Reductions and Causality (1)

jean-jacques.levy@inria.fr
Escuela de Ciencias Informáticas
Universidad de Buenos Aires

July 22, 2013

http://jeanjacqueslevy.net/courses/13eci

Plan

- independent statements
- control flow
- relaxed control flow
- data dependencies
- parallel processes
- functional languages and multi-threading

Weak memory models

CENTRE DE RECHERCHE COMMUN

INRIA
MICROSOFT RESEARCH

Intel whitepaper (1/3)

2.3 Loads may be reordered with older stores to different locations

Intel 64 memory ordering allows load instructions to be reordered with prior stores to a different location. However, loads are not reordered with prior stores to the same location.

The first example in this section illustrates the case in which a load may be reordered with an older store – i.e. if the store and load are to different non-overlapping locations.

Processor 0	Processor 1
mov [_x], 1 // M1 mov r1, [_y] // M2	mov [_y], 1 // M3 mov r2, [_x] // M4
Initially x == y == 0 r1 == 0 and r2 == 0 is allowed	

Table 2.3.a: Loads may be reordered with older stores

Demo1

Intel whitepaper (2/3)

2.1 Loads are not reordered with other loads and stores are not reordered with other stores

Intel 64 memory ordering ensures that loads are seen in program order, and that stores are seen in program order.

Processor 0	Processor 1
mov [_x], 1 // M1 mov [_y], 1 // M2	mov r1,[_y] // M3 mov r2, [_x] // M4
Initially x == y == 0 r1 == 1 and r2 == 0 is not allowed	

Table 2.1: Stores are not reordered with other stores

Demo2

Intel whitepaper (3/3)

2.5 Stores are transitively visible

Intel 64 memory ordering ensures transitive visibility of stores – i.e. stores that are causally related appear to execute in an order consistent with the causal relation.

Processor 0	Processor 1	Processor 2
mov [_x], 1 // M1	mov r1, [_x] // M2 mov [_y], 1 // M3	mov r2, [_y] // M4 mov r3, [_x] // M5
Initially x == y == 0 r1 == 1, r2 == 1, r3 == 0 is not allowed		

Table 2.5: Stores are transitively visible

Demo3

Causality

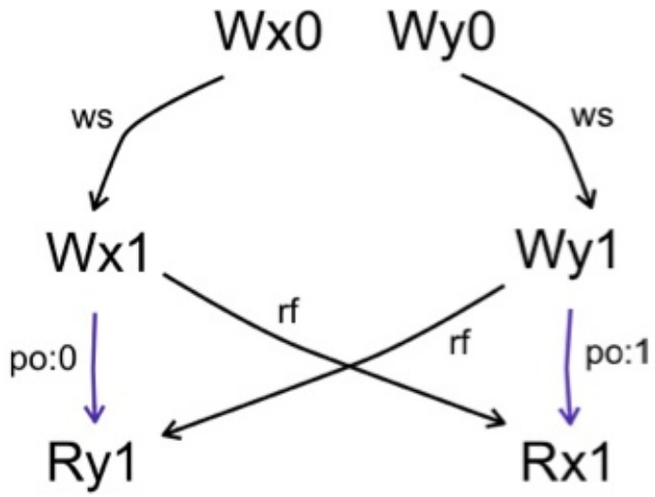
- Computations steps may be independent
- Others are causally related
- Refined to memory accesses
- Theory of causality ?

  - reductions steps in the λ-calculus (this course)
  - event structures (processes)
  - true concurrency
  - slicing (program analysis)
  - etc.

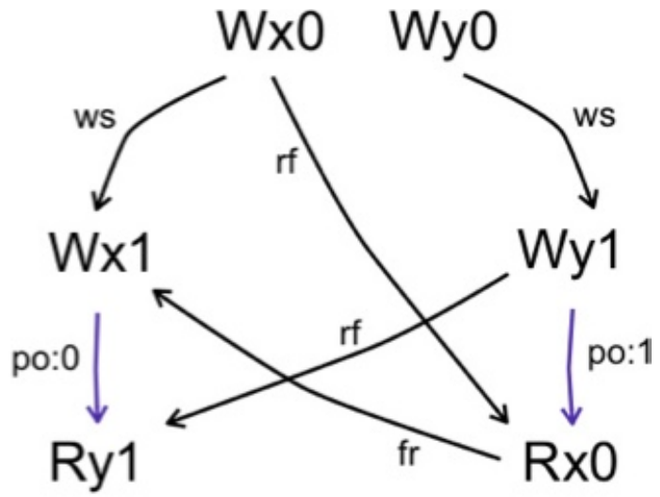
Intel 64 revisited (1/4)

- In SC (sequentially consistent), program order is strictly respected

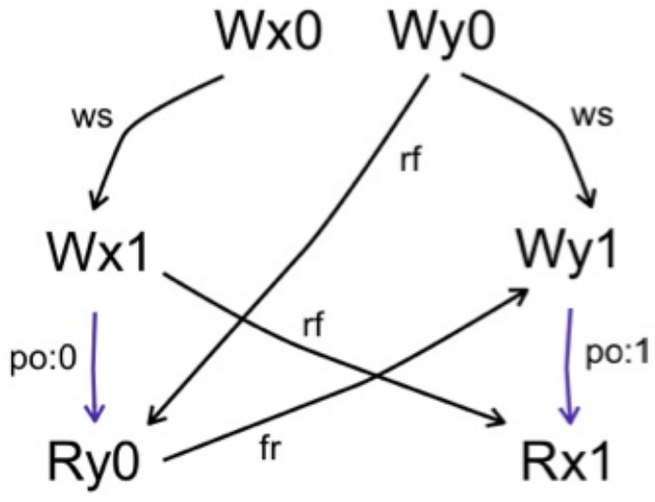
P0	P1
mov [x], 1	mov [y], 1
mov r1, [y]	mov r2, [x]



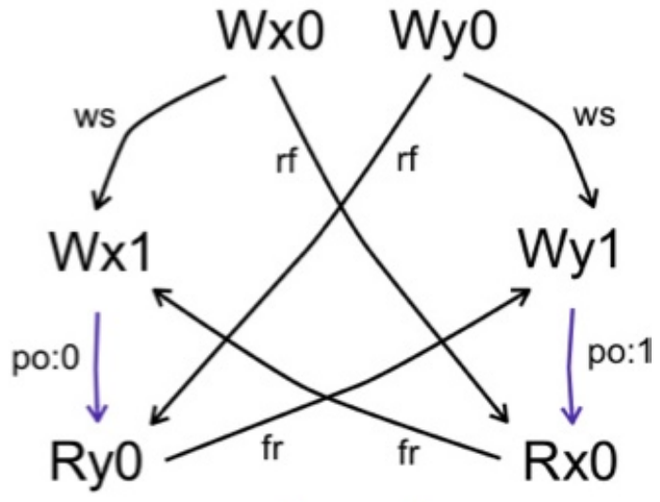
r1 = 1, r2 = 1



r1 = 1, r2 = 0



r1 = 0, r2 = 1



cyclic graph

Intel 64 revisited (2/4)

- Dependency relations

$A \xrightarrow{po} B$ is program order

$W \xrightarrow{ws} W$ is write serialization (acyclic) relation

$W \xrightarrow{rf} R$ is read from relation

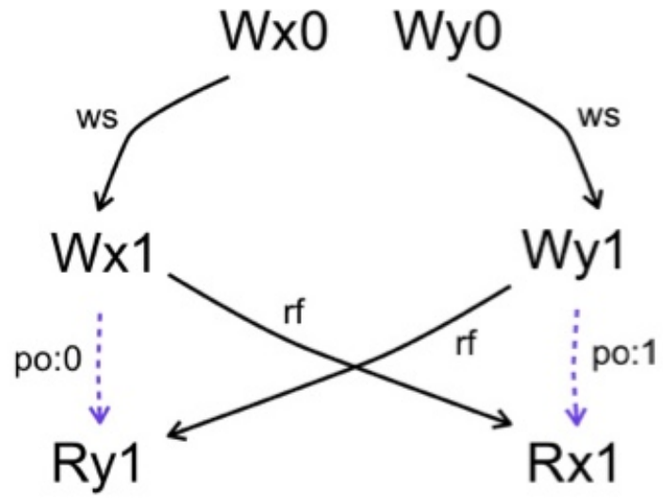
$R \xrightarrow{fr} W$ is from read relation when there is W' such that

$W' \xrightarrow{rf} R$ and $W' \xrightarrow{ws} W$

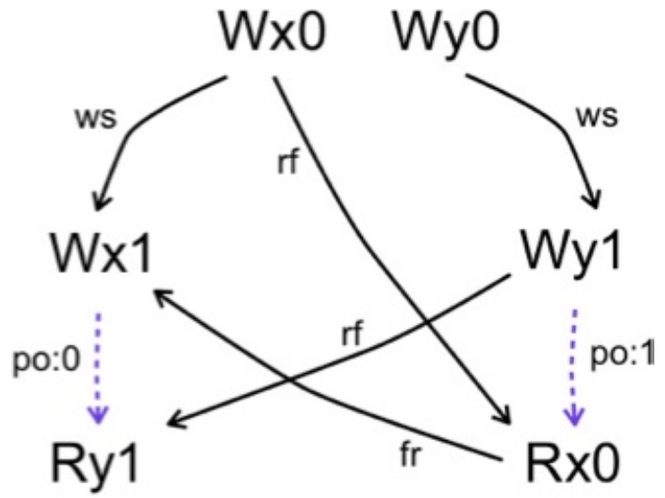
Intel 64 revisited (3/4)

- In TSO, *W* followed by *R* can be relaxed within program order

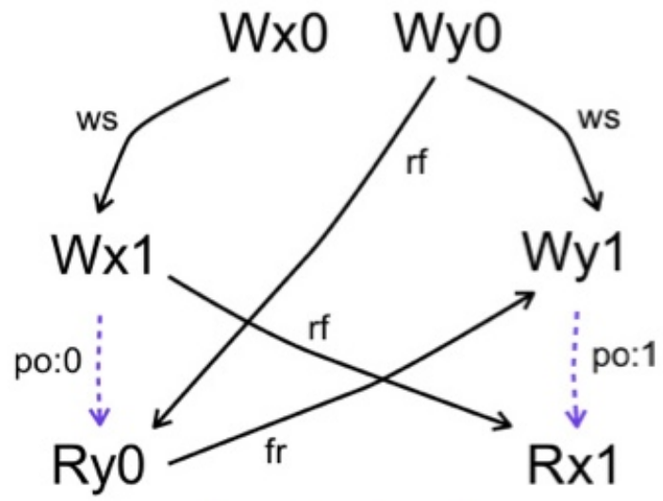
P0	P1
mov [x], 1	mov [y], 1
mov r1, [y]	mov r2, [x]



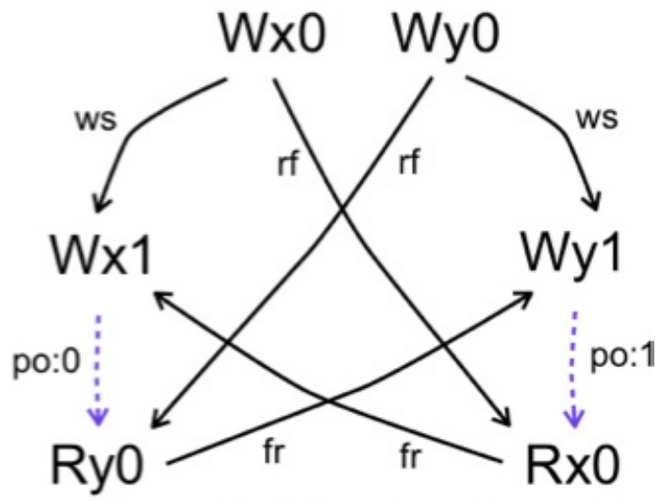
r1 = 1, r2 = 1



r1 = 1, r2 = 0



r1 = 0, r2 = 1

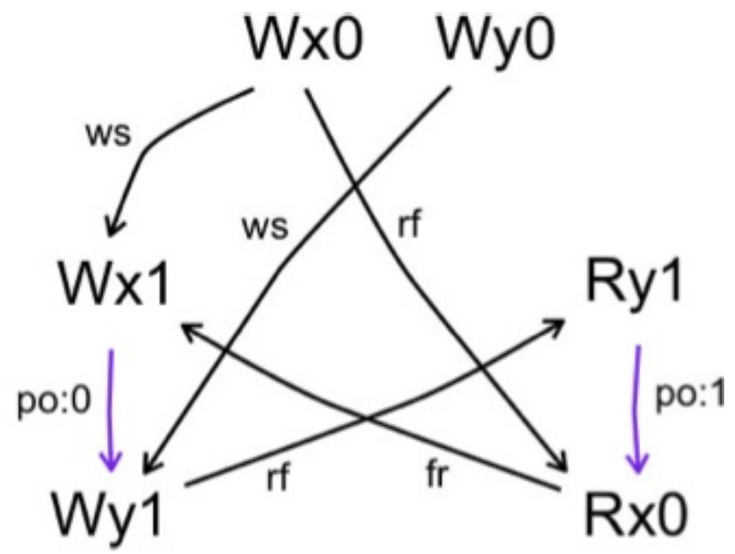


r1 = 0, r2 = 0

Intel 64 revisited (4/4)

