Lambda-Calculus (I)

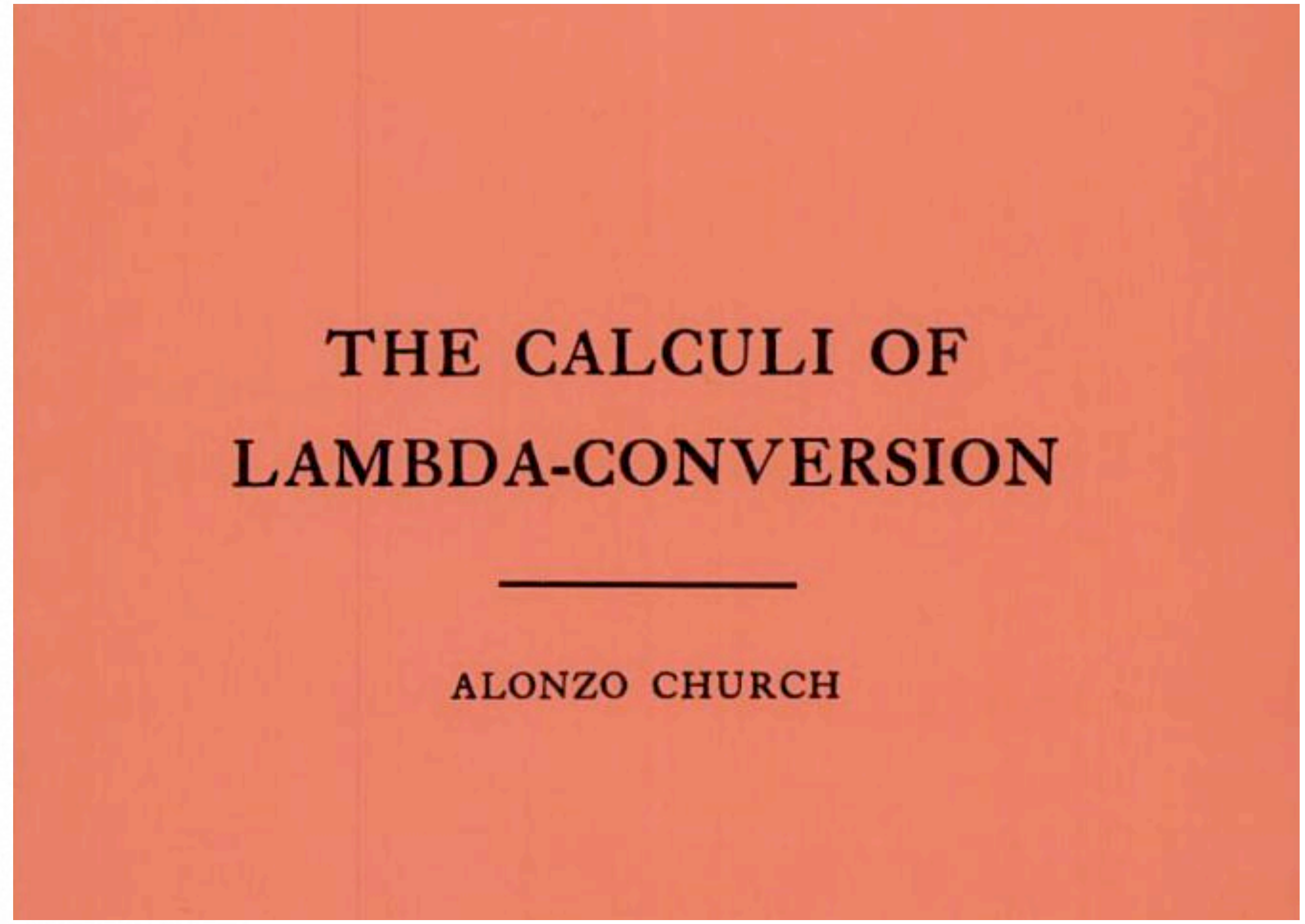
jean-jacques.levy@inria.fr

2nd Asian-Pacific Summer School
on Formal Methods

Tsinghua University,
August 23, 2010

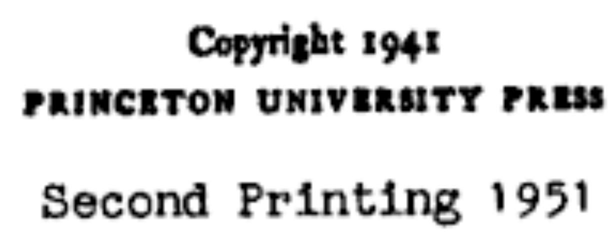
Plan

- computation models
- lambda-notation
- bound variables
- conversion rules
- reductions
- normal forms
- numeral systems
- lambda-definability



THE CALCULI OF
LAMBDA-CONVERSION

ALONZO CHURCH



Copyright 1941
PRINCETON UNIVERSITY PRESS

Second Printing 1951

Barendregt, Henk, The Lambda Calculus. Its Syntax and Semantics, Elsevier, 2nd edition, 1997.

Barendregt, Henk; Dezani, Mariangiola, Lambda calculi with Types, 2010.

Models of computation

CENTRE DE RECHERCHE COMMUN
INRIA MICROSOFT RESEARCH

Computation models

- [machines] automata theory -- **Turing** machines
- [character strings] formal grammars, Thue systems, **Post**
- [numbers] **Kleene** recursive functions theory
- [terms] **Church** **lambda-calculus**, term rewriting systems

Applications to logic

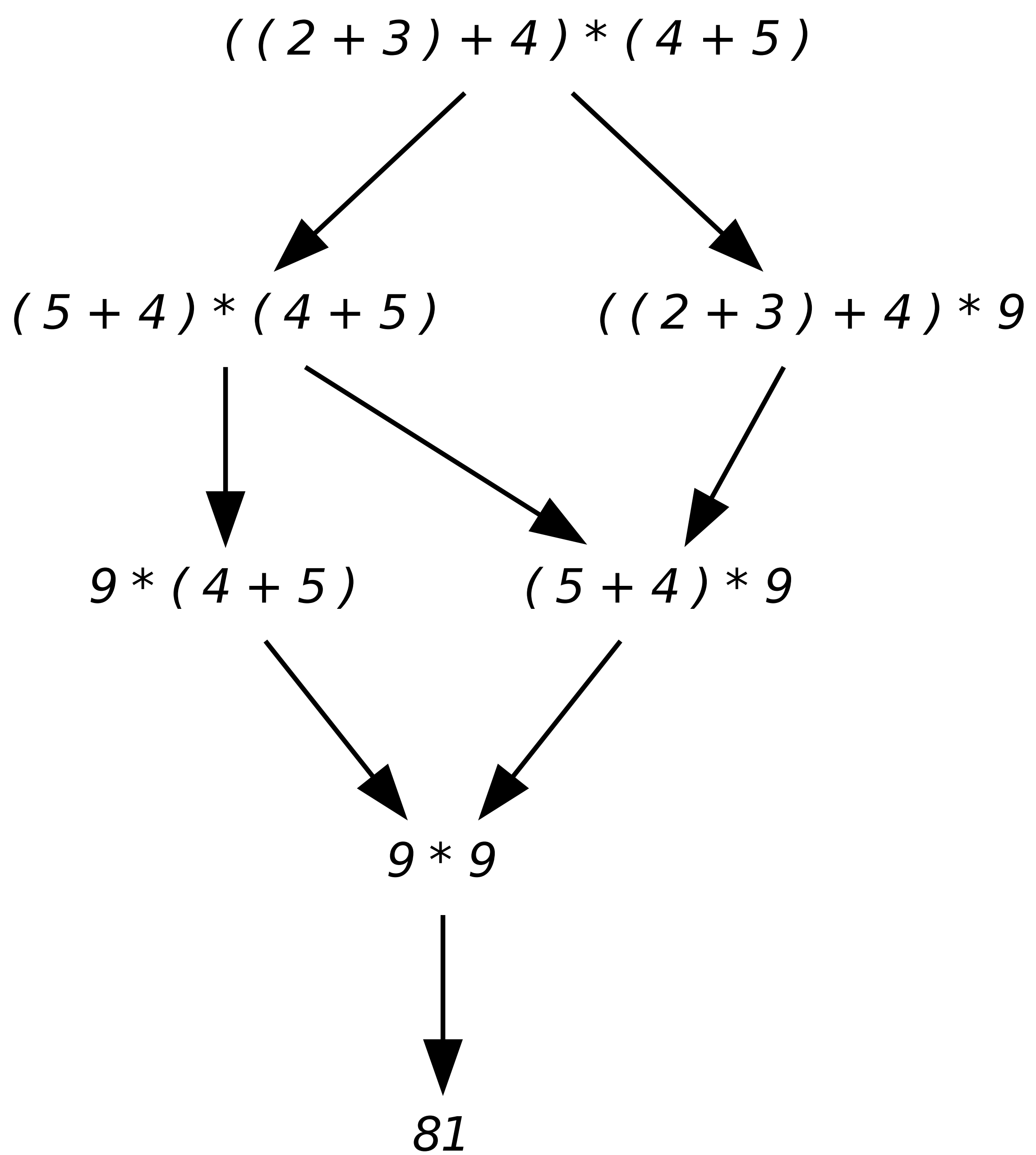
- [cut elimination] 2nd order arithmetic -- **Howard, Girard**
- [higher order dependent types] HOL, Isabelle, Coq -- **Coquand, Huet**

