Gonthier algorithm

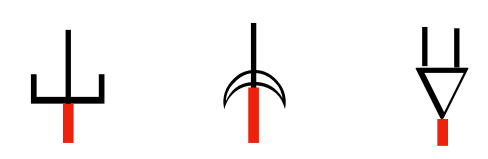
Gonthier algorithm

- fanin's ∇ and $\triangle$ fanout's behave as $@$ and λ
- simplifying Lamping's enclosures delimited by opening croissants ' $\mathbb{C}$ ' and closing square brackets ' $\mathbb{J}$ '
- all operators are indexed (their depth)



Gonthier algorithm

- operators interact with their principal ports (in red) as in linear logic

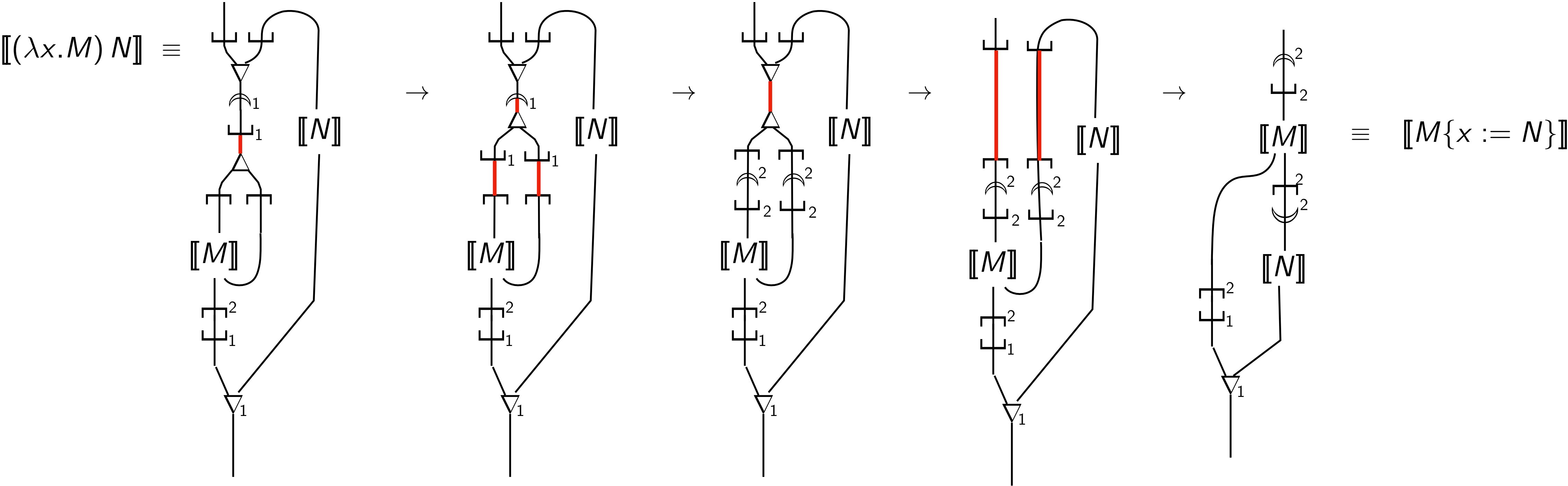
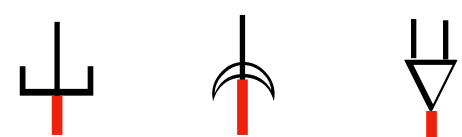