- In TSO, *W* followed by *R* is relaxed

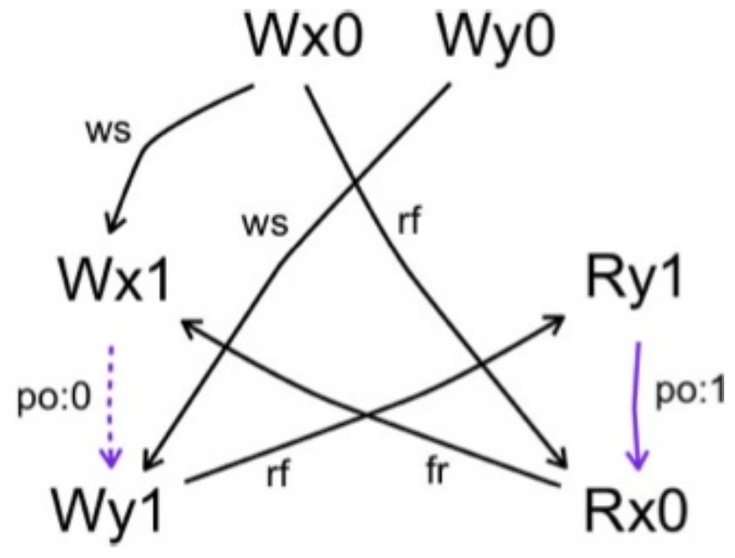
P0	P1
mov [x], 1	mov r1, [y]
mov [y], 1	mov r2, [x]



r1 = 1, r2 = 0 Cyclic graph

- In PSO, *W* followed by *W* to distinct location is relaxed

P0	P1
mov [x], 1	mov r1, [y]
mov [y], 1	mov r2, [x]



r1 = 1, r2 = 0

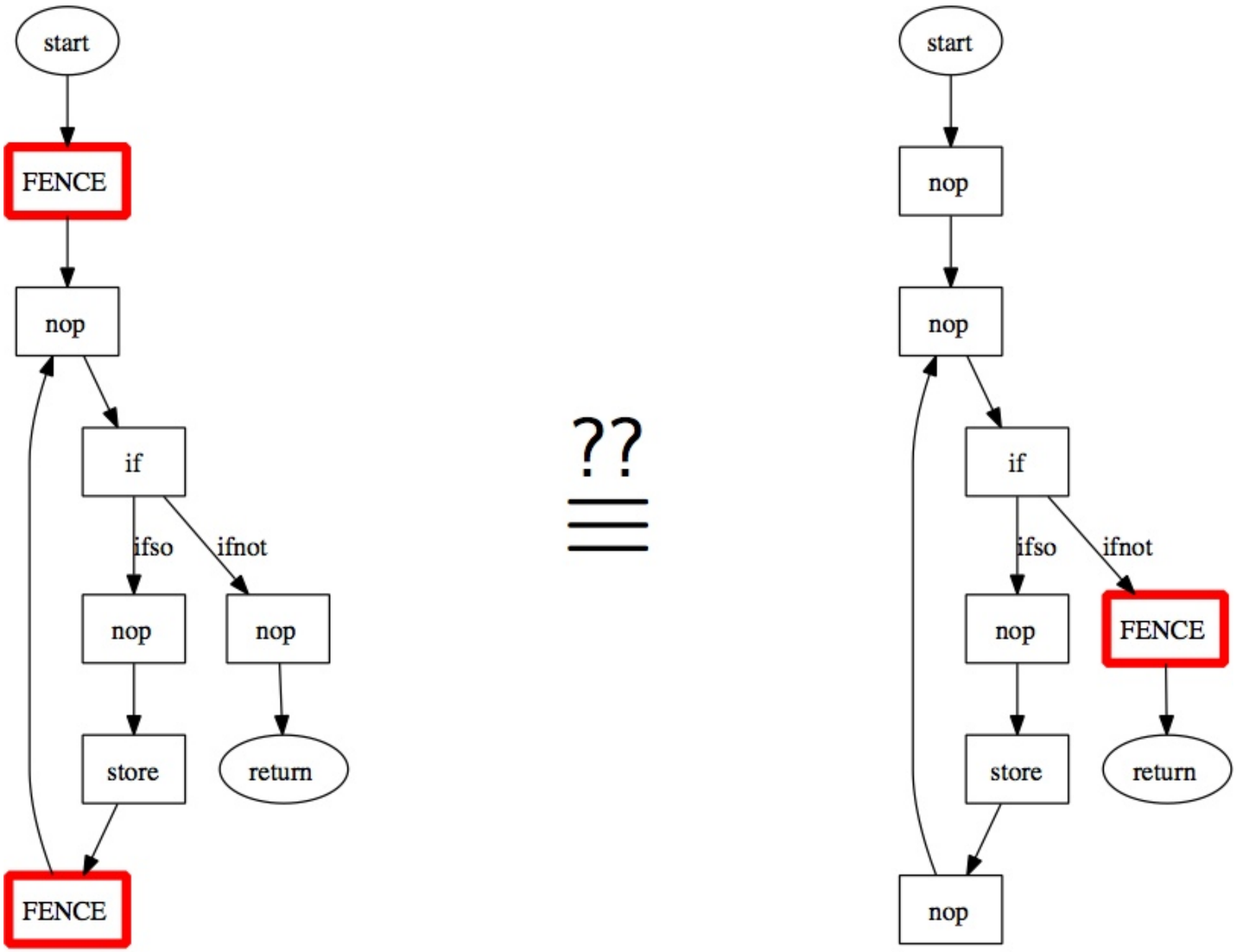
WMM

- axiomatic + operational models for Intel [~Cambridge] / Power [~INRIA]
- formalisation in HOL/Coq
- tests on real processor behaviour
  http://www.cl.cam.ac.uk/~pes20/ppc-supplemental

- formal proof of simple concurrent code (eg. Linux spinlocks)
- operational reasoning: data-race freedom, separation logic
- certified compiler for concurrent languages
  http://www.cl.cam.ac.uk/~pes20/CompCertTSO

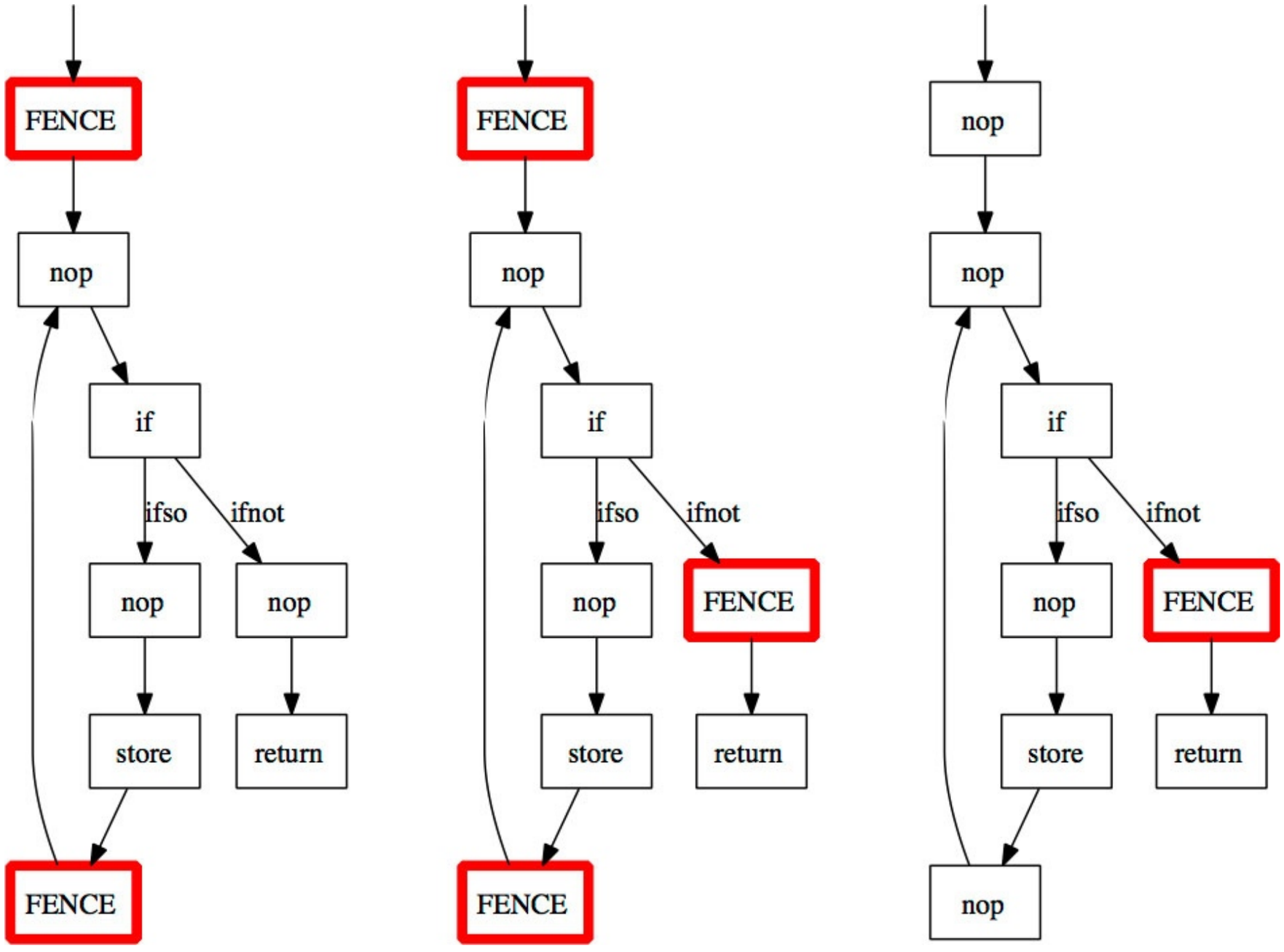
[Zappa Nardelli, Maranget, Alglave, Braibant, Sewell et al]
[POPL 09, CACM 10; DAMP 09, CAV 10, PLDI 11; TACAS 11; POPL 11]

WMM and optimization (1/2)



Fences elimination with TSO $\simeq$ 3 kloCoq

WMM and optimization (2/2)



[Vafeiadis, Zappa Nardelli] SAS/2010

PCF

(a small untyped functional language)

[Plotkin 74]

CENTRE DE RECHERCHE COMMUN
INRIA
MICROSOFT RESEARCH

PCF language

- Terms

M, N, P	::=	*x, y, z, ...*	(variables)
	|	$\lambda x.M$	(M as function of x)
	|	*M (N)*	(M applied to N)
	|	`n`	(natural integer constant)
	|	$M \otimes N$	(arithmetic op)
	|	`ifz` *P* `then` *M* `else` *N*	(conditionnal)
	|	*Y*	(recursion)

- Calculations ("reductions")

$$(\lambda x.M)(N) \longrightarrow M\{x := N\}$$

$$\underline{m} \otimes \underline{n} \longrightarrow \underline{m \otimes n}$$

$$\texttt{ifz } \underline{0} \texttt{ then } M \texttt{ else } N \longrightarrow M$$

$$\texttt{ifz } \underline{n+1} \texttt{ then } M \texttt{ else } N \longrightarrow N$$

$$Yf \longrightarrow f(Yf)$$

Examples of PCF terms (1/2)

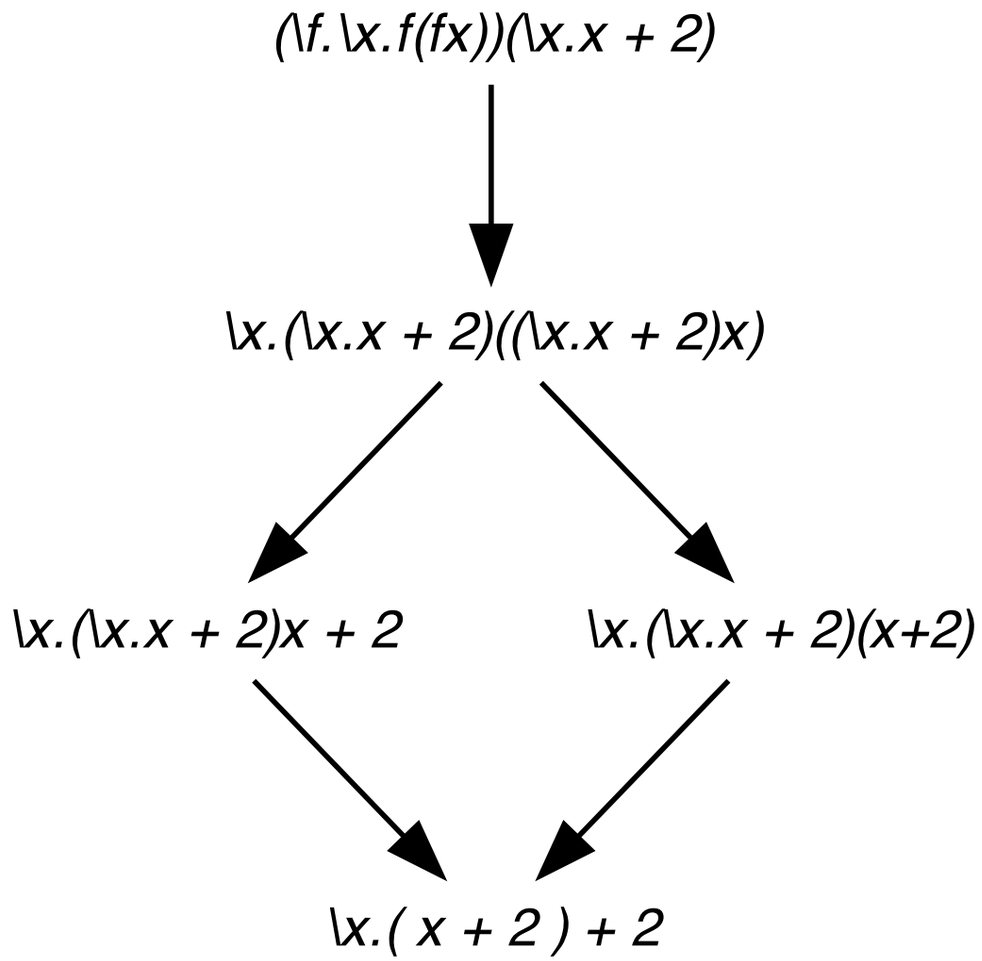
$(\lambda x.\, x + 1)3 \longrightarrow 3 + 1 \longrightarrow 4$

