5th Asian-Pacific Summer School on Formal Methods

August 5-10, 2013, Tsinghua University, Beijing, China

Functions

jean-jacques.levy@inria.fr

August 5, 2013

http://sts.thss.tsinghua.edu.cn/Coqschool2013

Notes adapted from
Assia Mahboubi
(coq school 2010, Paris) and
Benjamin Pierce (software
foundations course, UPenn)

Plan

- functions and λ-notation
- higher-order functions
- data types
- notation in Coq
- enumerated sets
- pattern-matching on constructors

Functions and λ-notation

CENTRE DE RECHERCHE COMMUN
INRIA
MICROSOFT RESEARCH

Functional calculus (1/6)

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

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

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

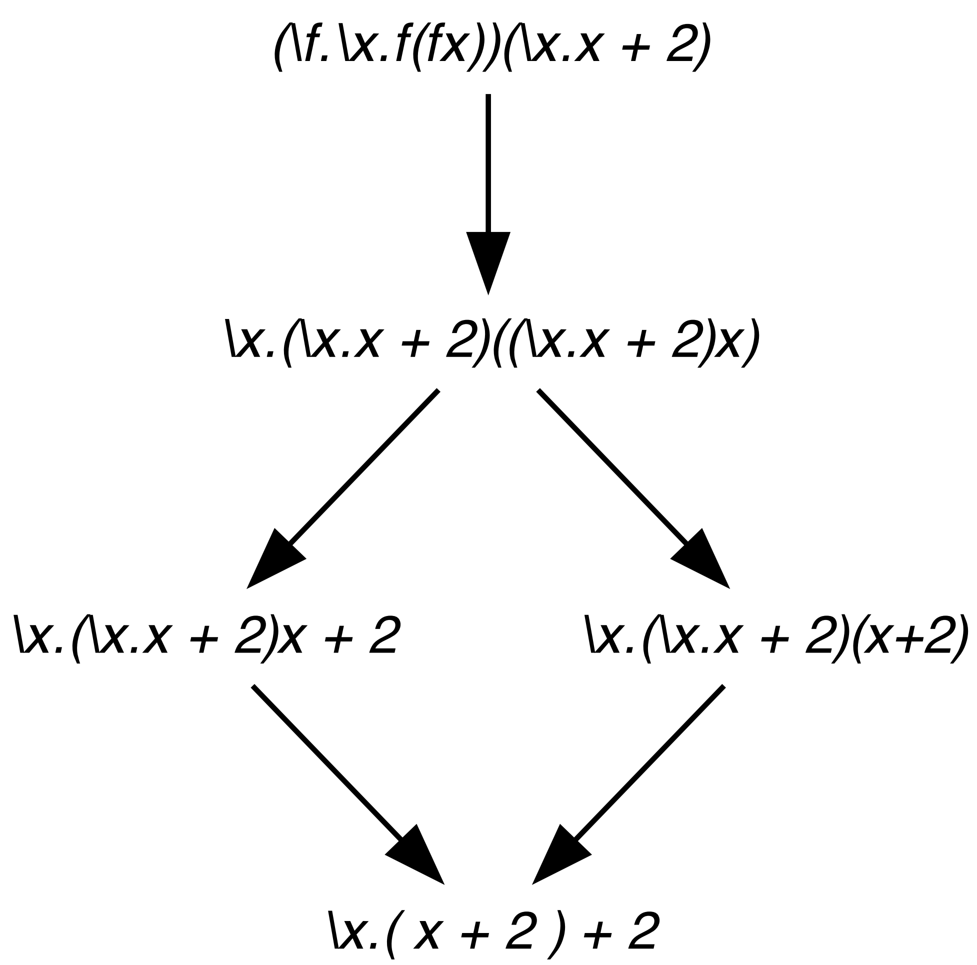
$(\lambda x.\lambda y.x+y)3\ 2 =$

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

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

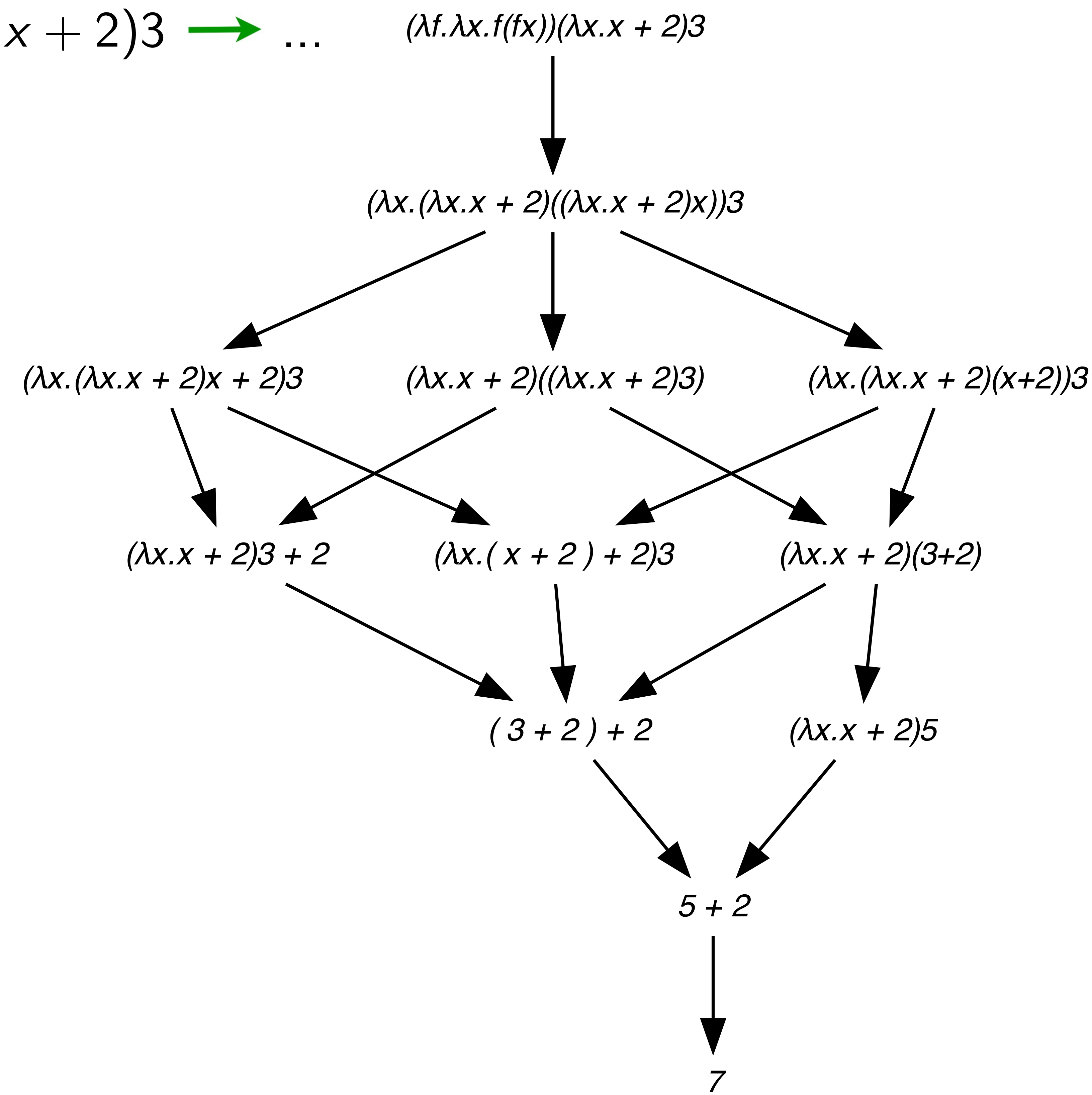
Functional calculus (2/6)