Computing with terms

$2 + 3 \longrightarrow 5$

$(2 + 3) + 4 \longrightarrow 5 + 4 \longrightarrow 9$

$((2 + 3) + 4) * (4 + 5) \longrightarrow \ldots$



Computing with terms

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

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

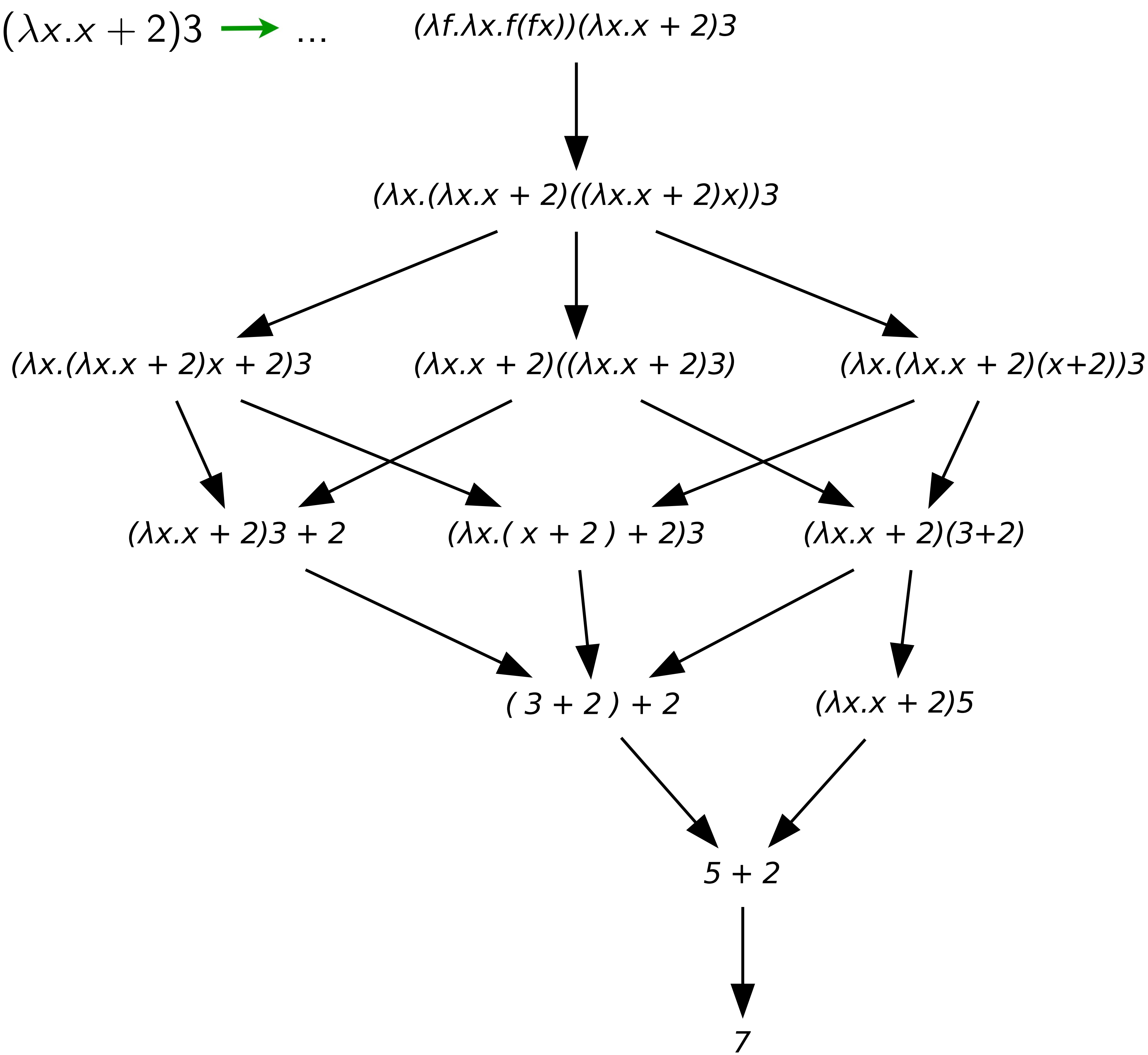
$(\lambda f.\, f 3)(\lambda x.\, x + 2) \longrightarrow (\lambda x.\, x + 2)3 \longrightarrow 3 + 2 \longrightarrow 5$

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

$$\begin{tikzcd}
 & (\lambda f.\lambda x.f(fx))(\lambda x.x + 2) \arrow[d] & \\
 & \lambda x.(\lambda x.x + 2)((\lambda x.x + 2)x) \arrow[dl] \arrow[dr] & \\
\lambda x.(\lambda x.x + 2)x + 2 \arrow[dr] & & \lambda x.(\lambda x.x + 2)(x+2) \arrow[dl] \\
 & \lambda x.(\ x + 2\ ) + 2 &
\end{tikzcd}$$

Computing with terms

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



Computation model

- define a **minimum** set
- no instructions, no states, only **expressions**
- no arithmetic
- just a calculus of **functions**
- functions applied to functions
- functions as results

interesting ?

λ-calculus

CENTRE DE RECHERCHE
COMMUN

INRIA
MICROSOFT RESEARCH

The lambda-calculus

- Lambda terms

M, N, P	::=	*x, y, z, ...*	(variables)
	|	($\lambda x.M$)	(M as function of x)
	|	($M \ \ N$)	(M applied to N)
	|	*c, d, ...*	(*constants*)

- Calculations “reductions”

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

Abbreviations

$MM_1M_2\cdots M_n$ for $(\cdots((MM_1)M_2)\cdots M_n)$

$(\lambda x_1x_2\cdots x_n\,.\,M)$ for $(\lambda x_1.(\lambda x_2.\cdots(\lambda x_n\,.\,M)\cdots))$

external parentheses and parentheses after a dot may be forgotten

Exercice 1

Write following terms in long notation:

$\lambda x.x,\ \lambda x.\lambda y.x,\ \lambda xy.x,\ \lambda xyz.y,\ \lambda xyz.zxy, \lambda xyz.z(xy),$
$(\lambda x.\lambda y.x)MN,\ (\lambda xy.x)MN,\ (\lambda xy.y)MN,\ (\lambda xy.y)(MN)$

Examples

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

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

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

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

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

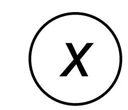
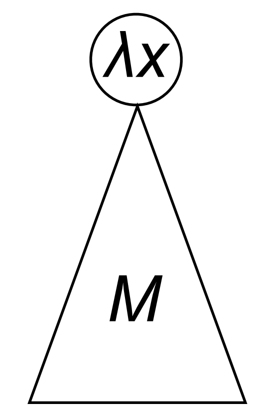
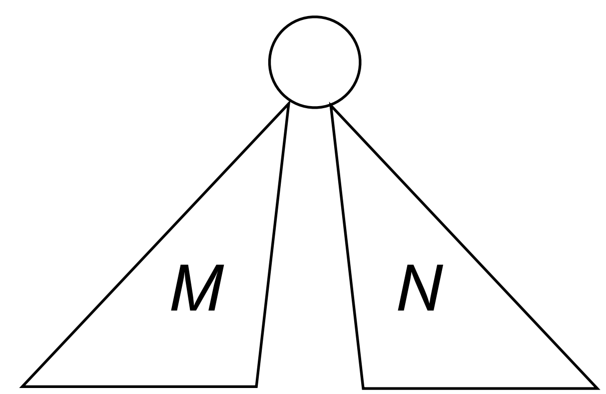
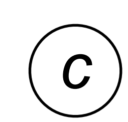
$$f(Y_f) \longrightarrow f(f(Y_f)) \longrightarrow \cdots \longrightarrow f^n(Y_f) \longrightarrow \cdots$$

Recapitulation

- calculus is more complex than expected
- looping expressions !!
- recursion operator seems definable
- when termination ?
- consistency ?
- computing power ?

Abstract syntax

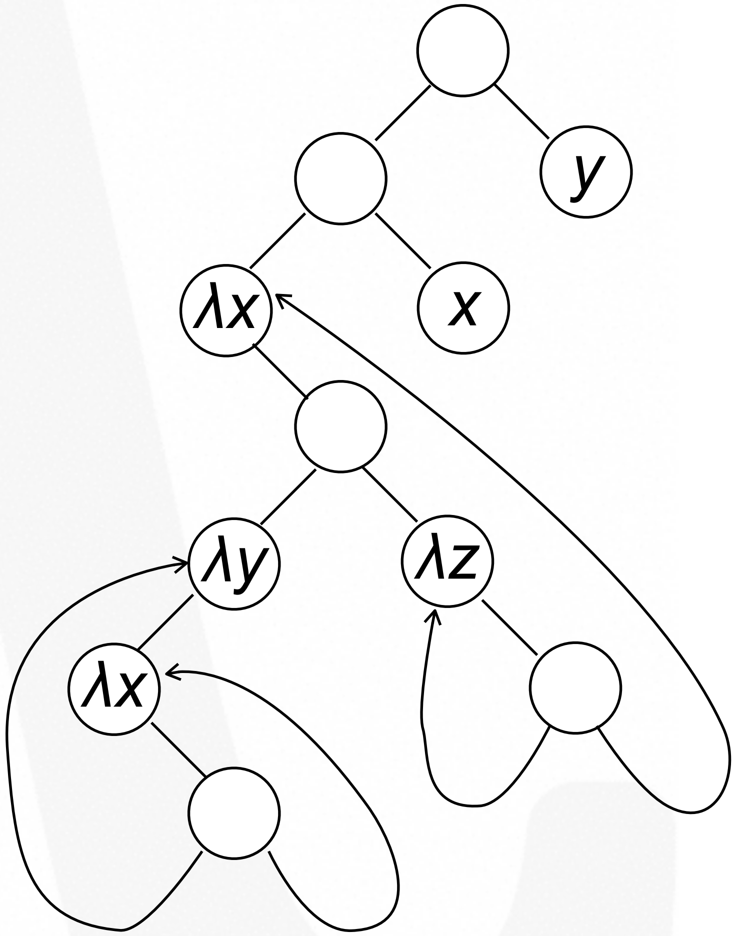
- The syntax of lambda-terms can be abstracted as:

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

Abstract syntax

- Example: $(\lambda x.(\lambda y.\lambda x.y\,x)\,(\lambda z.z\,x))x\,y$

  is



Bound variables

$(\lambda x.(\lambda y.\lambda x.y\,x)\,(\lambda z.z\,x))x\,y$ (rightmost *x*, *y* are free)

Exercice 2

- Show binders of bound variables in

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

$(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))(\lambda f\,x\,y.x(f\,y))$

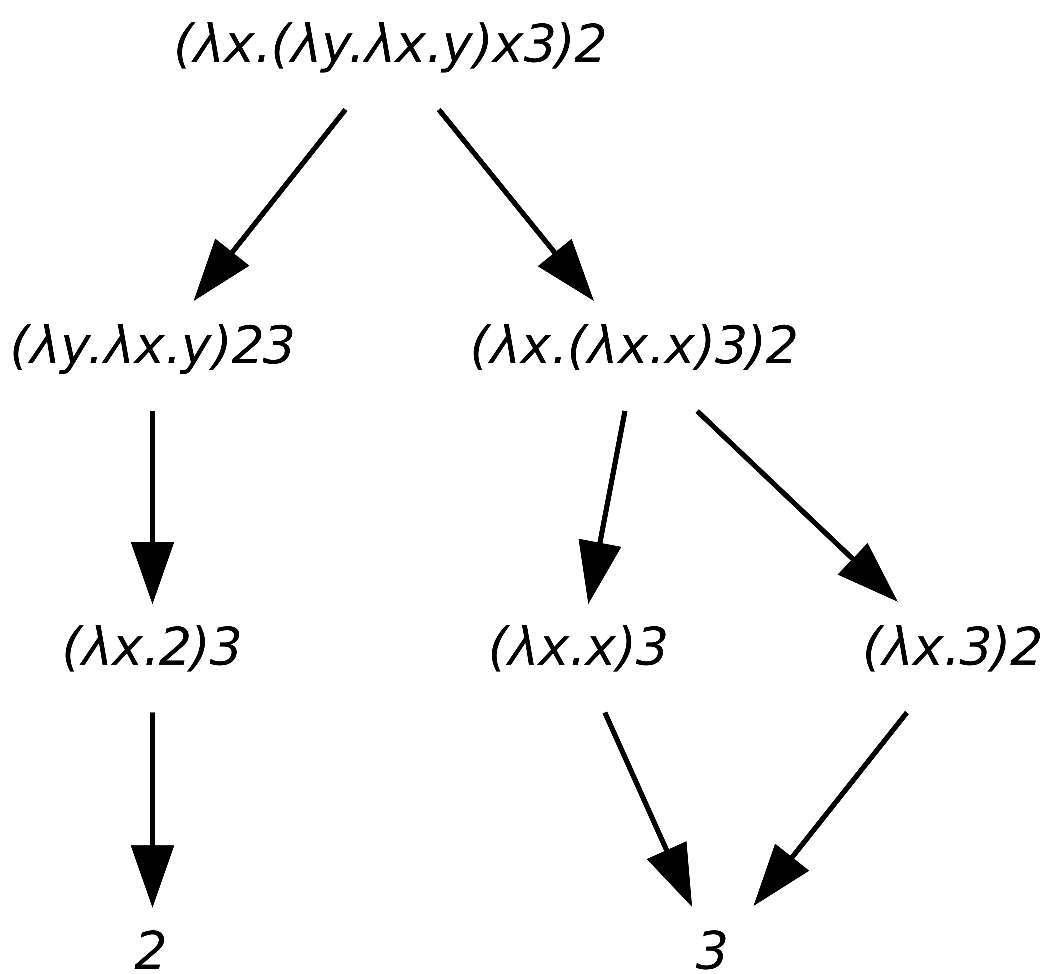
$(\lambda f.f((\lambda x.x)3))(\lambda x.\lambda y.x)$