$(\lambda x.\, 2 * x + 2)4 \longrightarrow 2 * 4 + 2 \longrightarrow 8 + 2 \longrightarrow 10$

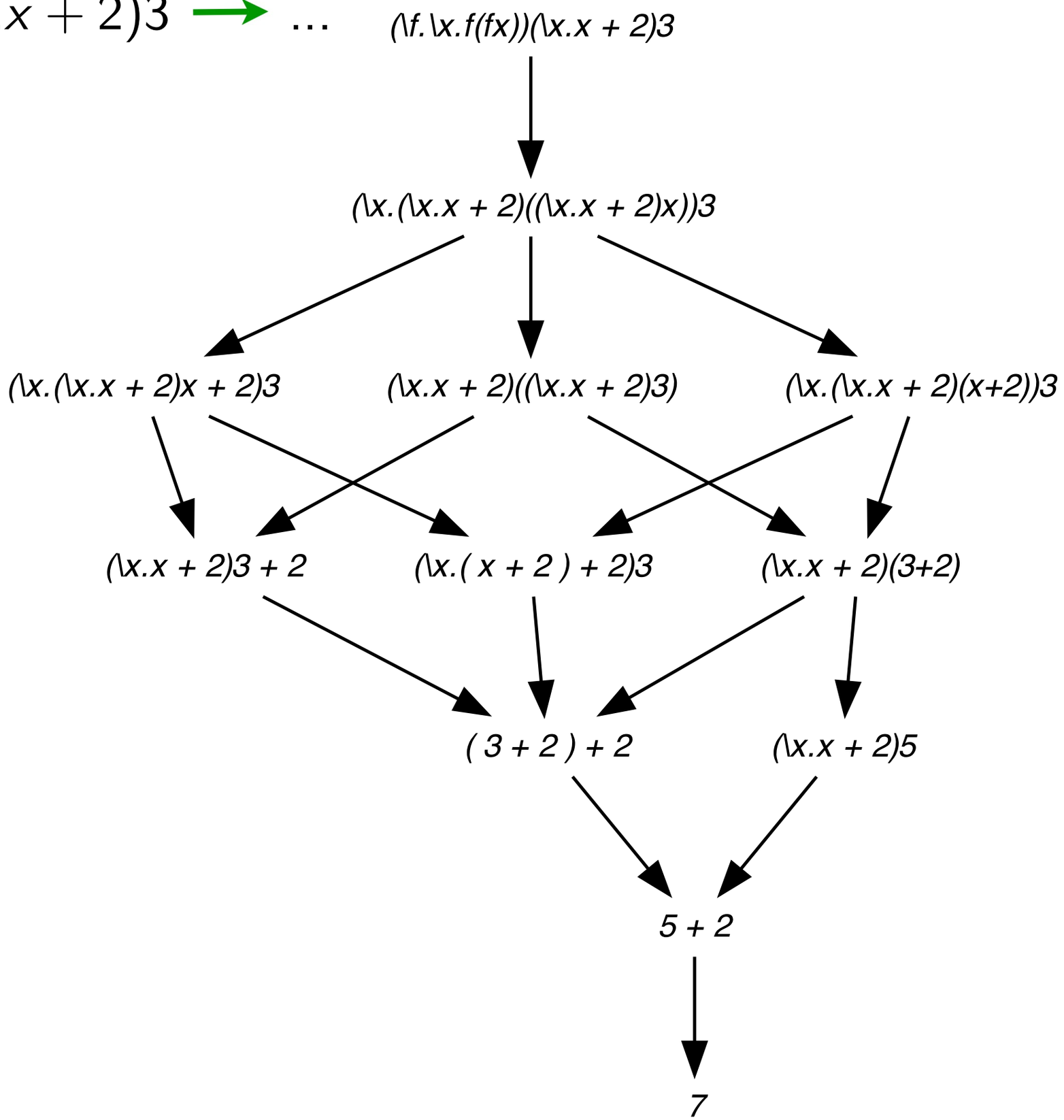
$(\lambda f.f3)(\lambda x.\, x + 2) \longrightarrow (\lambda x.\, x + 2)3 \longrightarrow 3 + 2 \longrightarrow 5$

$(\lambda f.\lambda x.f(f\, x))(\lambda x.x + 2) \longrightarrow \ldots$

$$(\lambda f.\lambda x.f(f\ x))(\lambda x.\ x + 2) \longrightarrow \ldots$$



$(\lambda f.\lambda x.f(f\ x))(\lambda x.x+2)3 \rightarrow \ldots$



Examples of PCF terms (2/2)

$\texttt{Fact}(3)$

$\texttt{Fact} = Y(\lambda f.\lambda x.\ \texttt{ifz}\ x\ \texttt{then}\ 1\ \texttt{else}\ x \star f(x-1))$

Thus following term:

$(\lambda\, \texttt{Fact}\, .\, \texttt{Fact}(3))$

$(Y(\lambda f.\lambda x.\ \texttt{ifz}\ x\ \texttt{then}\ 1\ \texttt{else}\ x \star f(x-1)))$

also written

$(\lambda\, \texttt{Fact}\, .\, \texttt{Fact}(3))$

$(\ (\lambda Y.Y(\lambda f.\lambda x.\ \texttt{ifz}\ x\ \texttt{then}\ 1\ \texttt{else}\ x \star f(x-1)))$

$\quad (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))\ )$

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))

↓

(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3

↓

(\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1))3

↓

(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))3

↓

(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))3

↓

(\x.ifz x then 1 else x * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))3

↓

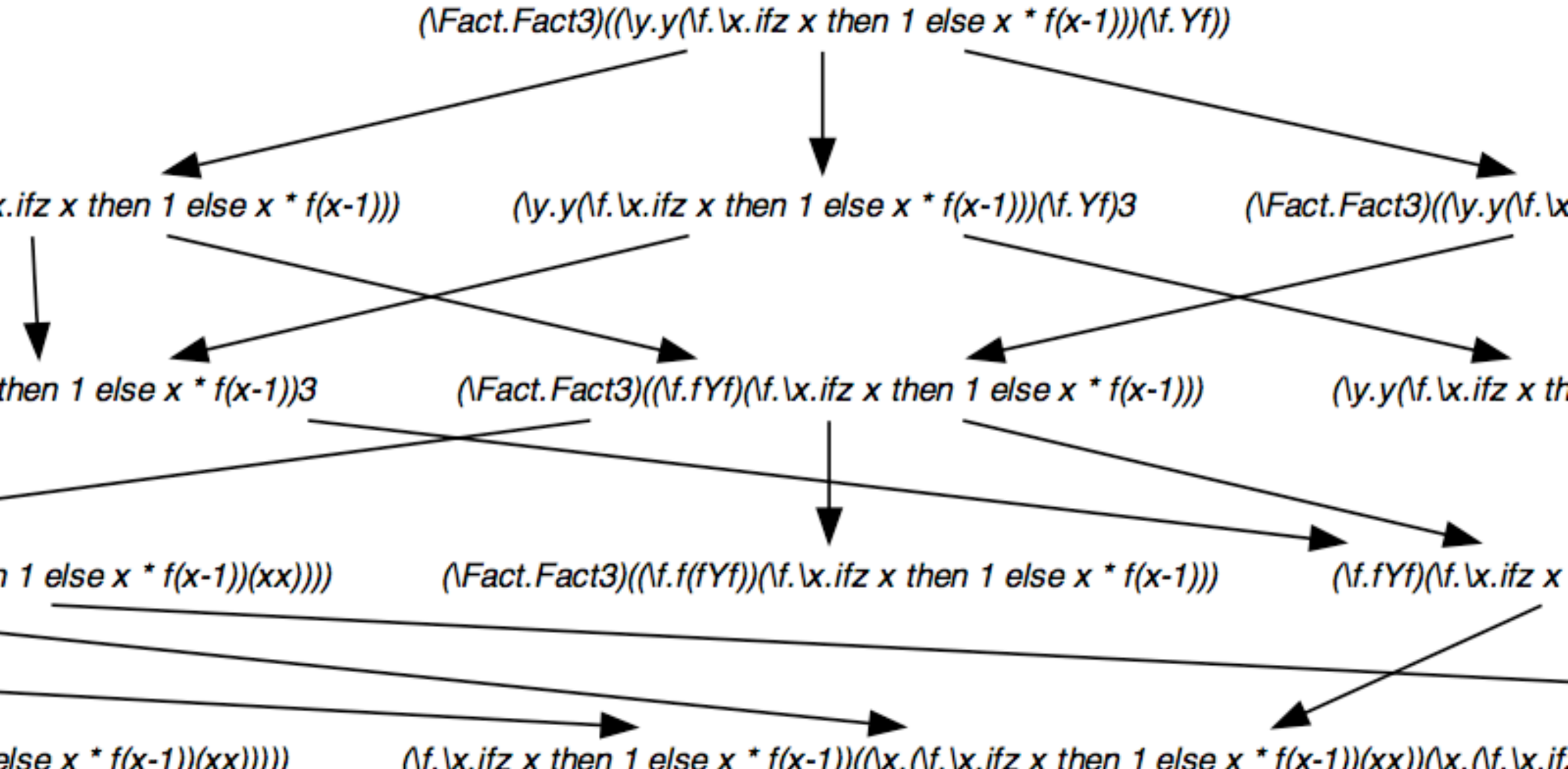
ifz 3 then 1 else 3 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(3-1)