- coding of λ -expressions



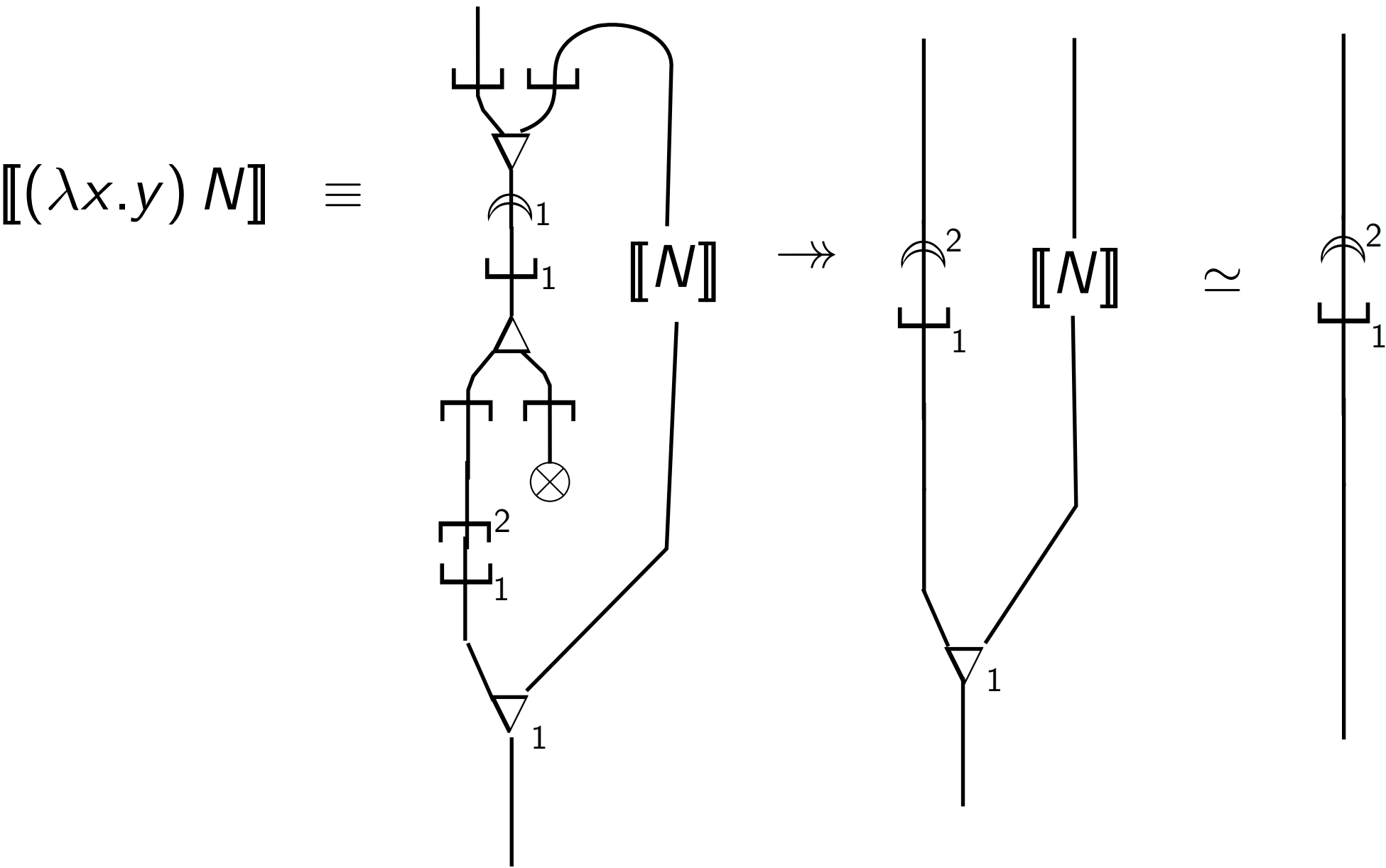