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

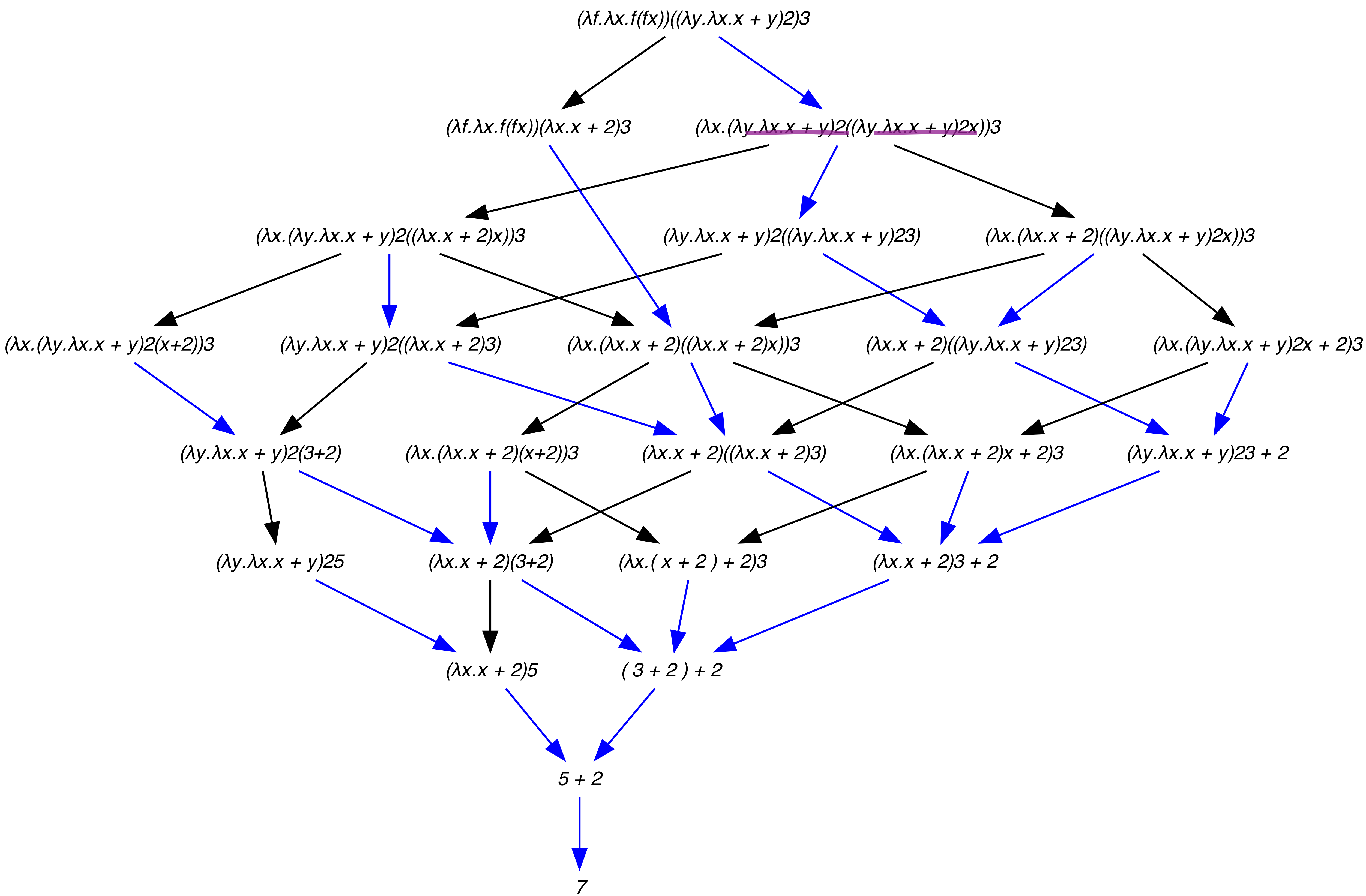


Functional calculus (3/6)

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



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



Functional calculus (5/6)