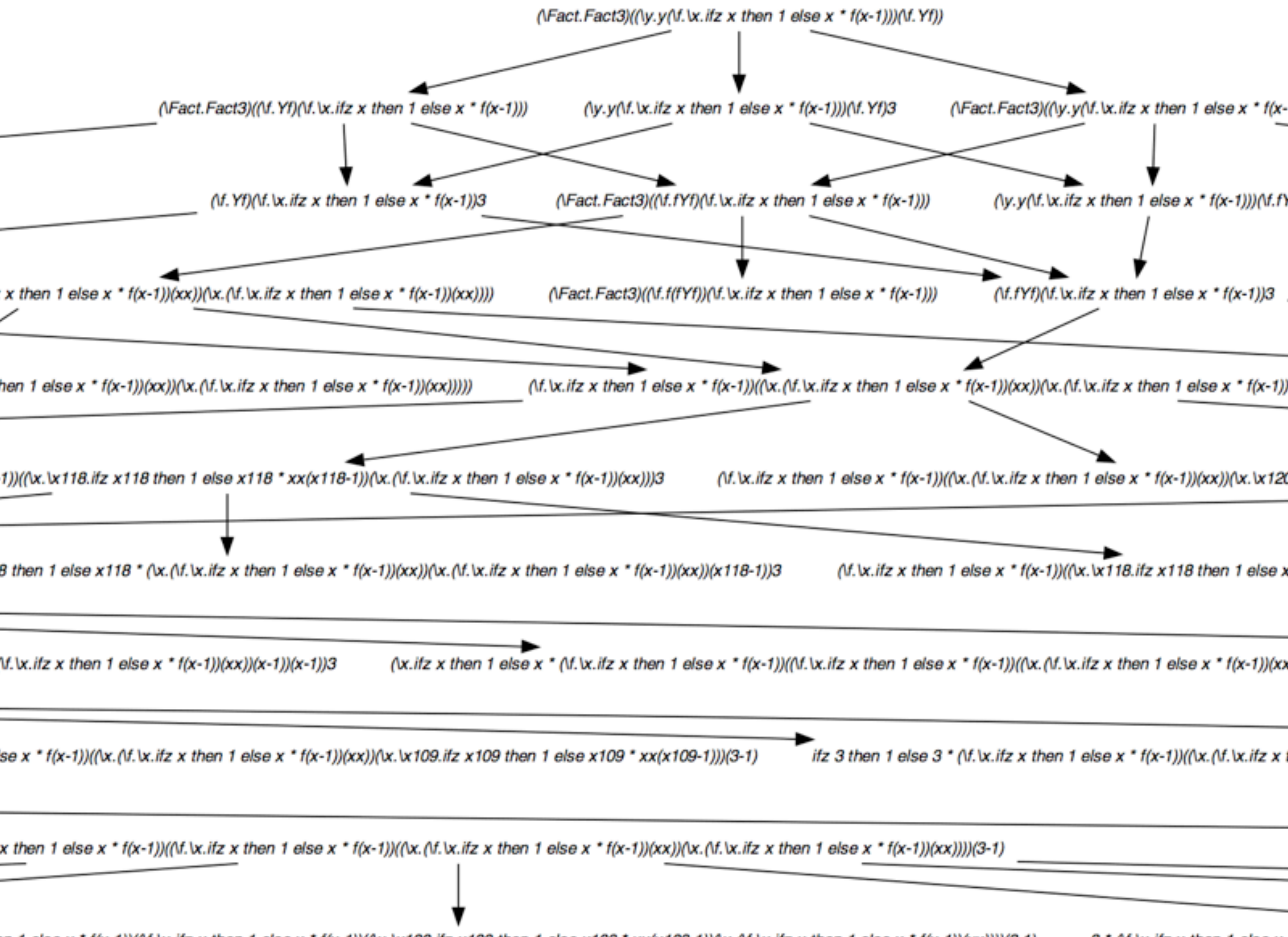
↓

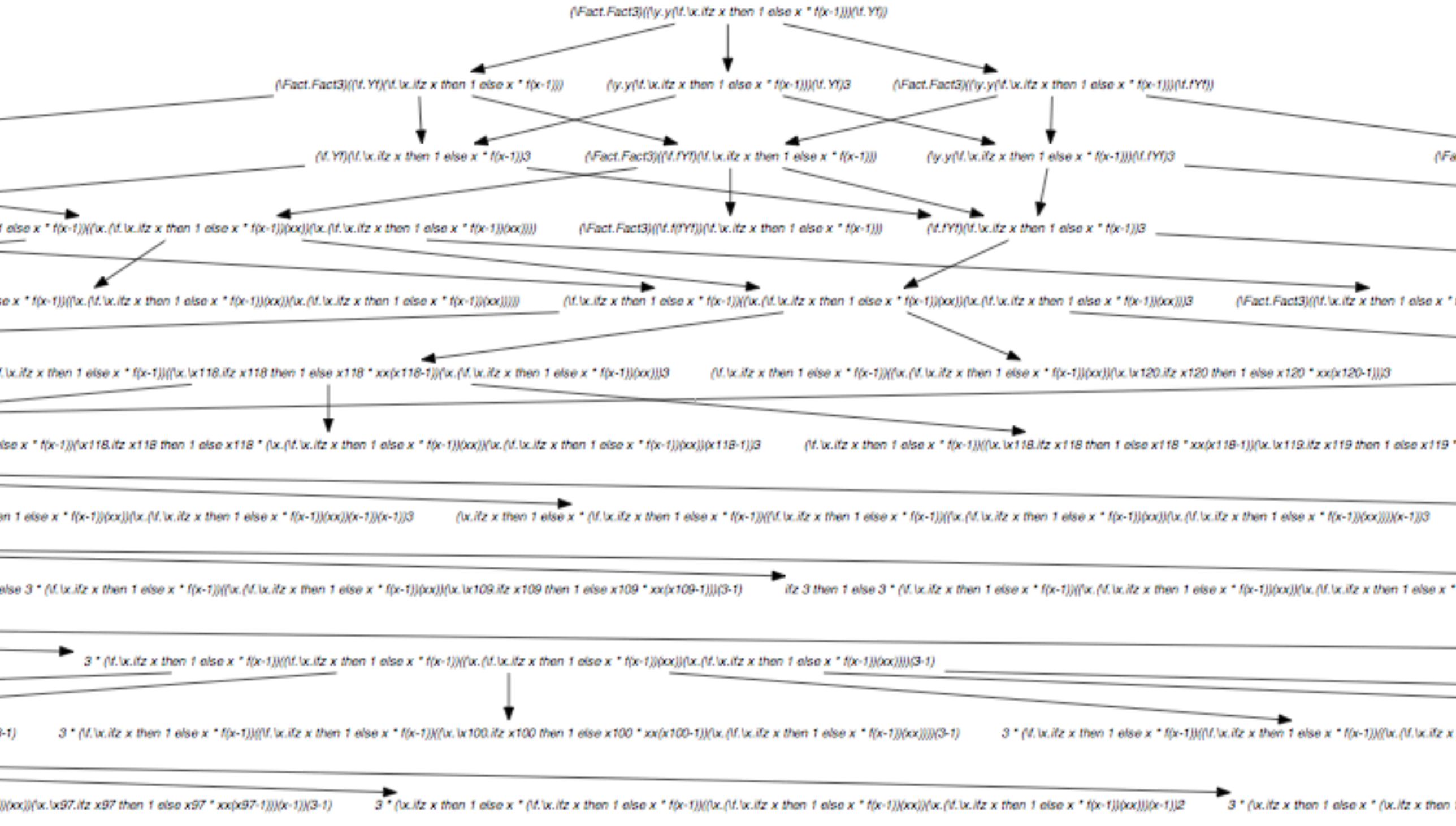
3 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(3-1)

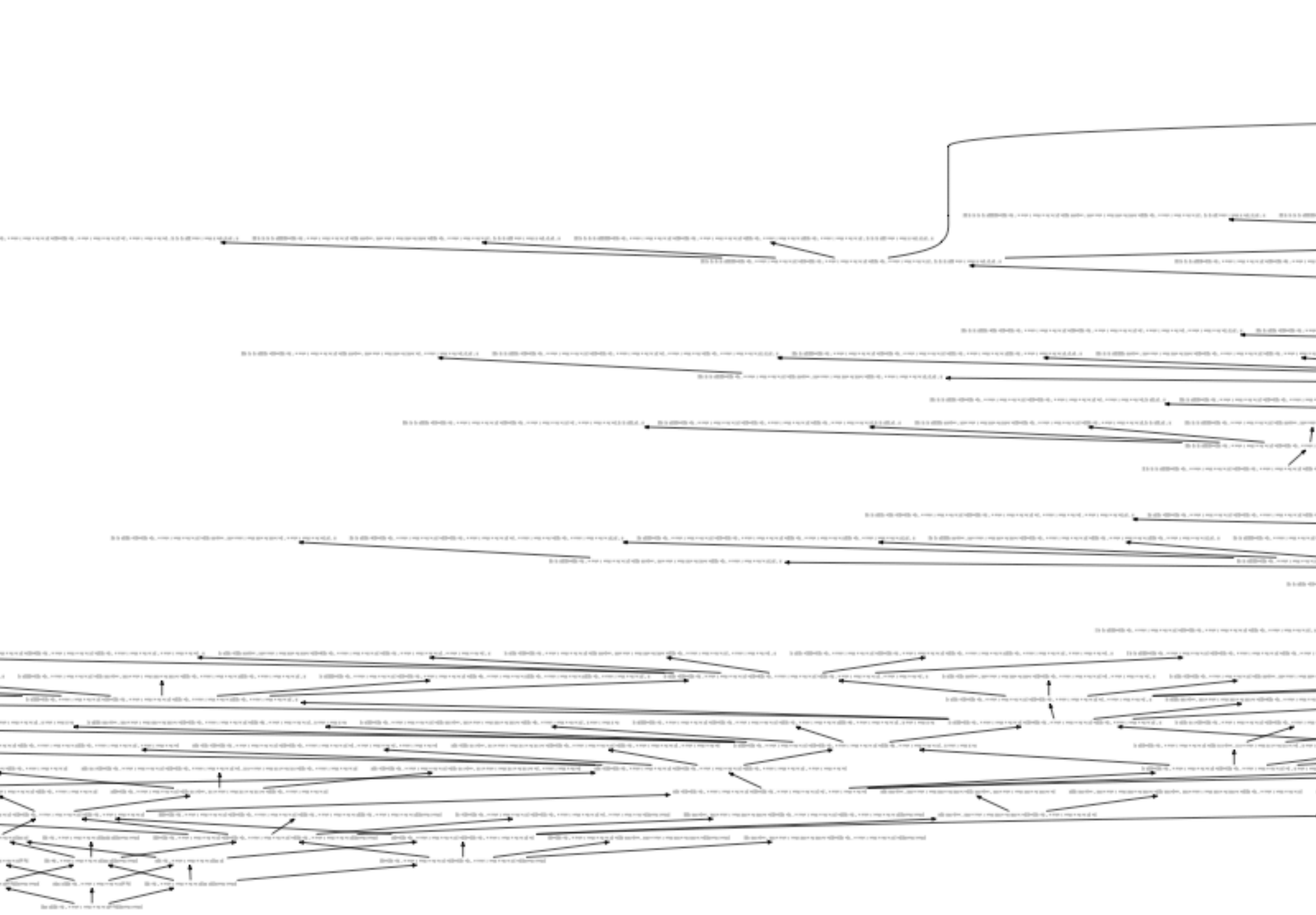
↓

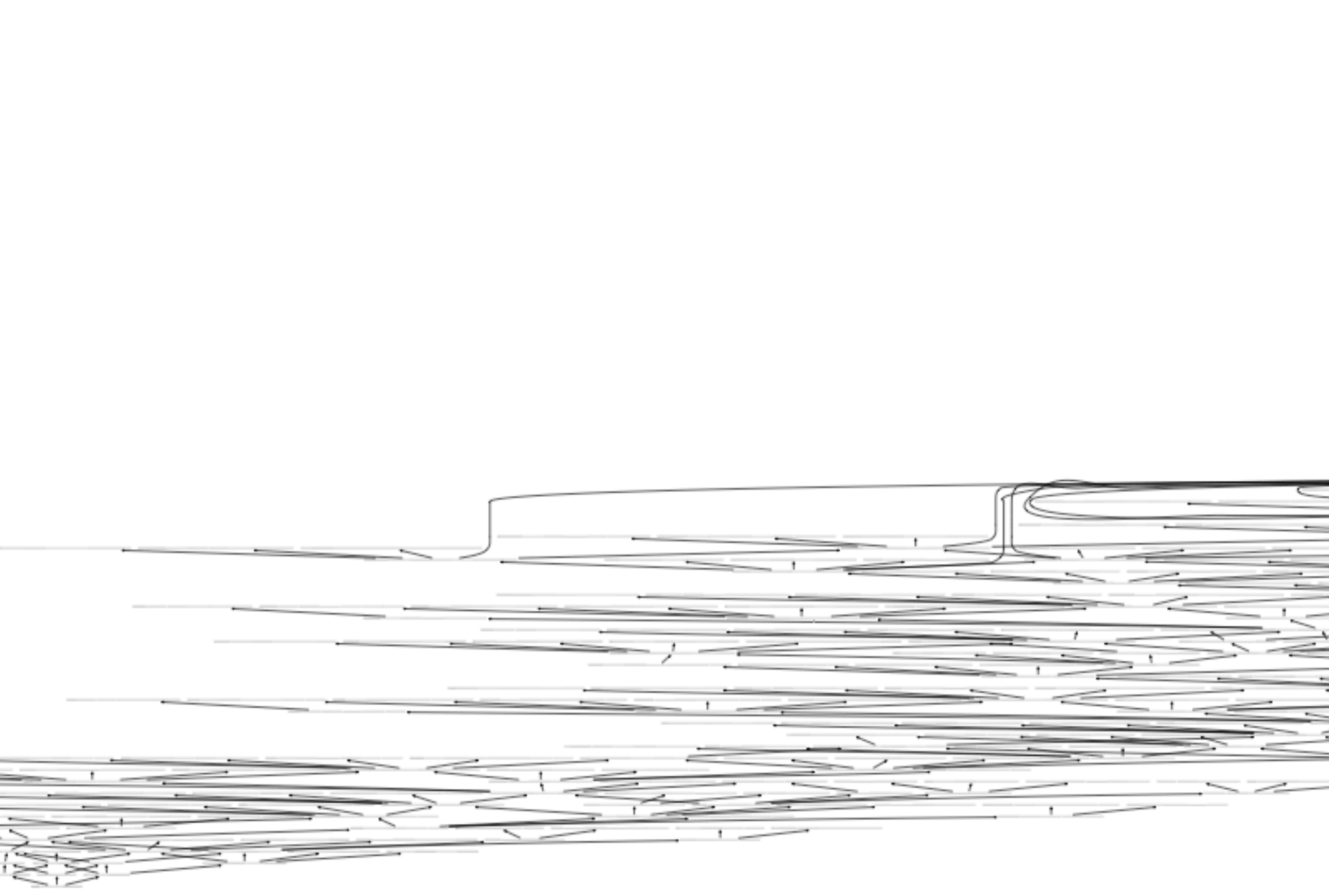
3 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(3-1)