Bound variables

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

incorrect

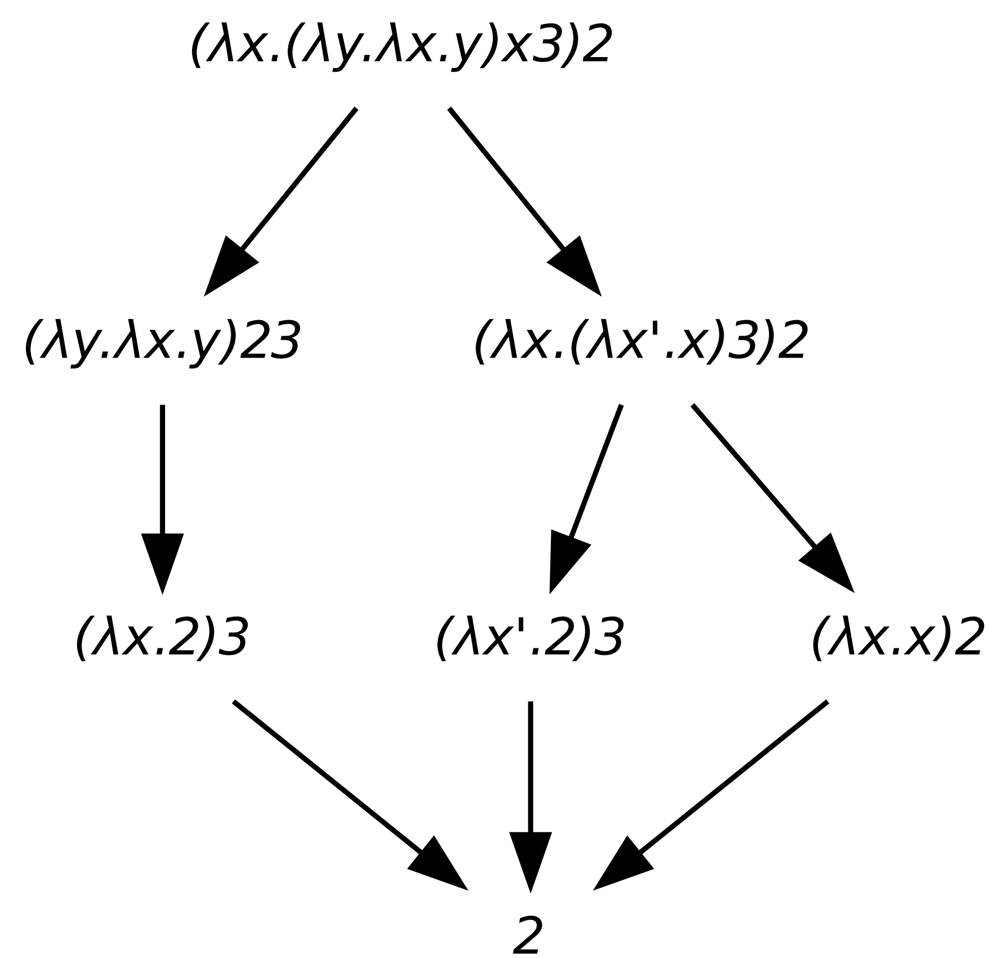
(dynamic binding: Lisp)



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

correct

(lexical binding: Scheme)



Exercice 2bis

Why Lisp is consistent ?

Bound variables

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

$$(\lambda y.\lambda x.y)x \;=_\alpha\; (\lambda y.\lambda x'.y)x \longrightarrow \lambda x'.x$$

- **renaming** of bound variables
- **names** of bound variables are **not important**
- standard in many other calculi

$$\int_0^{\pi/2} cos(x)dx \;=\; \int_0^{\pi/2} cos(x')dx' \qquad \sum_{i=1}^{9} a_i \;=\; \sum_{j=1}^{9} a_j$$

$$\lambda x.x + 2 \;=_\alpha\; \lambda y.y + 2 \qquad \lambda xy.x + y \;=_\alpha\; \lambda yx.y + x$$

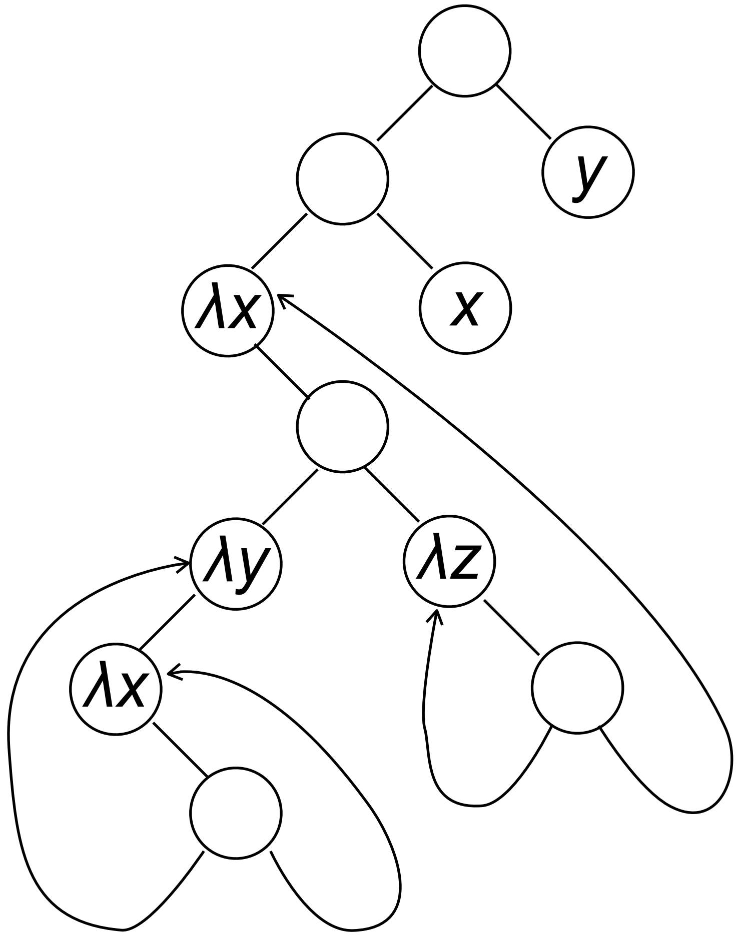
Bound variables

- **de Bruijn indices** is a systematic computer representation of bound variables

- for each occurence of a bound variable, one counts the number of binders to traverse to reach its binder.

- Example: $(\lambda x.(\lambda y.\lambda x.y\ x)\ (\lambda z.z\ x))x\ y$

  is $(\lambda.(\lambda.\lambda.\underline{1}\ \underline{0})\ (\lambda.\underline{0}\ \underline{1}))x\ y$



Substitution

$x\{y := P\} = x \qquad\qquad c\{y := P\} = c$

$y\{y := P\} = P$

$(MN)\{y := P\} = M\{y := P\}\ N\{y := P\}$

$(\lambda y.M)\{y := P\} = \lambda y.M$

$(\lambda x.M)\{y := P\} = \lambda x'.M\{x := x'\}\{y := P\}$

where $x' = x$ if y not free in M or x not free in P,
otherwise x' is the first variable not free in M and P.
(we suppose that the set of variables is infinite and enumerable)

Free variables

$\mathrm{var}(x) = \{x\} \qquad\qquad \mathrm{var}(c) = \emptyset$

$\mathrm{var}(MN) = \mathrm{var}(M) \cup \mathrm{var}(N)$

$\mathrm{var}(\lambda x.M) = \mathrm{var}(M) - \{x\}$

Conversion rules

$$
\begin{array}{rcll}
\lambda x.M & \rightarrow_{\alpha} & \lambda x'.M\{x := x'\} & (x' \notin \mathrm{var}(M)) \\
(\lambda x.M)N & \rightarrow_{\beta} & M\{x := N\} & \\
\lambda x.Mx & \rightarrow_{\eta} & M & (x \notin \mathrm{var}(M))
\end{array}
$$

- left-hand-side of conversion rule is a **redex** (reductible expression)
- α-redex, β-redex, η-redex, ...
- we forget indices when clear from context, often β

Reduction step

- let R be a redex in M. Then one can contrat redex R in M and get N:

$$M \xrightarrow{R} N$$

Reductions

$$M \xrightarrow{\star} N \quad \text{when} \quad M = M_0 \longrightarrow M_1 \longrightarrow M_2 \longrightarrow \cdots M_n = N \quad (n \geq 0)$$

- same with explicit contracted redexes

$$M = M_0 \xrightarrow{R_1} M_1 \xrightarrow{R_2} M_2 \cdots \xrightarrow{R_n} M_n = N$$

- and with named reductions

$$\rho : M = M_0 \xrightarrow{R_1} M_1 \xrightarrow{R_2} M_2 \cdots \xrightarrow{R_n} M_n = N$$

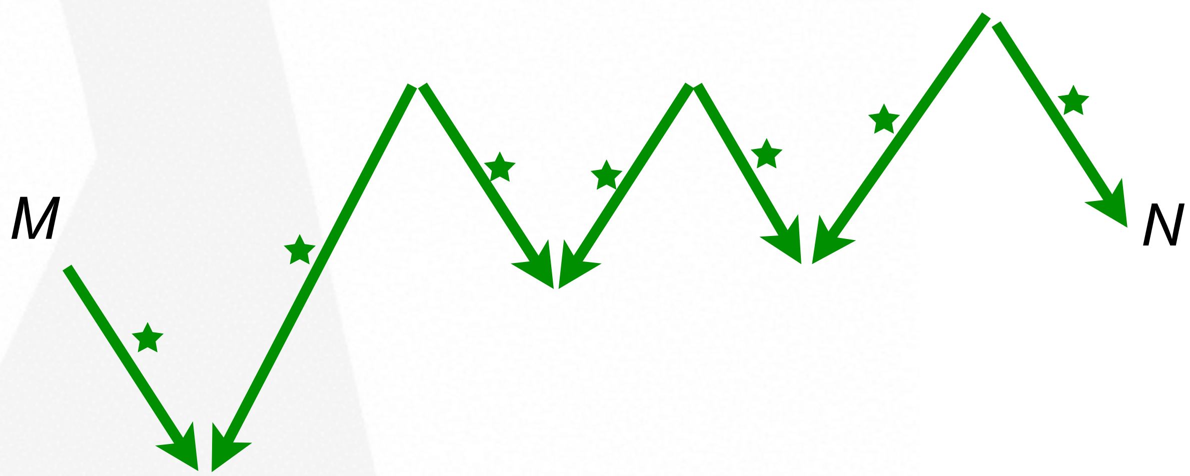
- we speak of redex occurences when specifying reduction steps,

  but it is convenient to confuse redexes and redex occurences when clear from context

Lambda theories

$M =_\beta N$ when M and N are related by a zigzag of reductions

M and N are said **interconvertible**



- Also $M =_\alpha N$, $M =_\eta N$, $M =_{\beta,\eta} N$, ...
- Interconvertibility is symmetric, reflexive, transivite closure of reduction relation
- or with notations of mathematical logic:

$$\alpha \vdash M = N, \quad \beta \vdash M = N, \quad \eta \vdash M = N, \quad \beta + \eta \vdash M = N, \ldots$$

- the syntactic equality $M = N$ will often stand for $M =_\alpha N$.