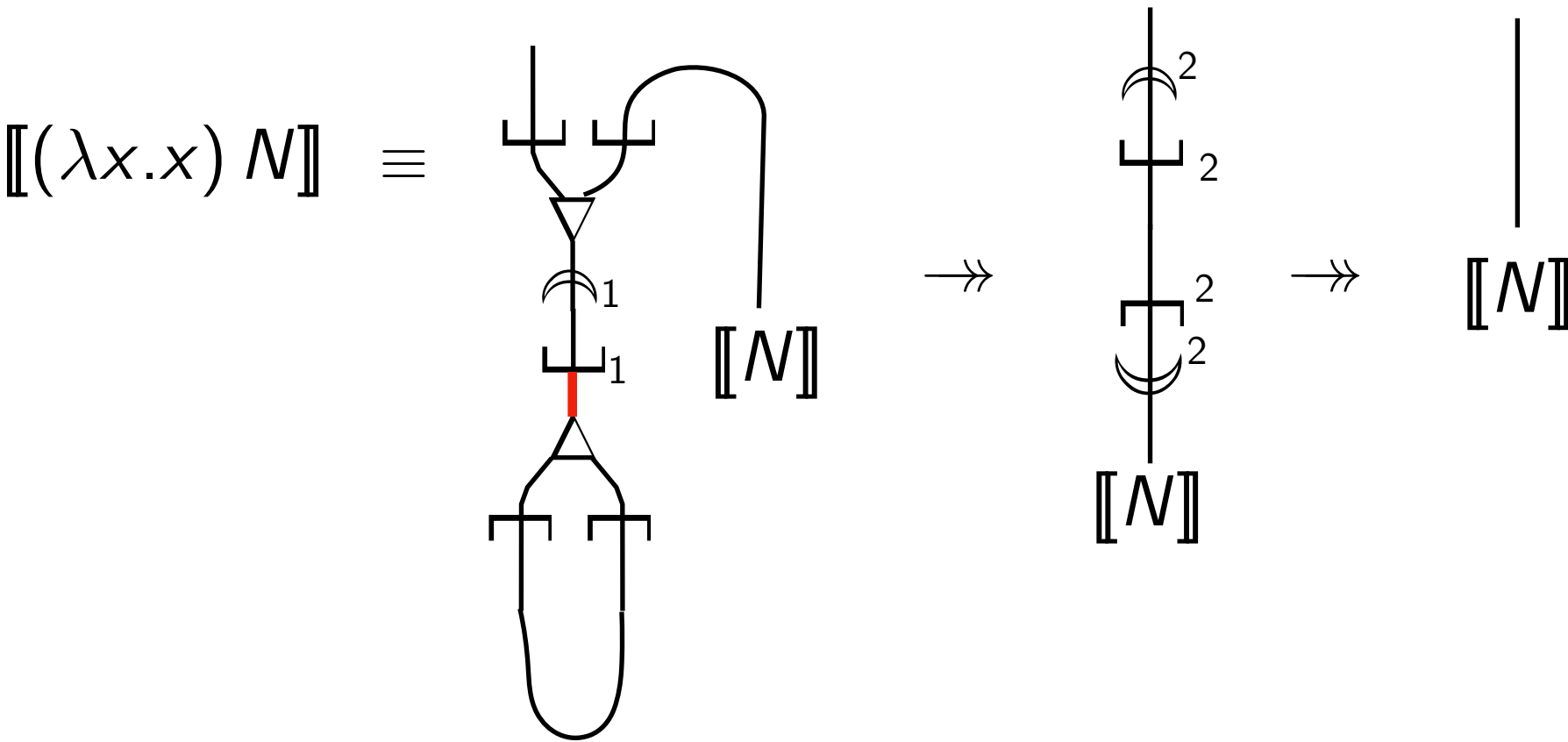
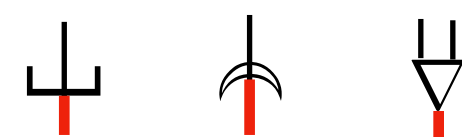
Gonthier algorithm