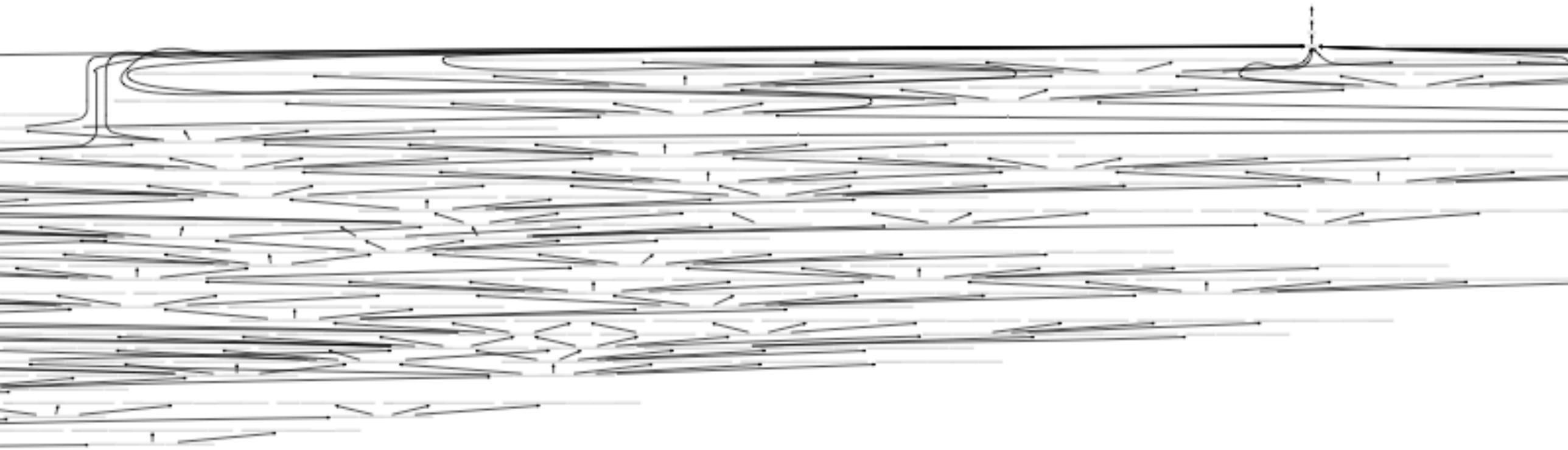

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x
then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x th
n 1 else x * f(x-1))(xx))))
(\Fact.Fact3)((\f.f(fYf))(\f.\x.ifz x then 1 else x * f(x-1)))
(\f.fYf)(\f.\x.ifz x
else x * f(x-1))(xx)))))
(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.if

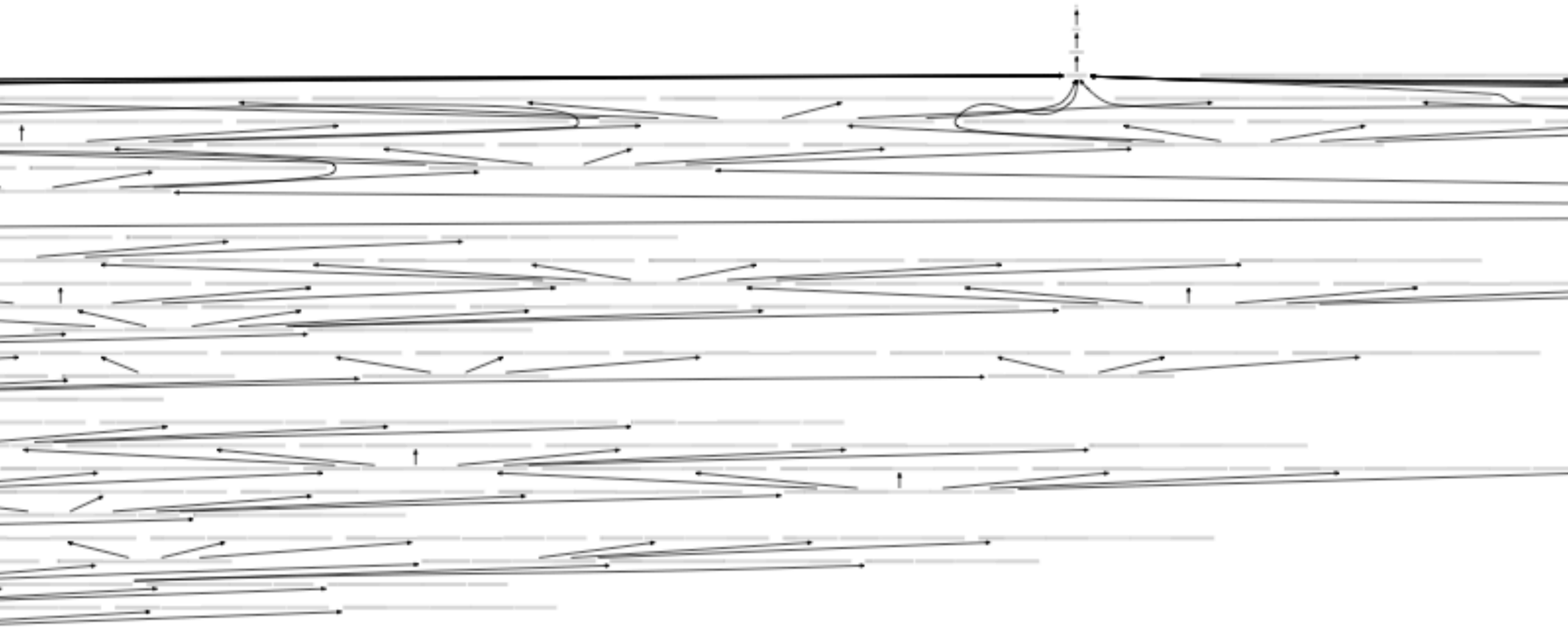

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
(\Fact.Fact3)((\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))
(\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.fY
x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))))
(\Fact.Fact3)((\f.f(fYf))(\f.\x.ifz x then 1 else x * f(x-1)))
(\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1))3
hen 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))))
(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))
1))((\x.\x118.ifz x118 then 1 else x118 * xx(x118-1))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))3
(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.\x120
8 then 1 else x118 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x118-1))3
(\f.\x.ifz x then 1 else x * f(x-1))((\x.\x118.ifz x118 then 1 else x
\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))(x-1))3
(\x.ifz x then 1 else x * (\f.\x.ifz x then 1 else x * f(x-1))((\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx
se x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.\x109.ifz x109 then 1 else x109 * xx(x109-1)))(3-1)
ifz 3 then 1 else 3 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x
x then 1 else x * f(x-1))((\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))))(3-1)

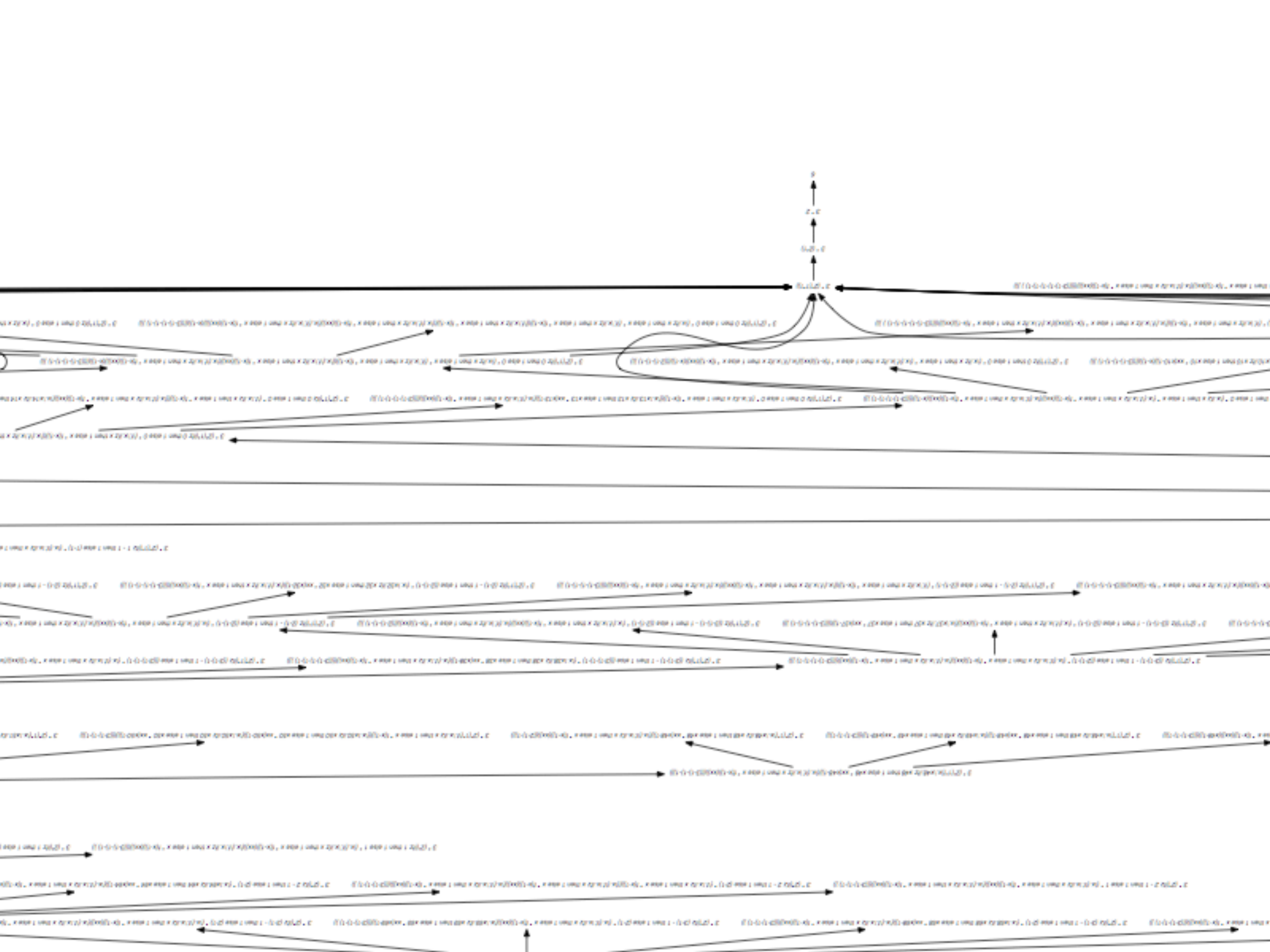

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
(\Fact.Fact3)((\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.fYf))
(\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.fYf)3
(\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.f(fYf))(\f.\x.ifz x then 1 else x * f(x-1)))
(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))3
(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))3
(\f.\x.ifz x then 1 else x * f(x-1))((\x.\x118.ifz x118 then 1 else x118 * xx(x118-1))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))3
(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\x120.ifz x120 then 1 else x120 * xx(x120-1)))3
(\f.\x.ifz x then 1 else x * f(x-1))(\x118.ifz x118 then 1 else x118 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x118-1))3
(\f.\x.ifz x then 1 else x * f(x-1))((\x.\x118.ifz x118 then 1 else x118 * xx(x118-1))(\x.\x119.ifz x119 then 1 else x119 *
(\x.ifz x then 1 else x * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(x-1))3
(\x.ifz x then 1 else x * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(x-1))3
ifz 3 then 1 else 3 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x *
3 * (\f.\x.ifz x then 1 else x * f(x-1))((\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(3-1)
3 * (\f.\x.ifz x then 1 else x * f(x-1))((\f.\x.ifz x then 1 else x * f(x-1))((\x.\x100.ifz x100 then 1 else x100 * xx(x100-1))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(3-1)
3 * (\f.\x.ifz x then 1 else x * f(x-1))((\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x
3 * (\x.ifz x then 1 else x * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(x-1))2
3 * (\x.ifz x then 1 else x * (\x.ifz x then