Exercice 3

- Find terms M such that:

$$M \longrightarrow M$$

$$M = M_0 \longrightarrow M_1 \longrightarrow M_2 \longrightarrow \cdots M_n = M \quad (M_i \text{ all distinct})$$

$$M =_\beta x\, M$$

$$M =_\beta \lambda x.M$$

$$M =_\beta MM$$

$$M =_\beta MN_1N_2 \cdots N_n \ \text{ for all } N_1, N_2, \ldots N_n$$

- Find term Y such that, for any M:

$$YM =_\beta M(YM)$$

- Find Y' such that, for any M:

$$Y'M \xrightarrow{*} M(Y'M)$$

- (difficult) Show there is only one redex R such that $R \longrightarrow R$

Normal forms

- An expression M without redexes **is in** normal form

$$M \not\rightarrow$$

- If M reduces to a normal form, then M **has a** normal form

$$M \xrightarrow{*} N, \quad N \text{ in normal form}$$

Exercice 4

- which of following terms are in β-normal form ?

in $\beta\eta$-normal form ?

$\lambda x.x$

$\lambda xy.x$

$\lambda xy.xy$

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

$\lambda x.x(\lambda xy.x)(\lambda x.x)$

$\lambda xy.x(\lambda xy.x)(\lambda x.yx)$

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

Exercice 5

- Show that if M is in normal form and $M \xrightarrow{*} N$, then $M = N$
- Show that:

1- $\lambda x.M \xrightarrow{*} N$ implies $N = \lambda x.N'$ and $M \xrightarrow{*} N'$

2- $MN \xrightarrow{*} P$ implies $M \xrightarrow{*} M'$, $N \xrightarrow{*} N'$ and $P = M'N'$
or $M \xrightarrow{*} \lambda x.M'$, $N \xrightarrow{*} N'$ and $M'\{x := N'\} \xrightarrow{*} P$

3- $xM_1M_2\cdots M_n \xrightarrow{*} N$ implies $M_1 \xrightarrow{*} N_1$, $M_2 \xrightarrow{*} N_2$, ... $M_n \xrightarrow{*} N_n$
and $xN_1N_2\cdots N_n = N$

4- $M\{x := N\} \xrightarrow{*} \lambda y.P$ implies $M \xrightarrow{*} \lambda y.M'$ and $M'\{x := N\} \xrightarrow{*} P$
or $M \xrightarrow{*} xM_1M_2\cdots M_n$ and $NM_1\{x := N\}\cdots M_n\{x := N\} \xrightarrow{*} \lambda y.P$

δ-rules

CENTRE DE RECHERCHE COMMUN
INRIA MICROSOFT RESEARCH

Adding δ-rules: PCF

- Terms of PCF

M, N, P	::=	$x, y, z, \ldots$	(variables)
	|	$\lambda x.M$	(M function of x)
	|	$M\ N$	(M applied to N)
	|	$\mathtt{n}$	(integer constant)
	|	$M \otimes N$	(arithmetic operation, +, *, -, /)
	|	$\mathtt{ifz}\ P\ \mathtt{then}\ M\ \mathtt{else}\ N$	(conditionnal)

- Conversion rules

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

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

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

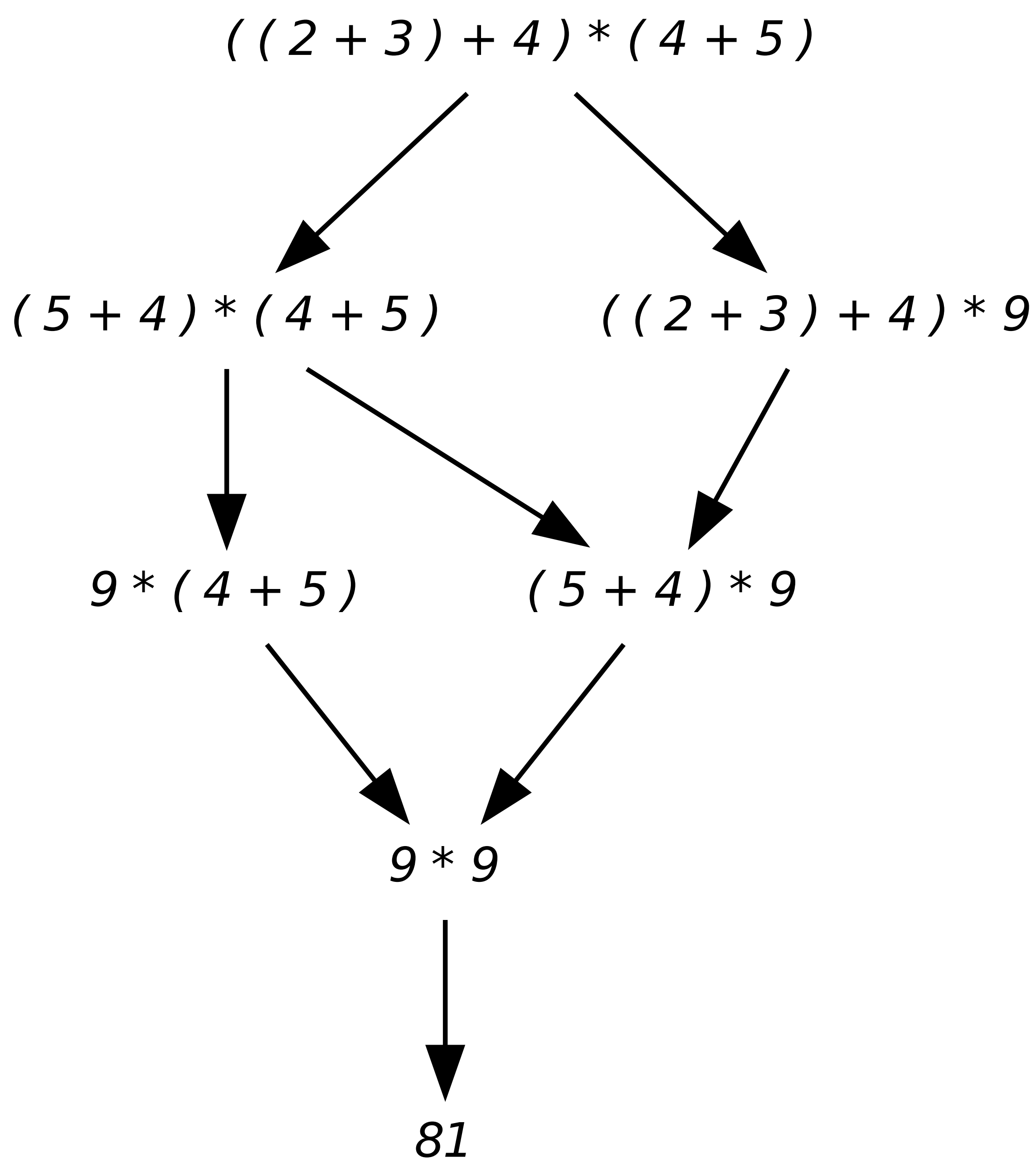
$$\mathtt{ifz}\ \underline{n+1}\ \mathtt{then}\ M\ \mathtt{else}\ N \longrightarrow N$$

Examples (bis)

$2+3 \longrightarrow 5$

$(2+3)+4 \longrightarrow 5+4 \longrightarrow 9$

$((2+3)+4)*(4+5) \longrightarrow \ldots$



Examples (bis)

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

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

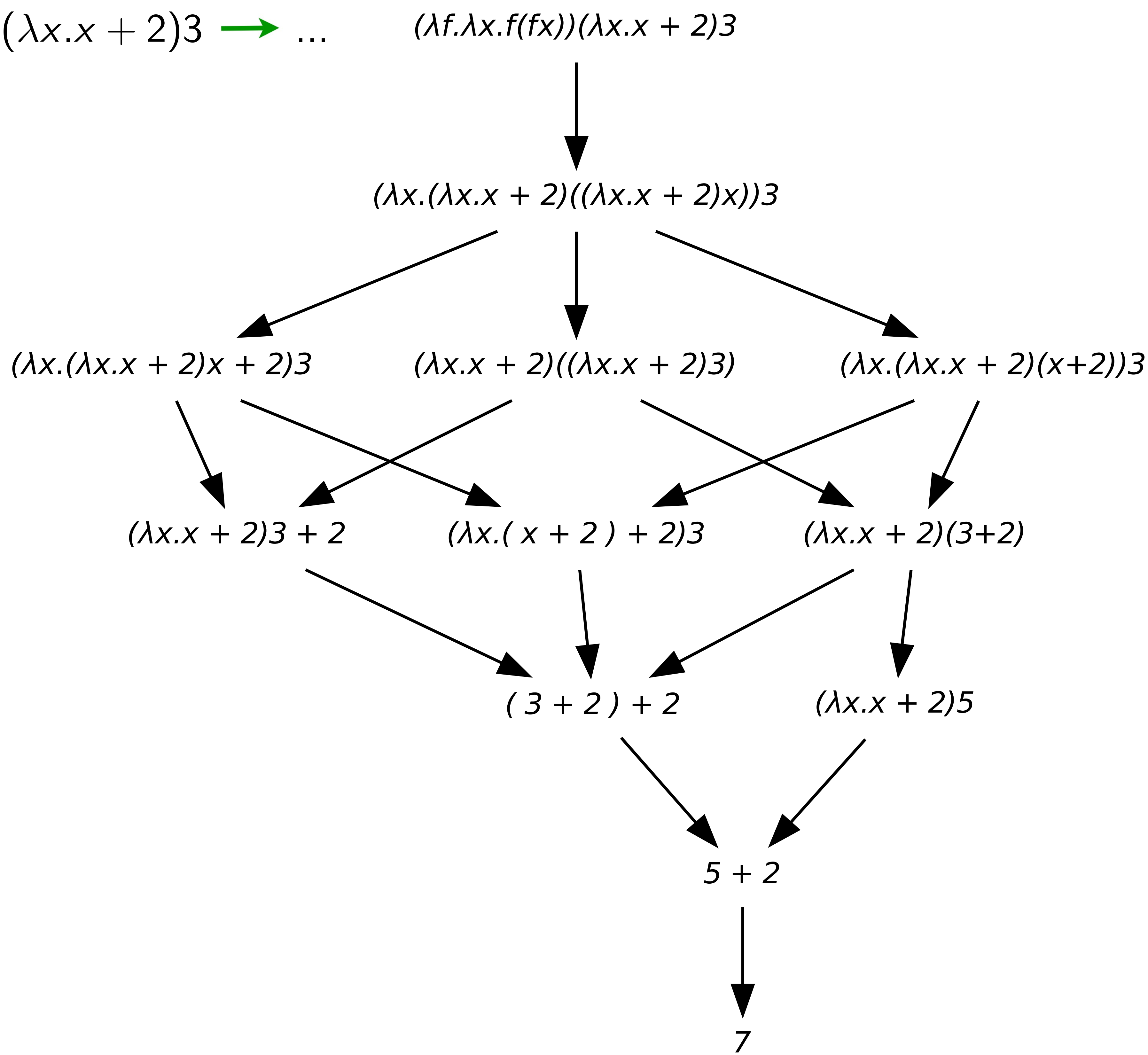
$(\lambda f.f3)(\lambda x.\, x + 2) \rightarrow (\lambda x.\, x + 2)3 \rightarrow 3 + 2 \rightarrow 5$