- beta-reduction step



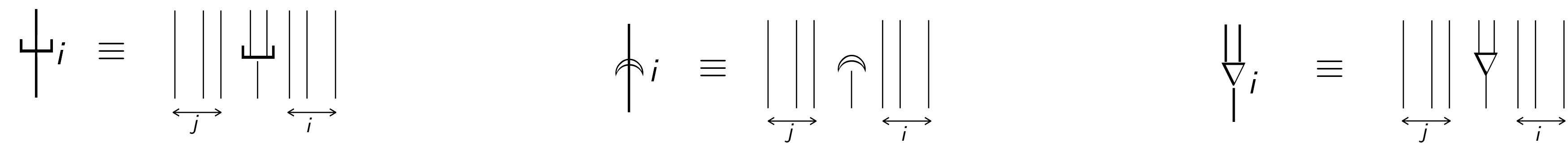
Gonthier algorithm

• examples

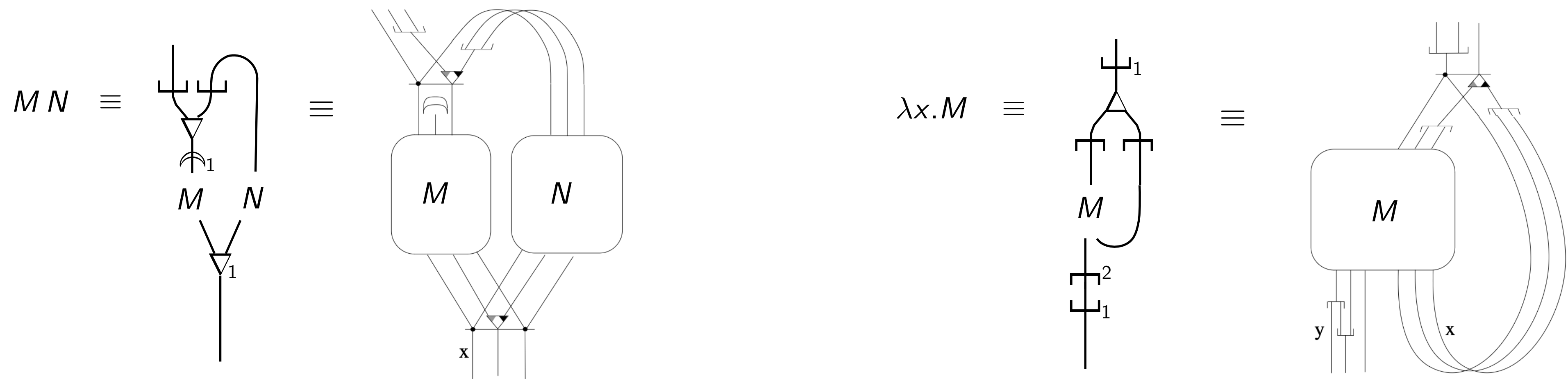


Gonthier algorithm

- operators can be decomposed