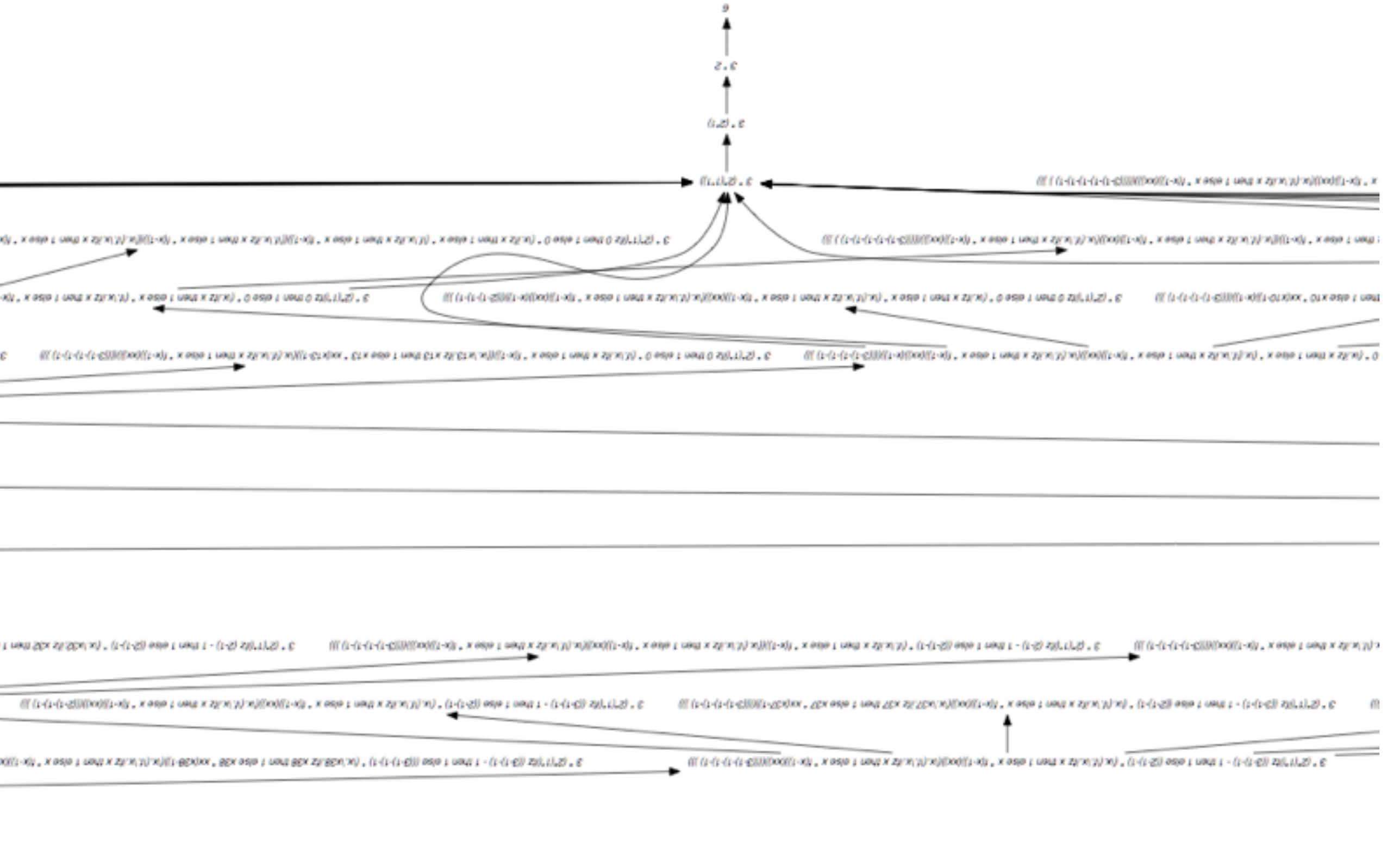


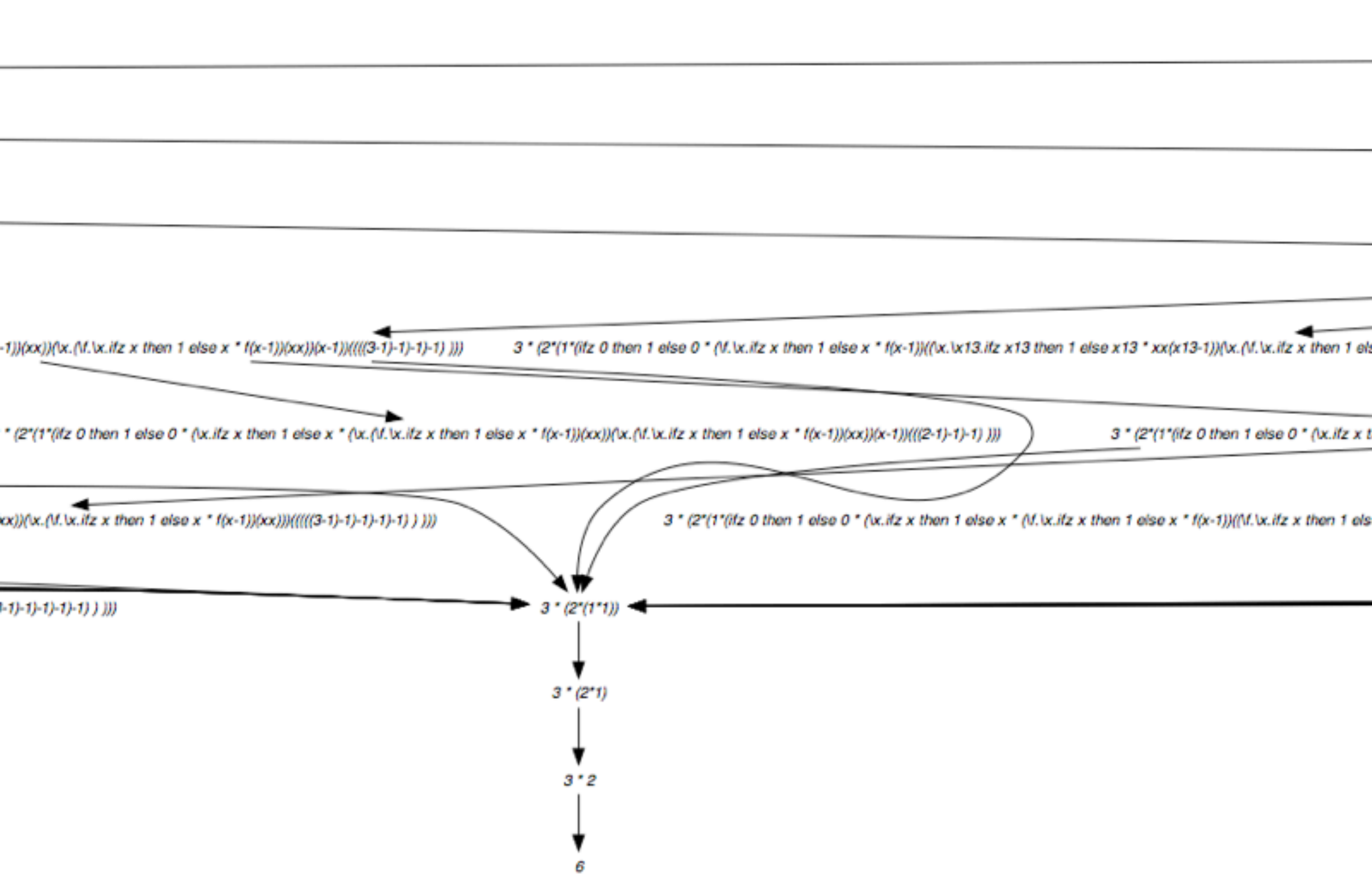


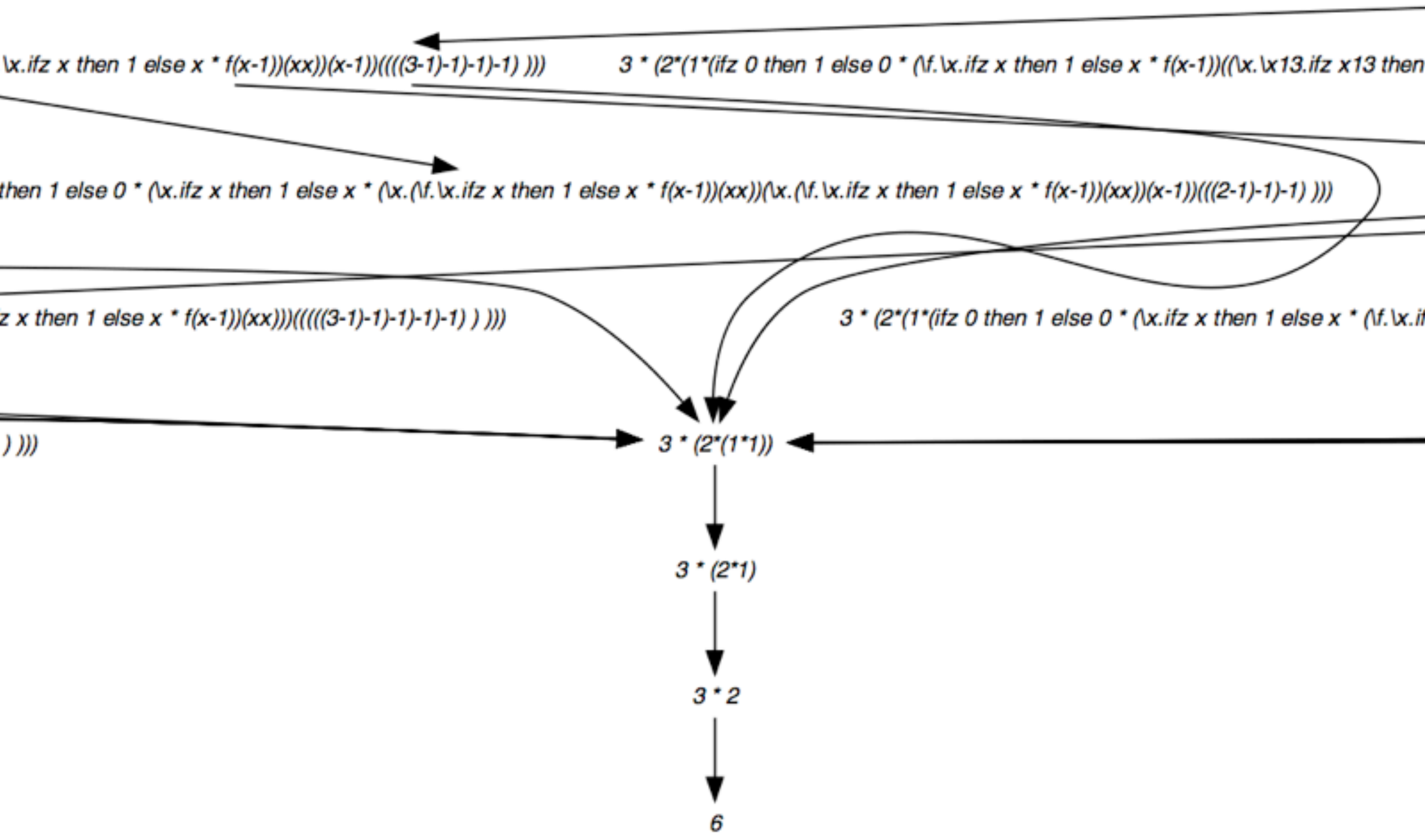









-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))((((3-1)-1)-1)-1)))) 3 * (2*(1*(ifz 0 then 1 else 0 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.\x13.ifz x13 then 1 else x13 * xx(x13-1))(\x.(\f.\x.ifz x then 1 els
* (2*(1*(ifz 0 then 1 else 0 * (\x.ifz x then 1 else x * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))(((2-1)-1)-1)))) 3 * (2*(1*(ifz 0 then 1 else 0 * (\x.ifz x t
xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))((((((3-1)-1)-1)-1)-1))))) 3 * (2*(1*(ifz 0 then 1 else 0 * (\x.ifz x then 1 else x * (\f.\x.ifz x then 1 else x * f(x-1))((\f.\x.ifz x then 1 els
-1)-1)-1)-1)-1)))))
3 * (2*(1*1))
3 * (2*1)
3 * 2
6

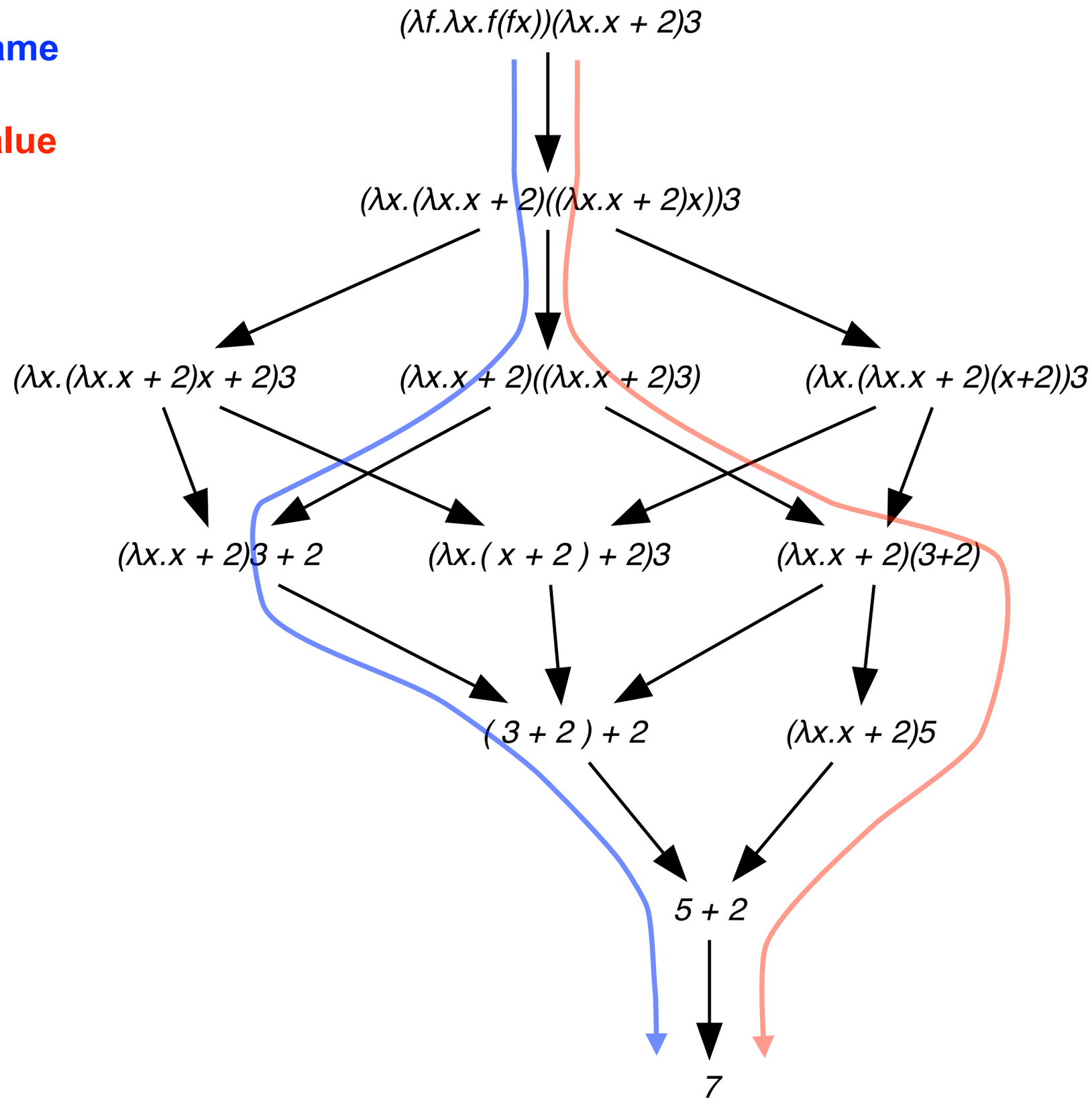

\x.ifz x then 1 else x * f(x-1))(xx))(x-1))((((3-1)-1)-1)-1))))
3 * (2*(1*(ifz 0 then 1 else 0 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.\x13.ifz x13 then
then 1 else 0 * (\x.ifz x then 1 else x * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))(((2-1)-1)-1))))
z x then 1 else x * f(x-1))(xx)))(((((3-1)-1)-1)-1)-1)))))
3 * (2*(1*(ifz 0 then 1 else 0 * (\x.ifz x then 1 else x * (\f.\x.if
))))
3 * (2*(1*1))
3 * (2*1)
3 * 2
6