$(\lambda f.\lambda x.f(f\,x))(\lambda x.x + 2) \rightarrow ..$

$$\begin{tikzcd}
& (\lambda f.\lambda x.f(fx))(\lambda x.x + 2) \arrow[d] & \\
& \lambda x.(\lambda x.x + 2)((\lambda x.x + 2)x) \arrow[dl] \arrow[dr] & \\
\lambda x.(\lambda x.x + 2)x + 2 \arrow[dr] & & \lambda x.(\lambda x.x + 2)(x+2) \arrow[dl] \\
& \lambda x.(\, x + 2\,) + 2 &
\end{tikzcd}$$

Examples (bis)

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



Examples

$\mathtt{Fact}(3)$

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

$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

can be written as a single term in:

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

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

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

(\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

↓

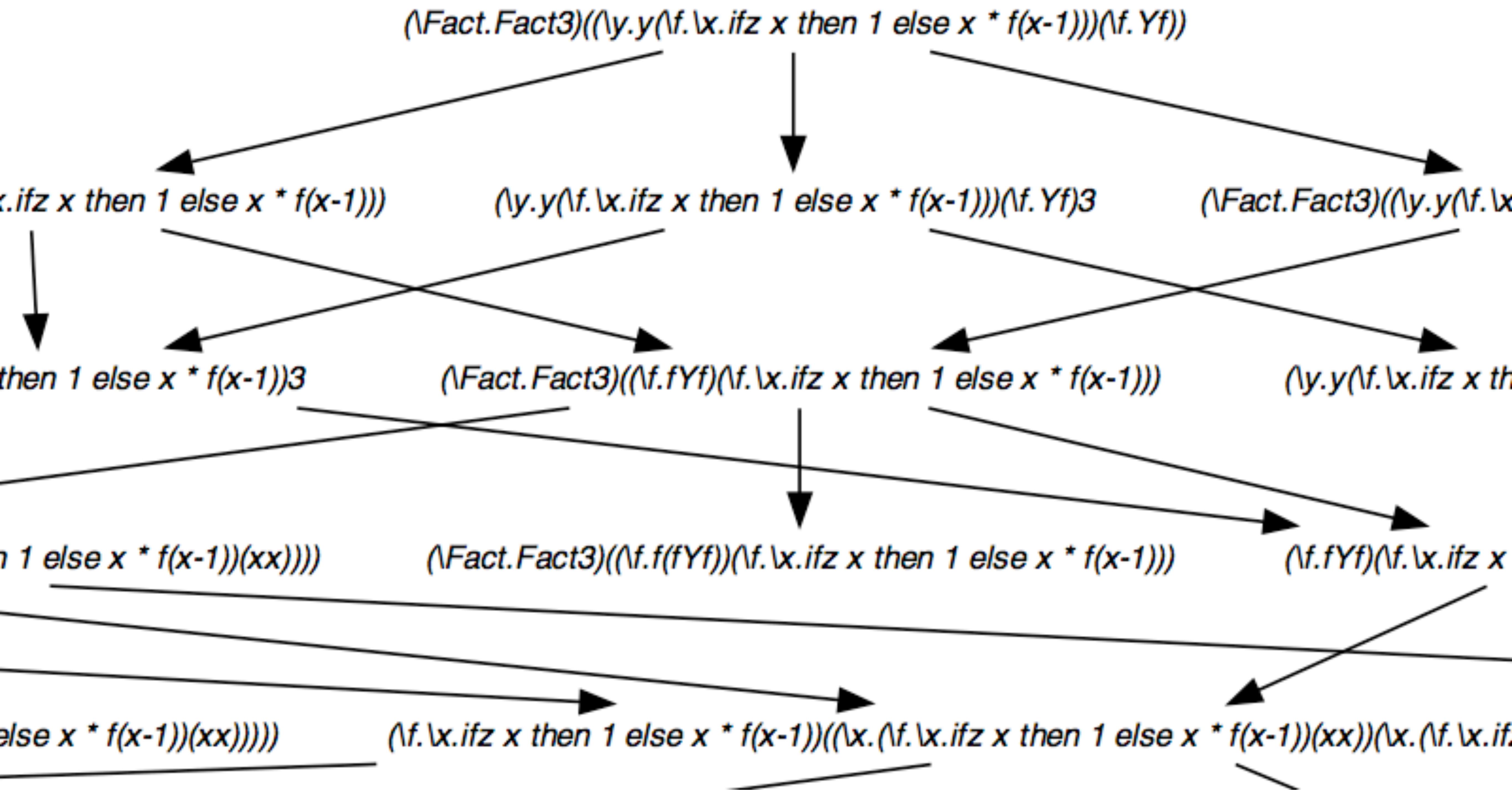
ifz 3 then 1 else 3 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(3-1)