with a (weird) bus notation

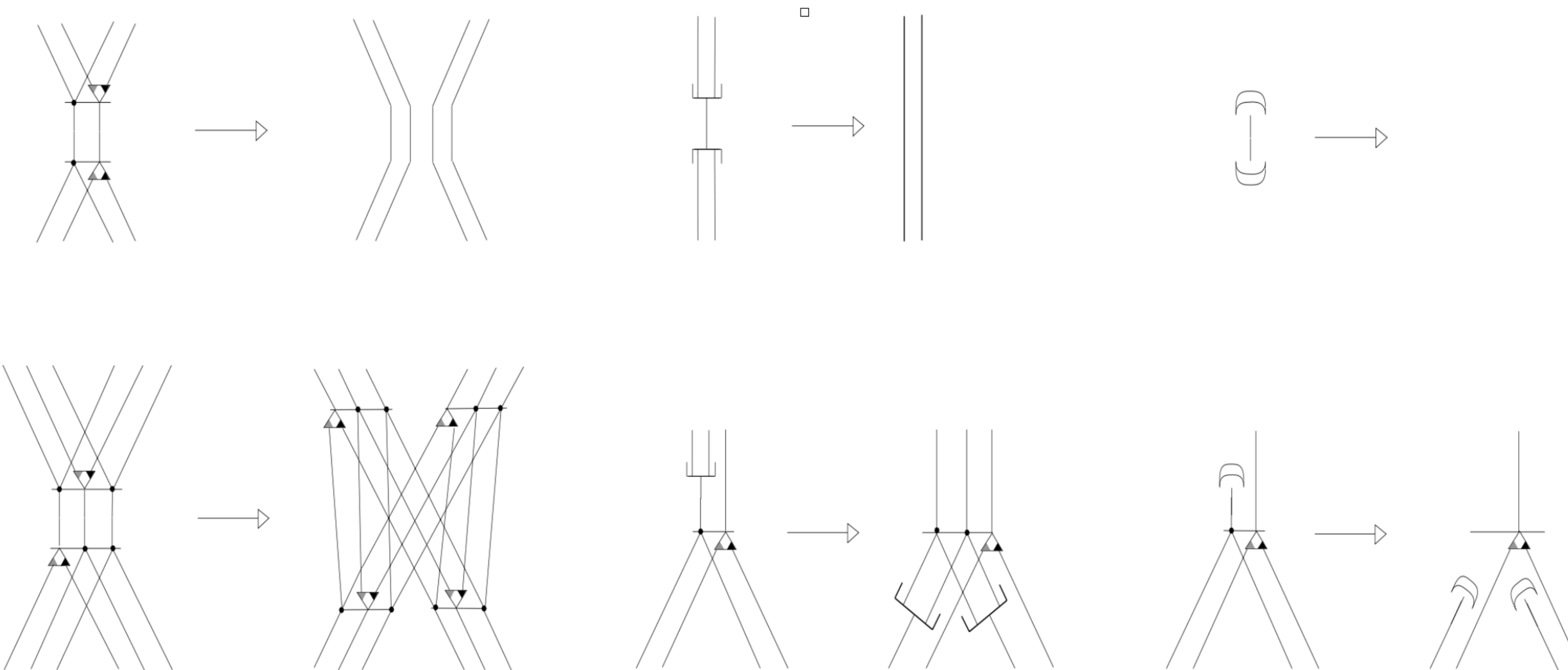


- coding of λ -expressions



Gonthier algorithm

- with the bus notation, interaction rules are “simpler” due to geometric properties