Conclusions

- many possible calculations
- unicity of result ? (confluence -- Church Rosser)
- several reduction strategies:
  - **Call by name** (contracts leftmost-outermost redex)
  - **Call by value** (first computes value of function arguments)
  - **Call by need** (CBN variant with sharing)

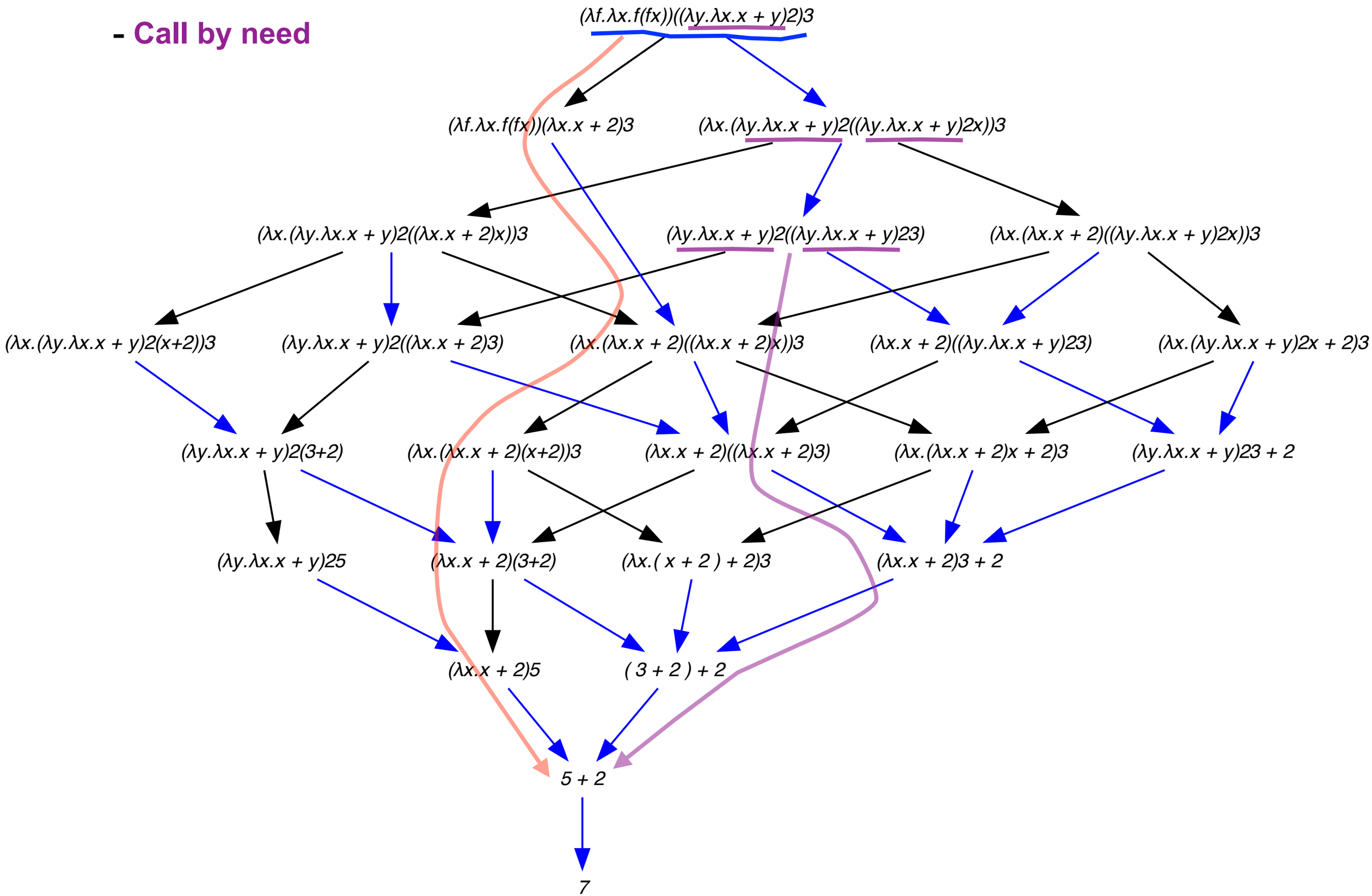
- **Call by name**

- **Call by value**



$(\lambda f.\lambda x.f(f\ x))((\lambda y.\lambda x.x + y)2)3 \rightarrow \ldots$

- Call by need



λ-calculus

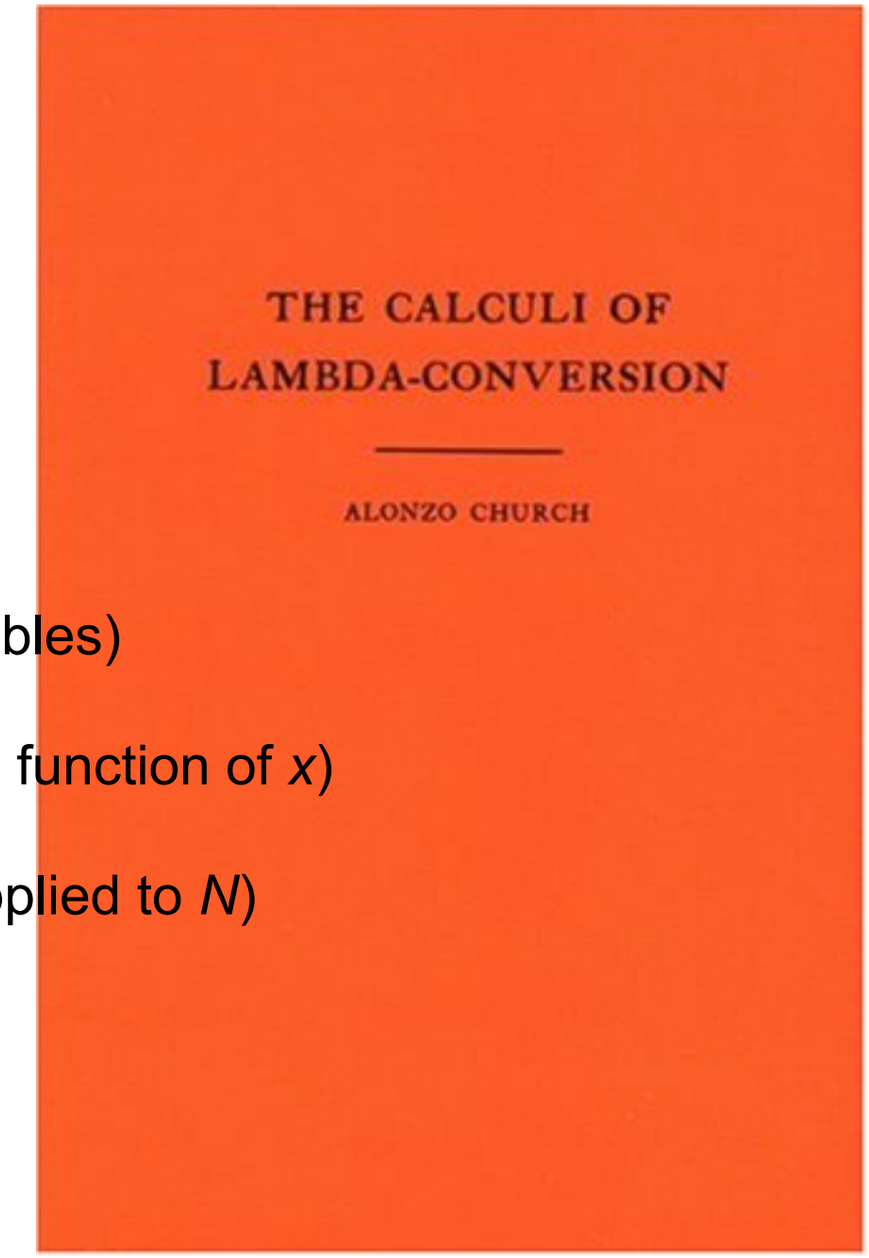
CENTRE DE RECHERCHE COMMUN

INRIA
MICROSOFT RESEARCH

The language

- lambda-terms

M, N, P	*::=*	*x, y, z, ...*	(variables)
	|	λ*x.M*	(*M* as function of *x*)
	|	*M (N)*	(*M* applied to *N*)



- Computations “reductions”

$$(\lambda x.M)(N) \longrightarrow M\{x := N\}$$

Examples of reductions (1/2)

- Examples

$(\lambda x.x)N \longrightarrow N$

$(\lambda f.f\ N)(\lambda x.x) \longrightarrow (\lambda x.x)N \longrightarrow N$

$(\lambda x.x\ N)(\lambda y.y) \longrightarrow (\lambda y.y)N \longrightarrow N$ (name of bound variable is meaningless)

$(\lambda x.\ x\ x)(\lambda x.xN) \longrightarrow (\lambda x.xN)(\lambda x.xN) \longrightarrow (\lambda x.xN)N \longrightarrow NN$

$(\lambda x.x)(\lambda x.x) \longrightarrow \lambda x.x$

Let $I = \lambda x.x$, we have $I(x) = x$ for all x.

Therefore $I(I) = I$. [Church 41]

Examples of reductions (1/2)

- Examples

$$(\lambda x.\, x\, x)(\lambda x.xN) \longrightarrow (\lambda x.xN)(\lambda x.xN) \longrightarrow (\lambda x.xN)N \longrightarrow NN$$

$$(\lambda x.\, x\, x)(\lambda x.\, x\, x) \longrightarrow (\lambda x.\, x\, x)(\lambda x.\, x\, x) \longrightarrow \cdots$$

- Possible to loop inside applications of functions ...

$$Y_f = (\lambda x.f(xx))(\lambda x.f(xx)) \longrightarrow f((\lambda x.f(xx))(\lambda x.f(xx))) = f(Y_f)$$

$$f(Y_f) \longrightarrow f(f(Y_f)) \longrightarrow \cdots \longrightarrow f^n(Y_f) \longrightarrow \cdots$$

- Every computable function can be computed by a lambda-term,

→ Church's thesis. [Church 41]

Fathers of computability

Alonzo Church

Stephen Kleene

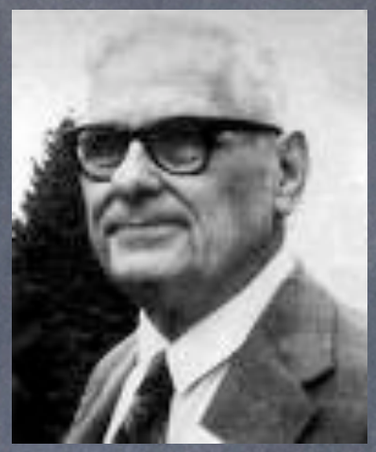
The Giants of computability

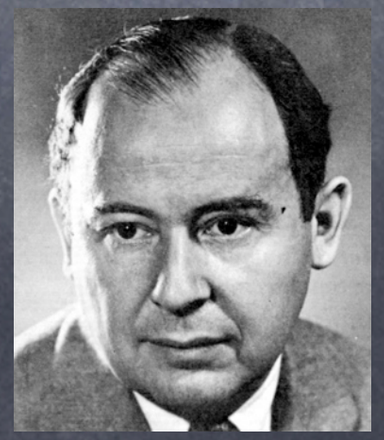
Hilbert → Gödel → Church → Turing

Church → Kleene

Post Curry

von Neumann





Independent computations everywhere

CENTRE DE RECHERCHE COMMUN

INRIA
MICROSOFT RESEARCH

Pure functional languages (1/2)

- Evaluations of subexpressions are independent

$$M\ M_1\, M_2 \cdots M_n$$

- Evaluations of M and M_i can be done in parallel
- No longer true if effects within some of M_i
  - In Haskell, monads are the only effect-ful subterms
  - monads may call pure functional terms
  - pure functional terms cannot call monads
  - effects are visible in types
- In other languages, static analysis necessary to detect effects

Pure functional languages (2/2)