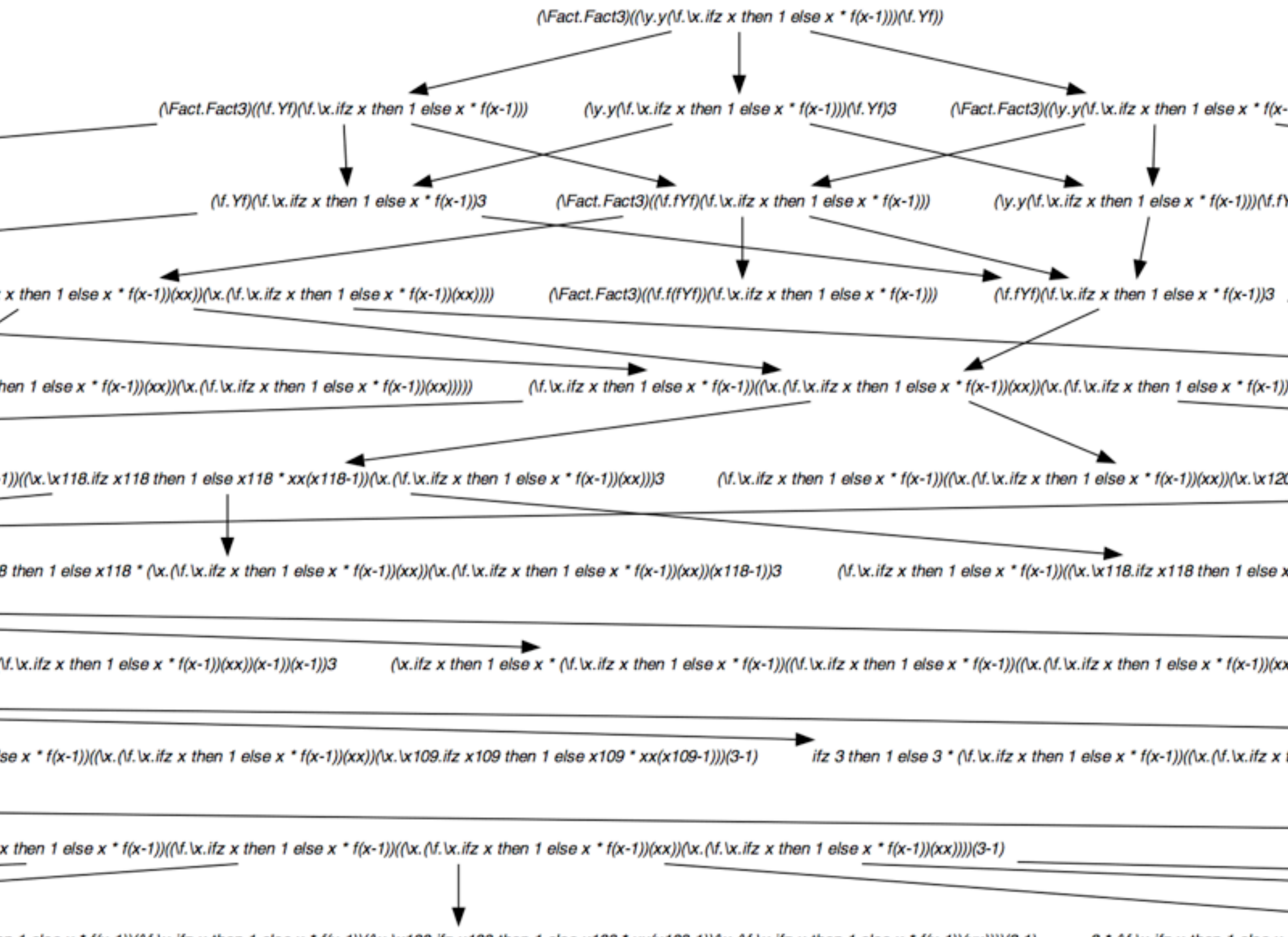
↓

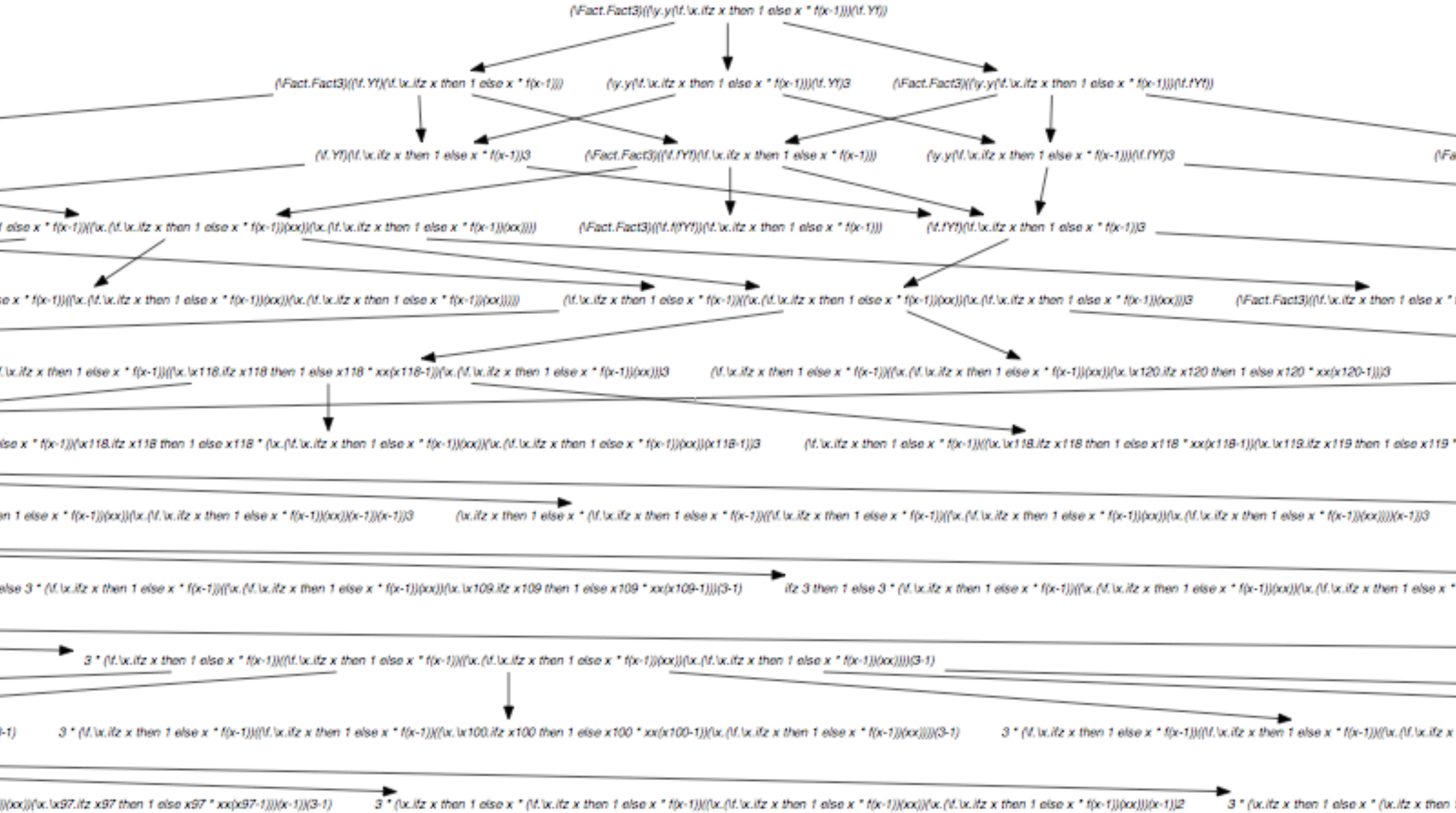
3 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(3-1)

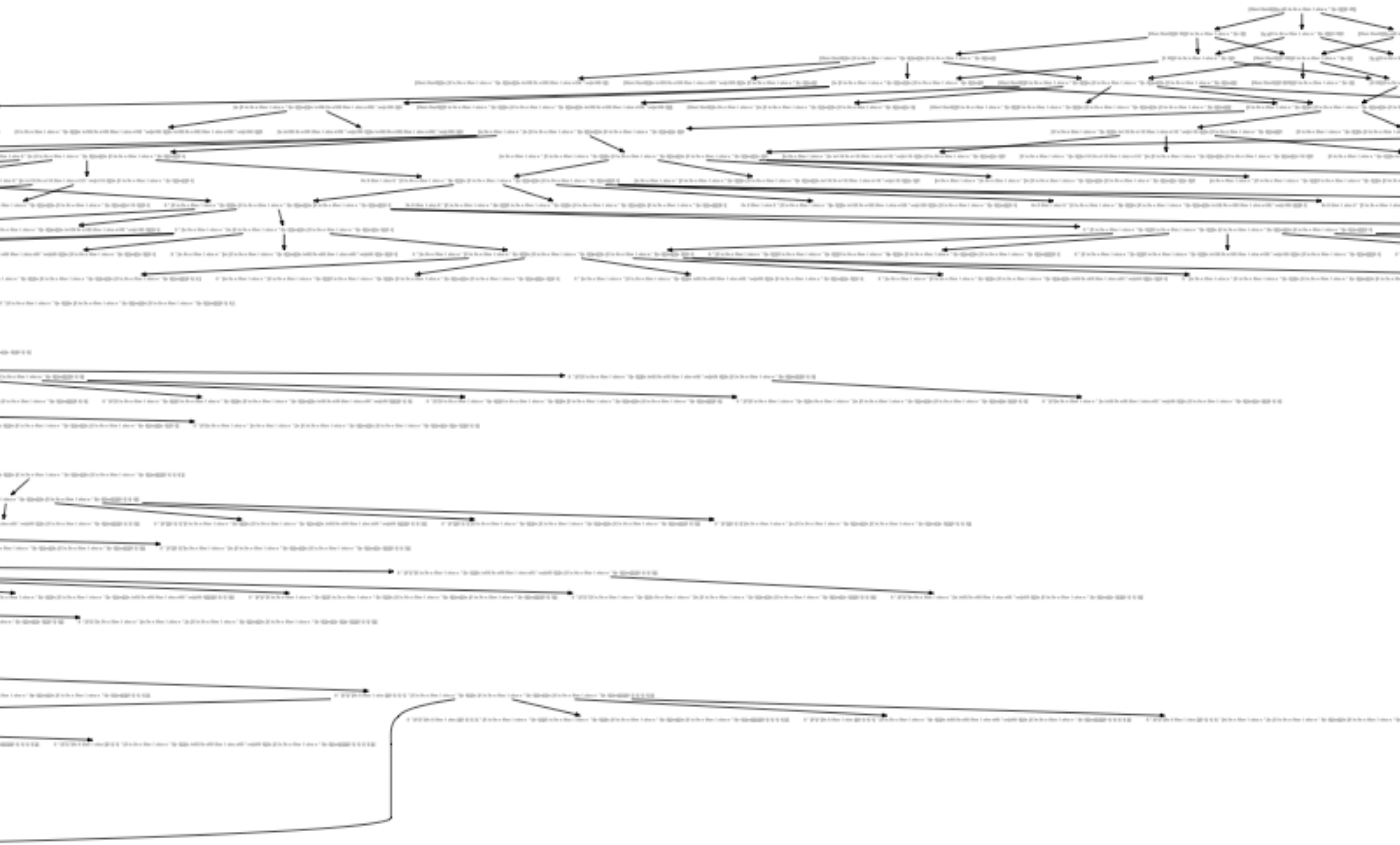
↓

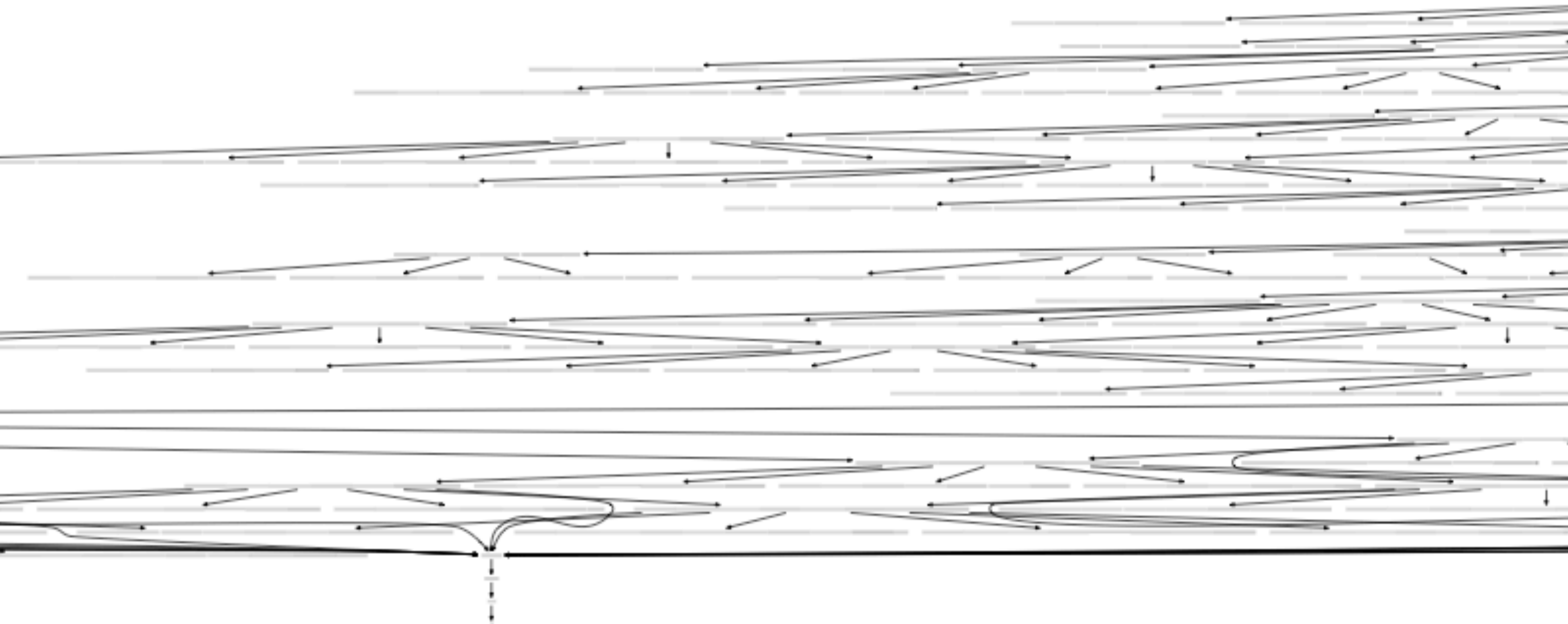
3 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(3-1)