Gonthier algorithm

- **semantics** of the sharing nets

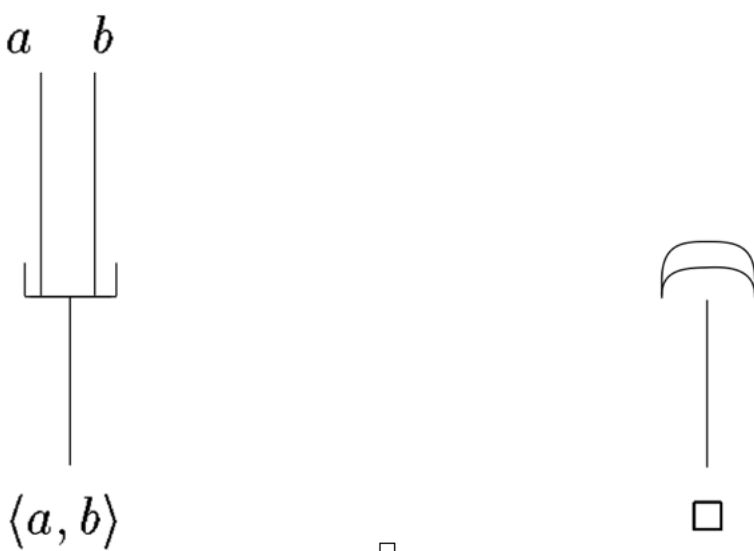
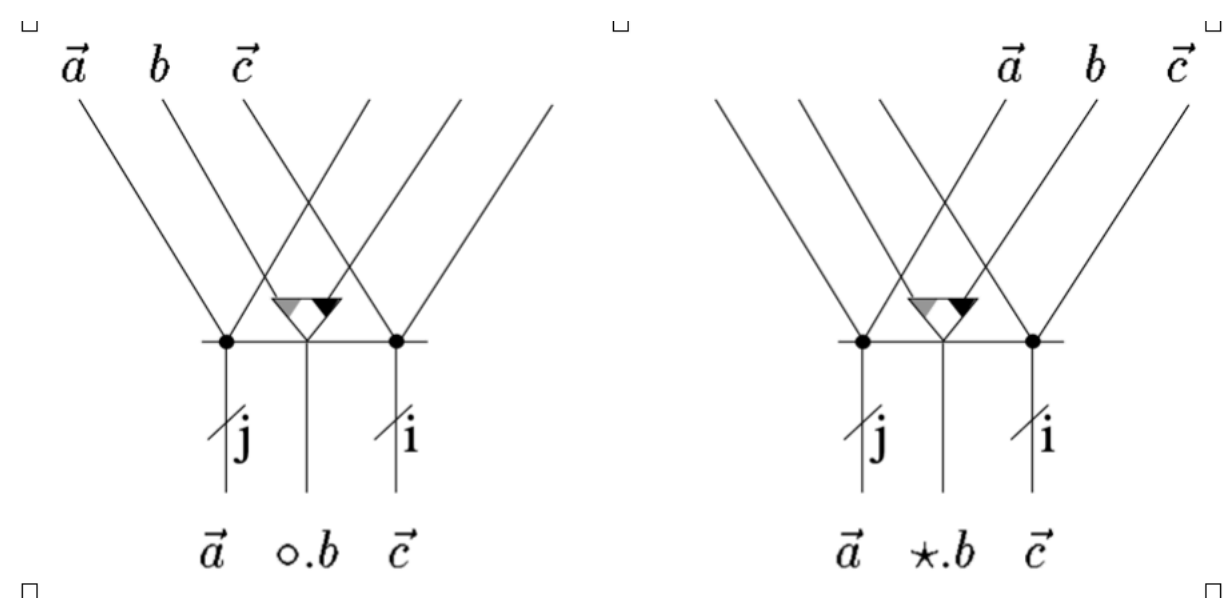
left wires (stack of contexts)