- Inside functional languages, there exists dependencies

$$M\,N \xrightarrow{*} (\lambda x\,.\,P)Q \longrightarrow P\{x := Q\}$$

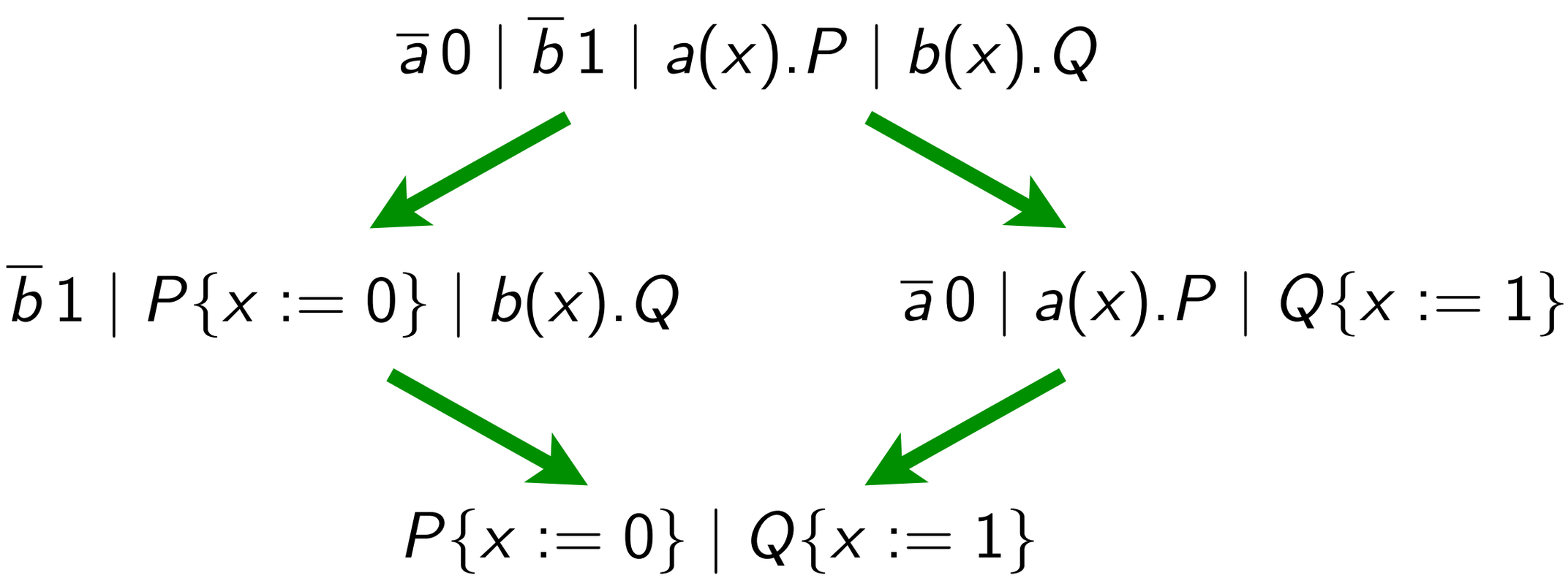
- Evaluations in M has to be done before toplevel redex

$$I\;I(Ia) \longrightarrow I(Ia) \longrightarrow Ia$$

where $I = \lambda x.x$

CCS or pi-calculus (1/2)

- Communications may be independent



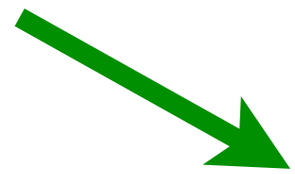
- but other transitions may be causally related

[Jean Krivine] (reversible CCS)

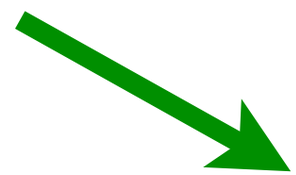
CCS or pi-calculus (2/2)

- example of causally related transitions

$$\overline{a}\,0 \mid a(x).b(y).P \mid \overline{b}\,1$$



$$b(y).P\{x := 0\} \mid \overline{b}\,1$$



$$P\{x := 0, y := 1\}$$

- causality in process algebras == event structures or Petri nets
- « true concurrency »

[Winskel, Boudol-Castellani]

CCS or pi-calculus (3/3)

- possible conflicts

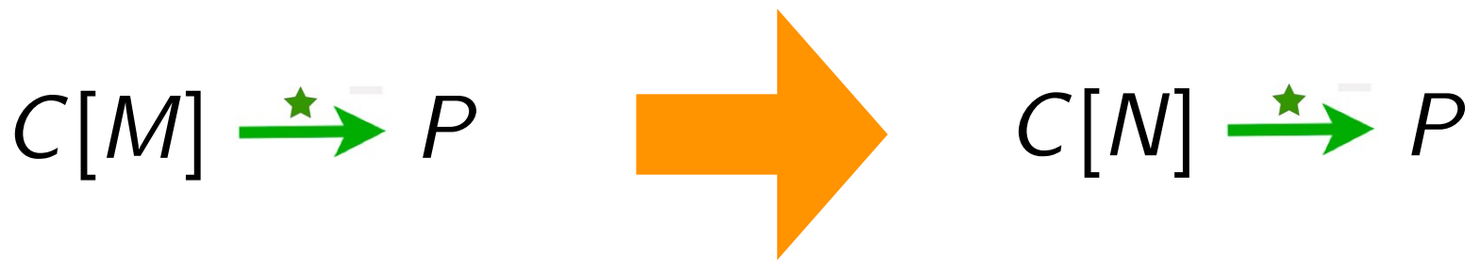
$$\overline{a}\,0 \mid \overline{b}\,1 \mid a(x).P \mid a(y).Q$$

#

$$\overline{b}\,1 \mid P\{x := 0\} \mid a(y).Q \qquad \overline{b}\,1 \mid a(x).P \mid Q\{y := 0\}$$

Non interference in Security

- Private and Public expressions



when *P* is public and *M*, *N* are private

- Computations in *M* do not interfer on result *P*

[Volpano-Smith, Pottier-Simonet, Boudol]

- ... information flow

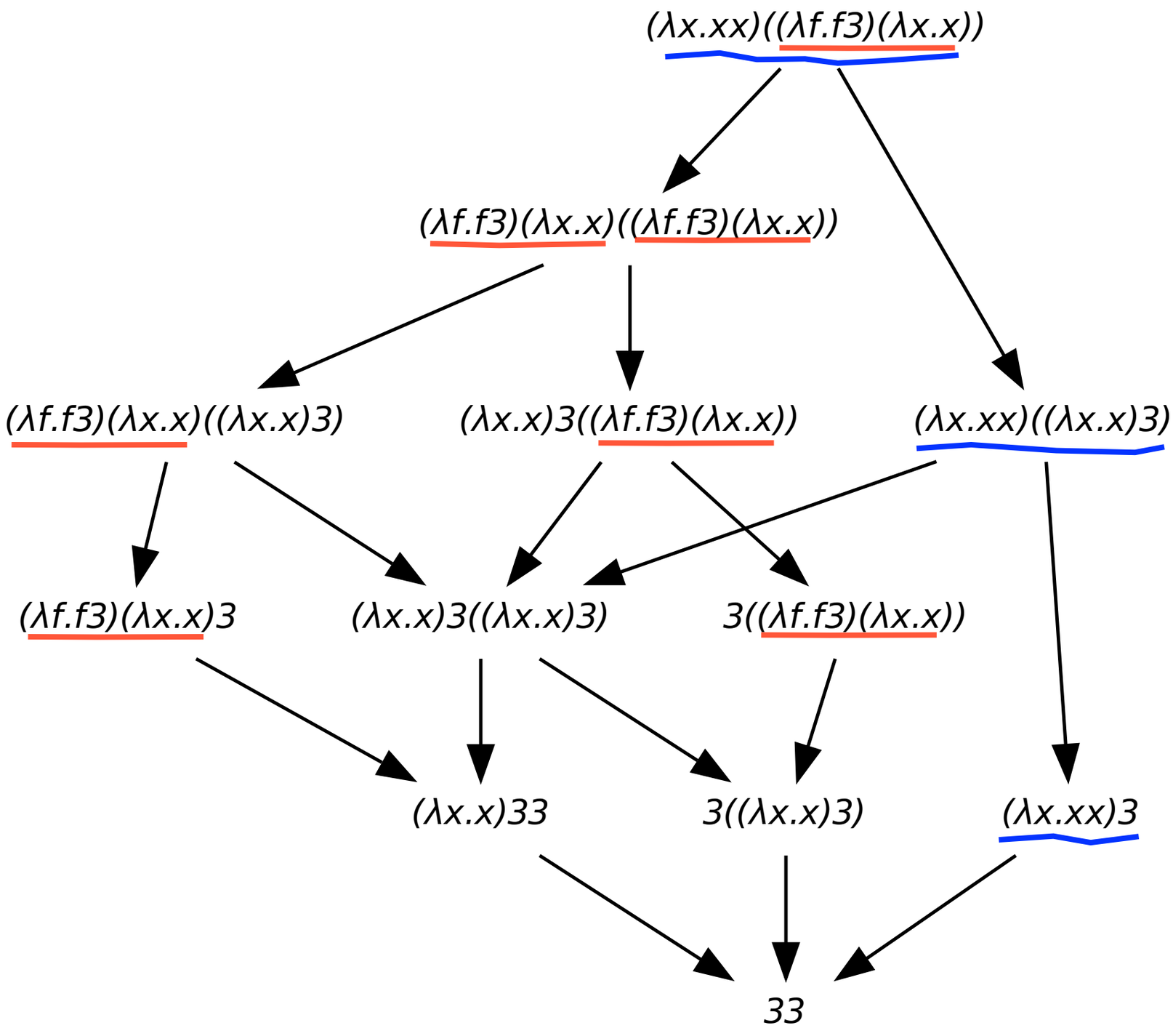
Exercices

CENTRE DE RECHERCHE COMMUN

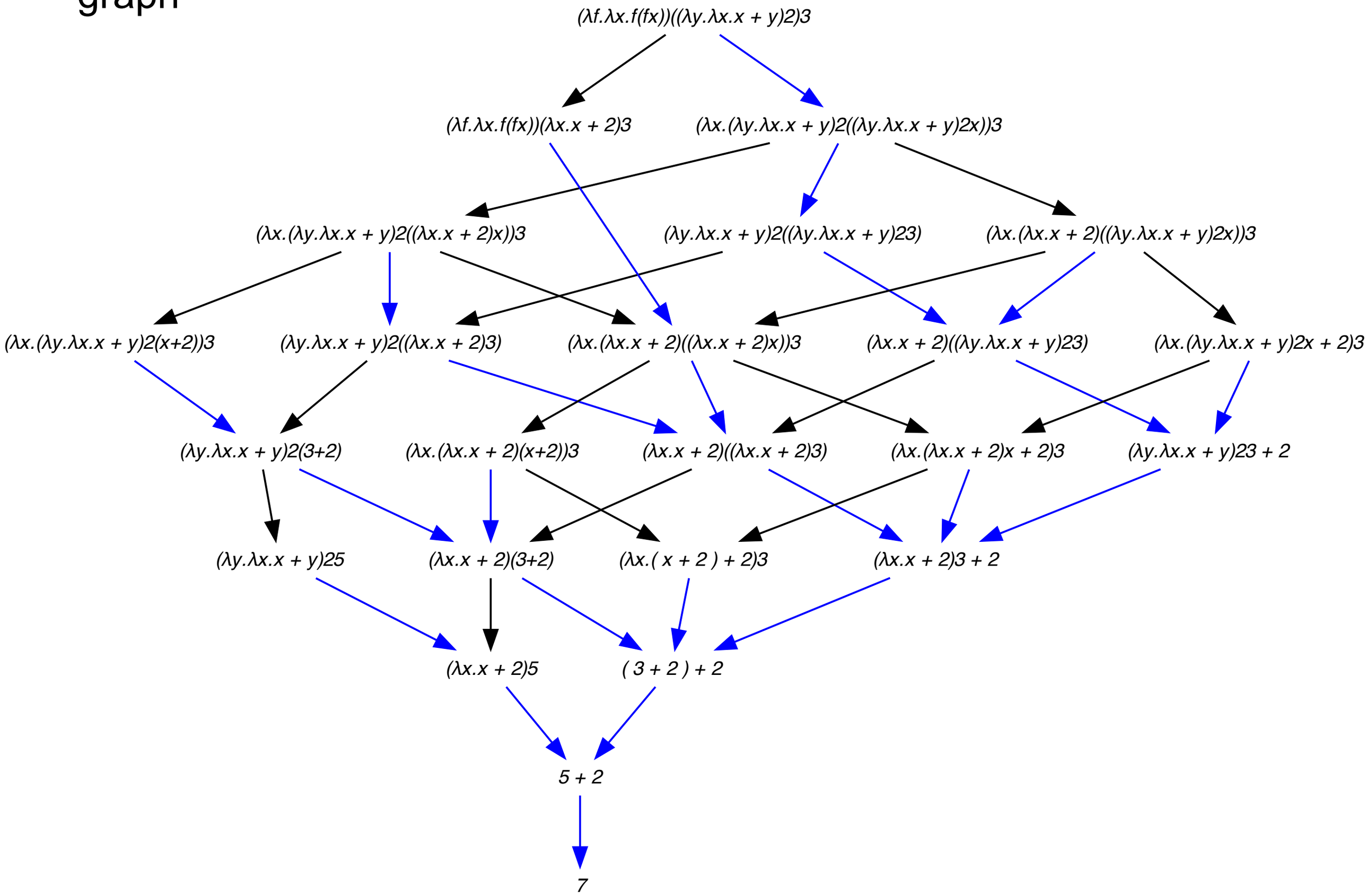
INRIA
MICROSOFT RESEARCH

Exercice

- Show reductions equivalent by permutations in following reduction graph



- Show reductions equivalent by permutations in following reduction graph



• Show reductions equivalent by permutations in following reduction graph