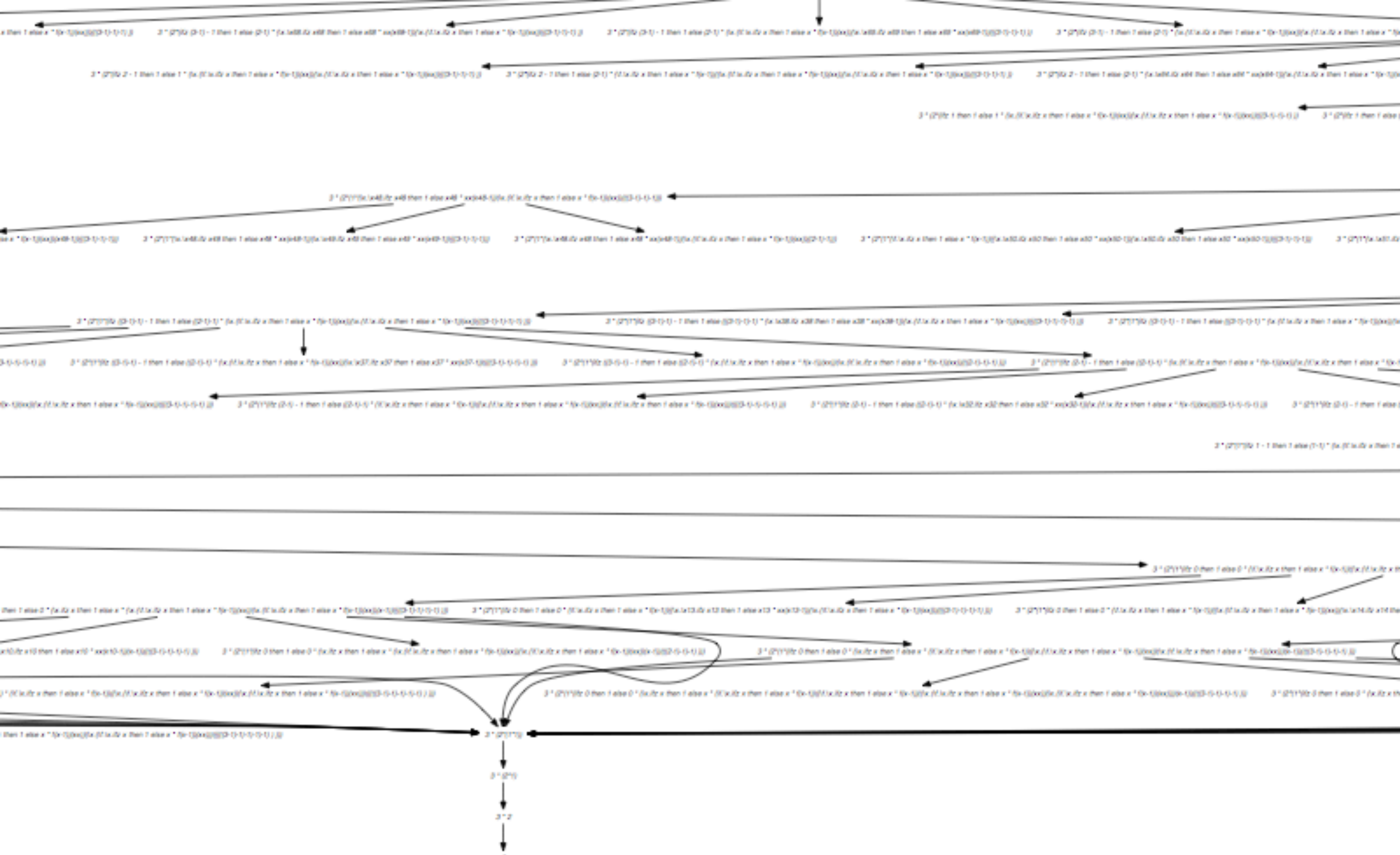

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x
then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x th
1 else x * f(x-1))(xx))))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

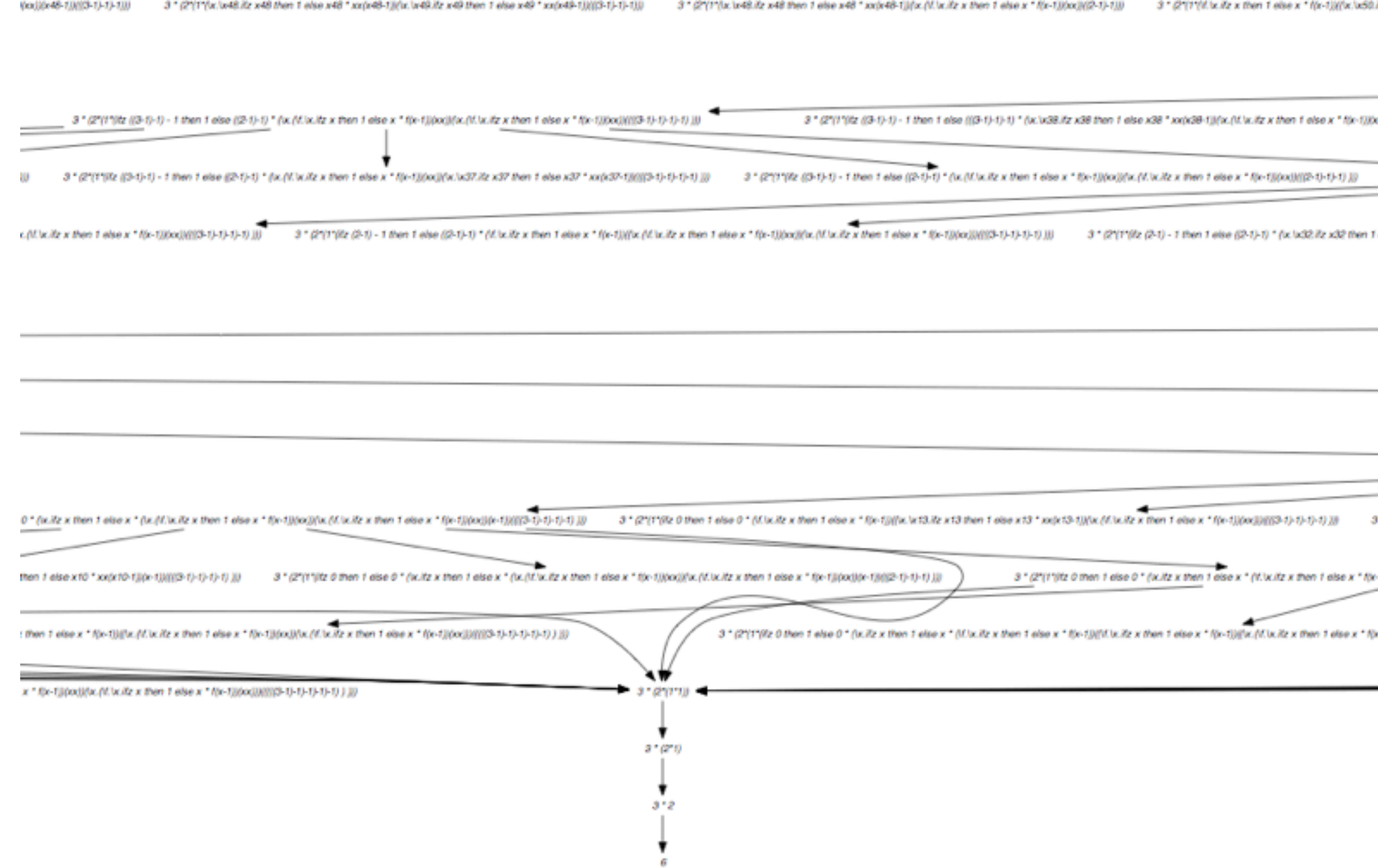

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
(\Fact.Fact3)((\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-
(\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz 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.\x12
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
(\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