$$\langle \dots \langle \langle a_1, a_2 \rangle, a_3 \rangle \dots a_n \rangle$$

right wires (list of associations context - operators

$$\otimes_1 \cdot \langle a_2, \otimes_2 \rangle \cdot \langle a_3, \otimes_3 \rangle \dots \cdot \langle a_{n-1}, \otimes_{n-1} \rangle \cdot a_n$$

- transformation of the sharing contexts



➡ read-back procedure

the BOHM
machine

Alternatives coding

- Asperti, Danos, Laneve, Regnier investigated relations with linear logic
- Asperti et al developped the Bologna Higher Order Machine to implement Lamping style machine
- they use the (more natural) “call-by-need” coding $D = (!D) \multimap D$ instead of $D = !(D \multimap D)$

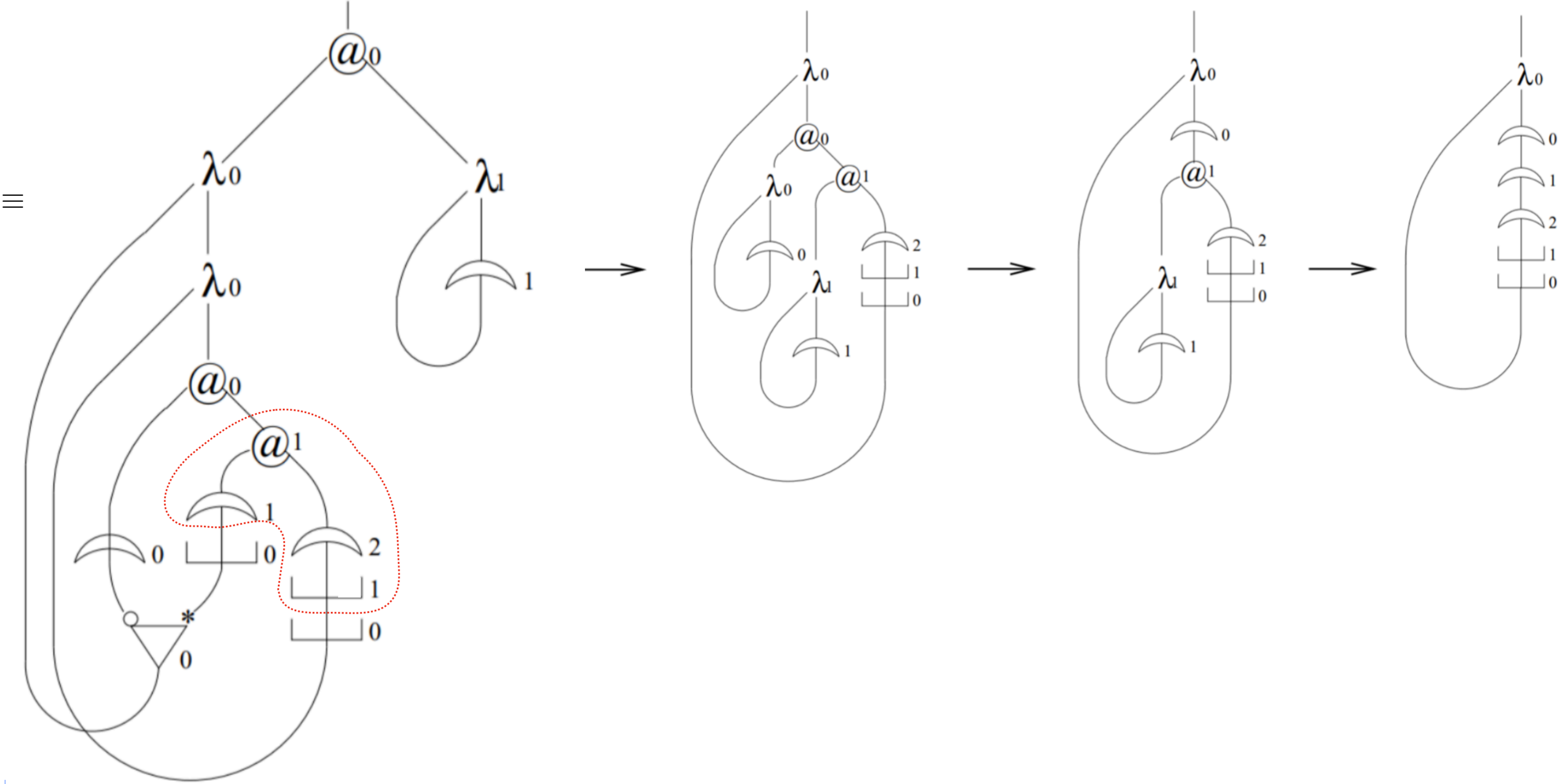
$$\llbracket x \rrbracket_n \equiv \text{arc}_n$$

$$\llbracket \lambda x. M \rrbracket_n \equiv \text{lambda node}$$

$$\llbracket M N \rrbracket_n \equiv \text{application node}$$

Example

$\llbracket (\lambda f. \lambda x. f(f\ x))\ I \rrbracket_0 \equiv$



Efficiency

- [Statman\[79\]](#) normalizing simply-typed λ -terms is not elementary recursive in size of term
- [Asperti-Mairson\[98\]](#) the number of simply-typed λ -terms parallel-steps is linear in size of term
- [Asperti-Coppola-Martini\[04\]](#) duplications are not Kalmar elementary recursive
- comments in interesting talk by [Laretto\[21\]](#) :
 - **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
- BOHM 1.0 and 1.1 is available on web at <https://github.com/asperti/BOHM1.1>
[BOHM language is PCF-style extension of untyped λ -calculus]

Conclusion

- this work established a **deeper** correspondence between λ -calculus and linear logic
- brave student should program the **GOHM** analogous to BOHM
- it works for linear logic without boxes
- and also in any calculi with **bound variables** (interaction systems)
- problem with **no difference** between typed and untyped calculi
- usual functional languages implement **weak reduction** where sharing is easier
- much of this work is due to the **clairvoyance** of Georges Gonthier



merci

Georges !