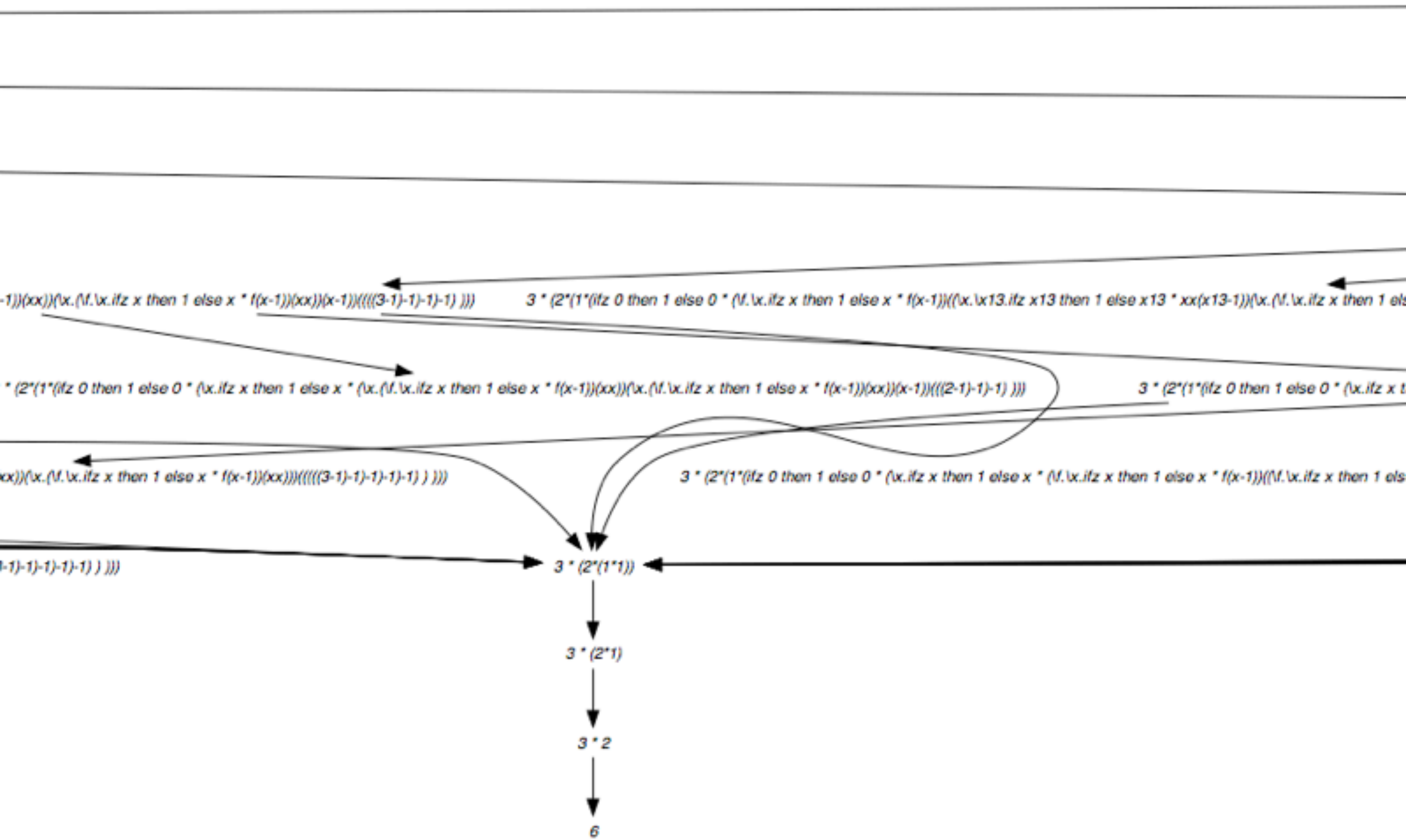


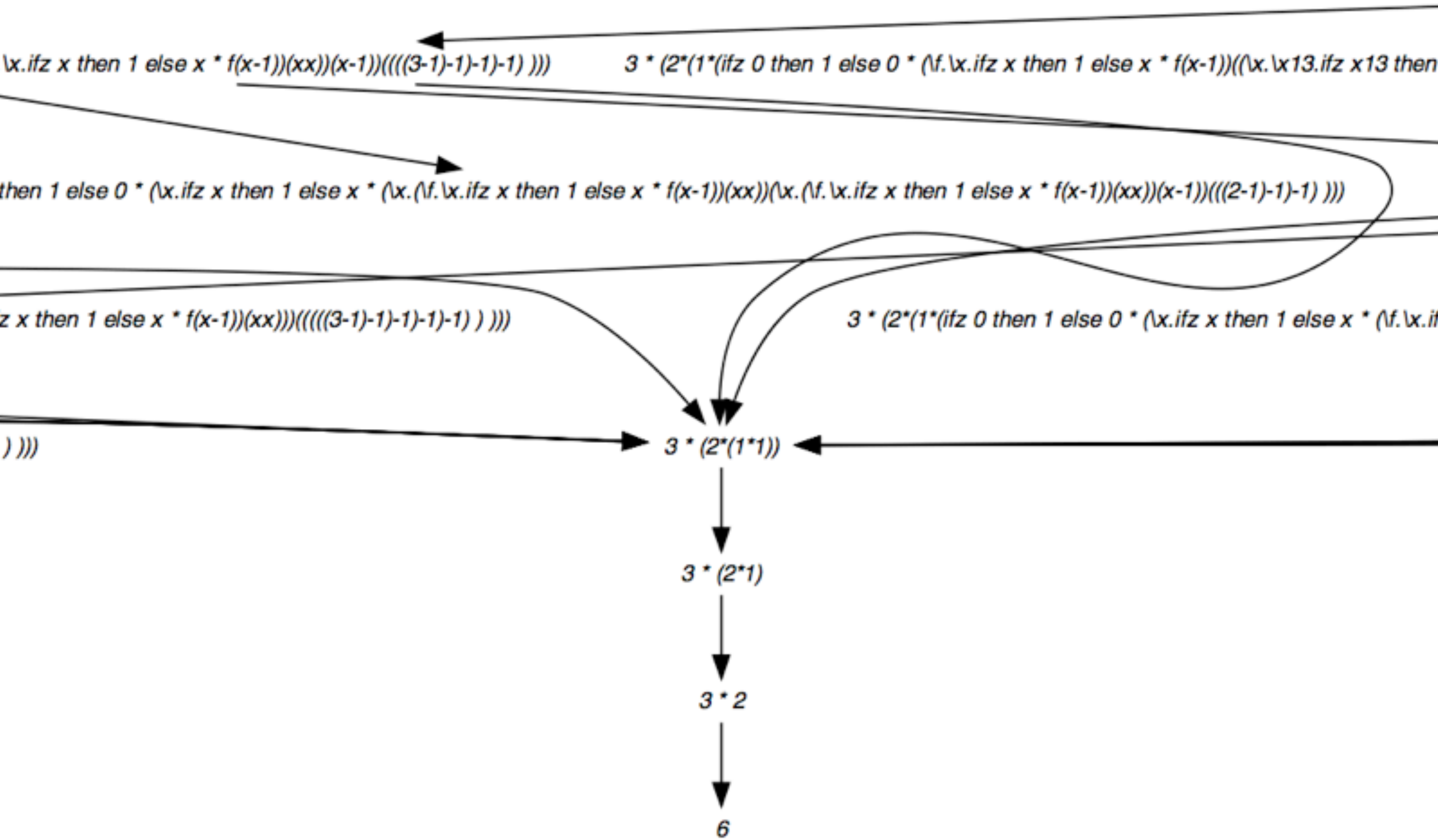



3 * (2*1)
3 * 2
6



3 * (2*(1*1))
3 * (2*1)
3 * 2
6


-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.i
))))
3 * (2*(1*1))
3 * (2*1)
3 * 2
6

λ-definability

Computing without δ-rules

- Numbers will be in **unary**-code

$$\mathbb{N} = 0 \oplus S(\mathbb{N})$$

with following implementation:

$$0 = \langle \texttt{True}, ? \rangle$$

$$1 = \langle \texttt{False}, 0 \rangle = \langle \texttt{False}, \langle \texttt{True}, ? \rangle \rangle$$

$$2 = \langle \texttt{False}, 1 \rangle = \langle \texttt{False}, \langle \texttt{False}, \langle \texttt{True}, ? \rangle \rangle \rangle$$

$$\vdots$$

$$n = \langle \texttt{False}, n - 1 \rangle = \underbrace{\langle \texttt{False}, \langle \texttt{False}, \cdots}_{n} \langle \texttt{True}, ? \rangle \rangle \rangle$$

Computing without δ-rules

- Booleans

$$\texttt{True} = \lambda x.\lambda y.x = K \qquad \texttt{True}\, M\, N \xrightarrow{\star} M$$

$$\texttt{False} = \lambda x.\lambda y.y \qquad \texttt{False}\, M\, N \xrightarrow{\star} N$$

- Pairs and Projections

$$\langle M, N \rangle = \lambda x.xMN \qquad \pi_1 \langle M, N \rangle \xrightarrow{\star} M$$

$$\pi_1 = \lambda x.x\, \texttt{True} \qquad \pi_2 \langle M, N \rangle \xrightarrow{\star} N$$

$$\pi_2 = \lambda x.x\, \texttt{False}$$

- Non-negative integers ...

$$0 = \langle \texttt{True}, \texttt{True} \rangle \qquad \texttt{isZero}\, 0 \xrightarrow{\star} \texttt{True}$$

$$n + 1 = \langle \texttt{False}, n \rangle \qquad \texttt{isZero}(n + 1) \xrightarrow{\star} \texttt{False}$$

$$\texttt{isZero} = \pi_1$$

Computing without δ-rules

- ... integers

$$\mathtt{Succ} = \lambda x.\langle \mathtt{False}, x\rangle$$

$$\mathtt{Pred} = \lambda x.\, \mathtt{isZero}\, x\ 0\ \pi_2$$

Other numeral system

- also named **Church's numerals**

$$n = \lambda f.\lambda x.\underbrace{f(f(\cdots\ f}_{n}(x)\cdots))$$

or

$$n = \lambda f.\ \underbrace{f \circ f \circ \cdots f}_{n}$$

was n+1 in Church's original monograph

Other numeral system

- Lambda-*I* calculus

$\lambda x.M$ (*M* depends upon *x*)

no $K = \lambda x.\lambda y.x$

- Church numerals

$n = \lambda f.\lambda x.f^n(x)$

$n \geq 1$

$n\, I \xrightarrow{\star} I$

$I = \lambda x.x$

- Pairs and projections

$\langle M, N\rangle = \lambda x.xMN$

$\pi_1 = \lambda p.p(\lambda x.\lambda y.\, y\, I\, x)$

$\pi_2 = \lambda p.p(\lambda x.\lambda y.\, x\, I\, y$

$\pi_1\langle m, n\rangle \xrightarrow{\star} m$

$\pi_2\langle m, n\rangle \xrightarrow{\star} n$

Other numeral system

- ... successor and predecessor

$$\texttt{Succ} = \lambda n.\lambda f.\lambda x.n\,f\,(f\,x)$$

$$\texttt{Pred} = \lambda n.\ \pi^3_3\ (n\,\phi\,\langle 1, 1, 1\rangle)$$

$$\phi = \lambda t.(\lambda x.\lambda y.\lambda z.\langle \texttt{Succ}\ x,\ x,\ y\rangle)(\pi^3_1\ t)(\pi^3_2\ t)(\pi^3_3\ t)$$

where $\pi^3_1,\ \pi^3_2,\ \pi^3_3$ are the 3 projections on triples

4	3	2
3	2	1
2	1	1
1	1	1

ϕ

shift register! FIFO

Church numeral system

Alonzo Church

Stephen Kleene

If L, M, N are formulas representing positive integers, then $2_1[M, N]$ conv M, $2_2[M, N]$ conv N, $3_1[L, M, N]$ conv L, $3_2[L, M, N]$ conv M, and $3_3[L, M, N]$ conv N.

Verification of this depends on the observation that, if M is a formula representing a positive integer, MI conv I (the mth power of the identity is the identity).

By the predecessor function of positive integers we mean the function whose value for the argument 1 is 1 and whose value for any other positive integer argument x is $x-1$. This function is λ-defined by

$$P \rightarrow \lambda a . 3_3(a(\lambda b[S(3_1 b), 3_1 b, 3_2 b])[1, 1, 1]).$$

For if K, L, M represent positive integers,

$$(\lambda b[S(3_1 b), 3_1 b, 3_2 b])[K, L, M] \text{ conv } [SK, K, L],$$


2 + 5 = 7
6 − 3 =

Programming languages

CENTRE DE RECHERCHE COMMUN

INRIA MICROSOFT RESEARCH

Towards programming languages

- Many δ-rules

- Adding types → never following terms :

+

= ???

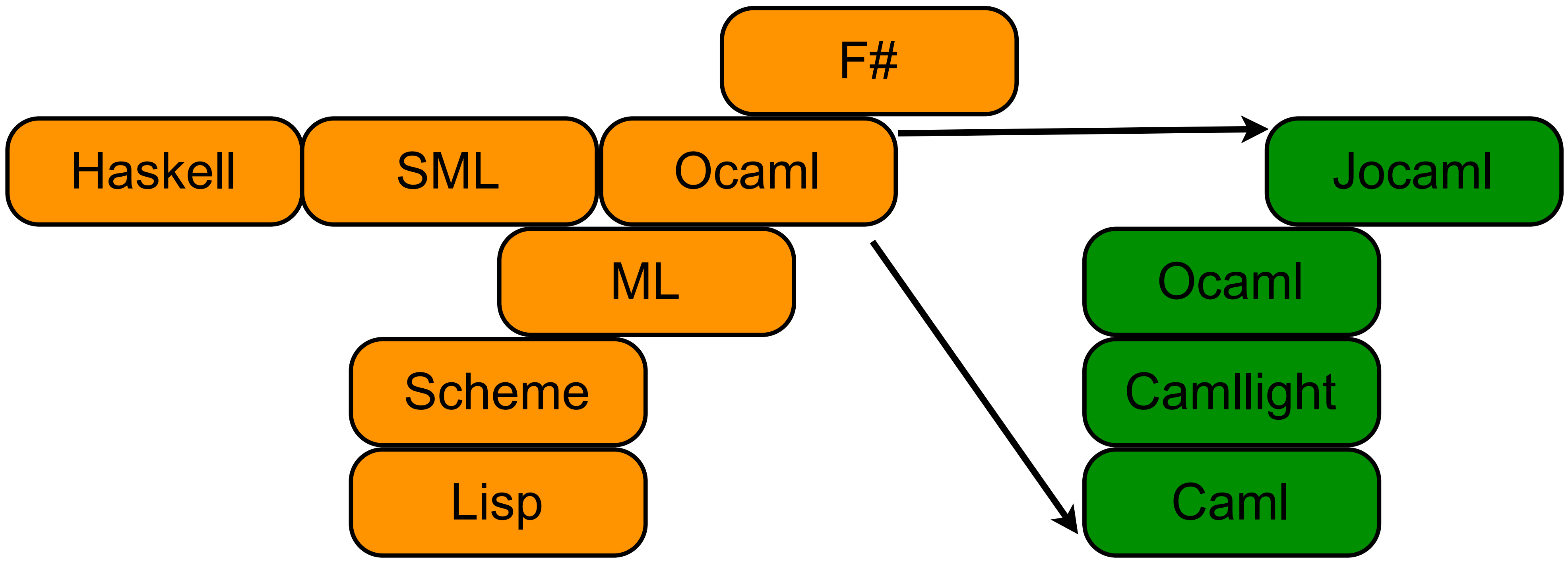
$3 + \lambda x.x$ $\quad$ $4(5)$ $\quad$ $20(\lambda x.x)$ $\quad$ $\texttt{ifz}\ \lambda x.x\ \texttt{then}\ 1\ \texttt{else}\ 3$

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

- Adding store and mutable values

Functional programming

- Scheme, SML, Ocaml, Haskell are functional programming languages
- they manipulate functions
- and try to reduce the number of memory states



Next class

CENTRE DE RECHERCHE COMMUN

INRIA MICROSOFT RESEARCH

Next class

- confluency

- consistency