$\texttt{Fact}(3)$

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

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

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

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

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

*(\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)*

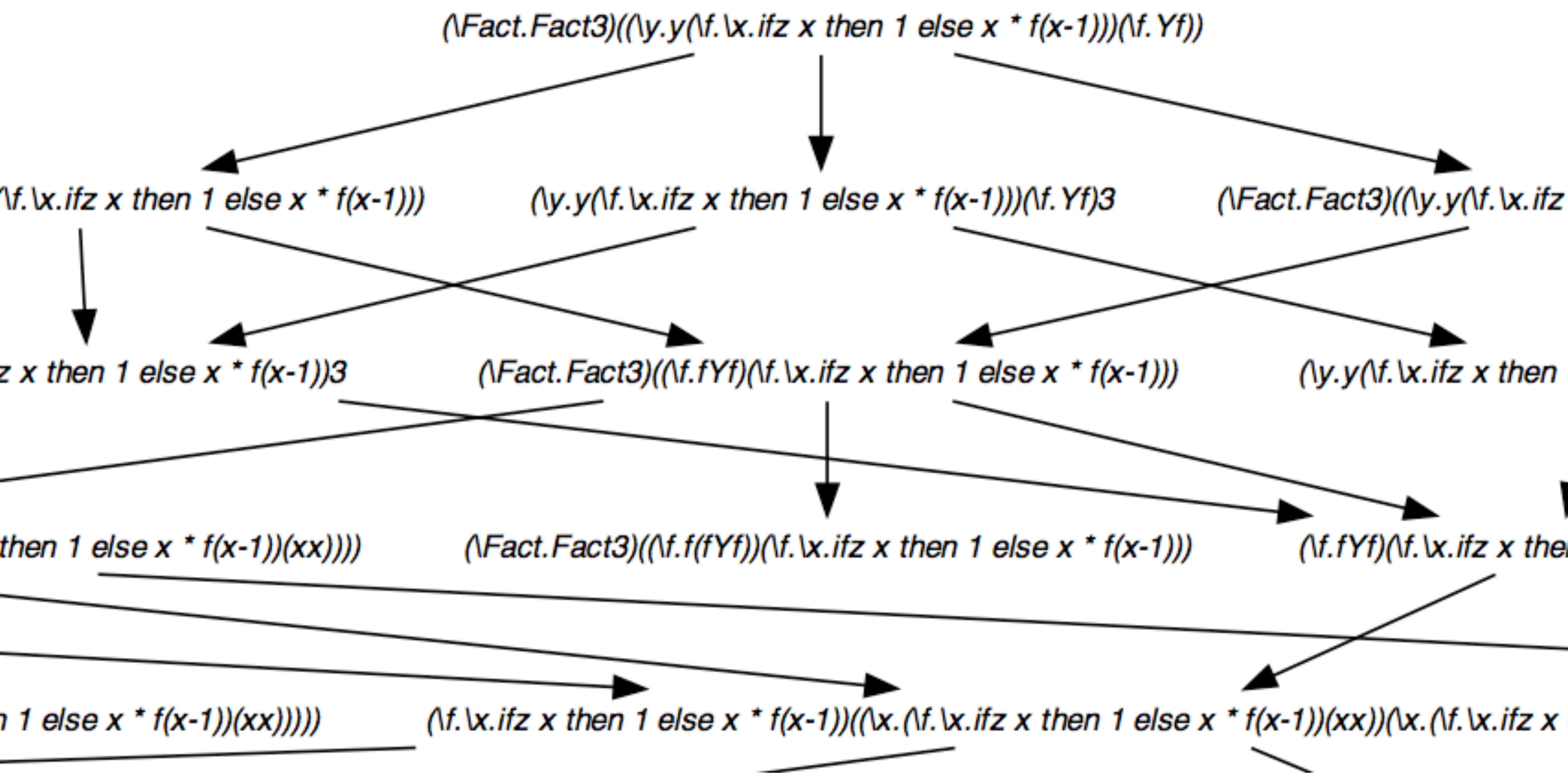
↓

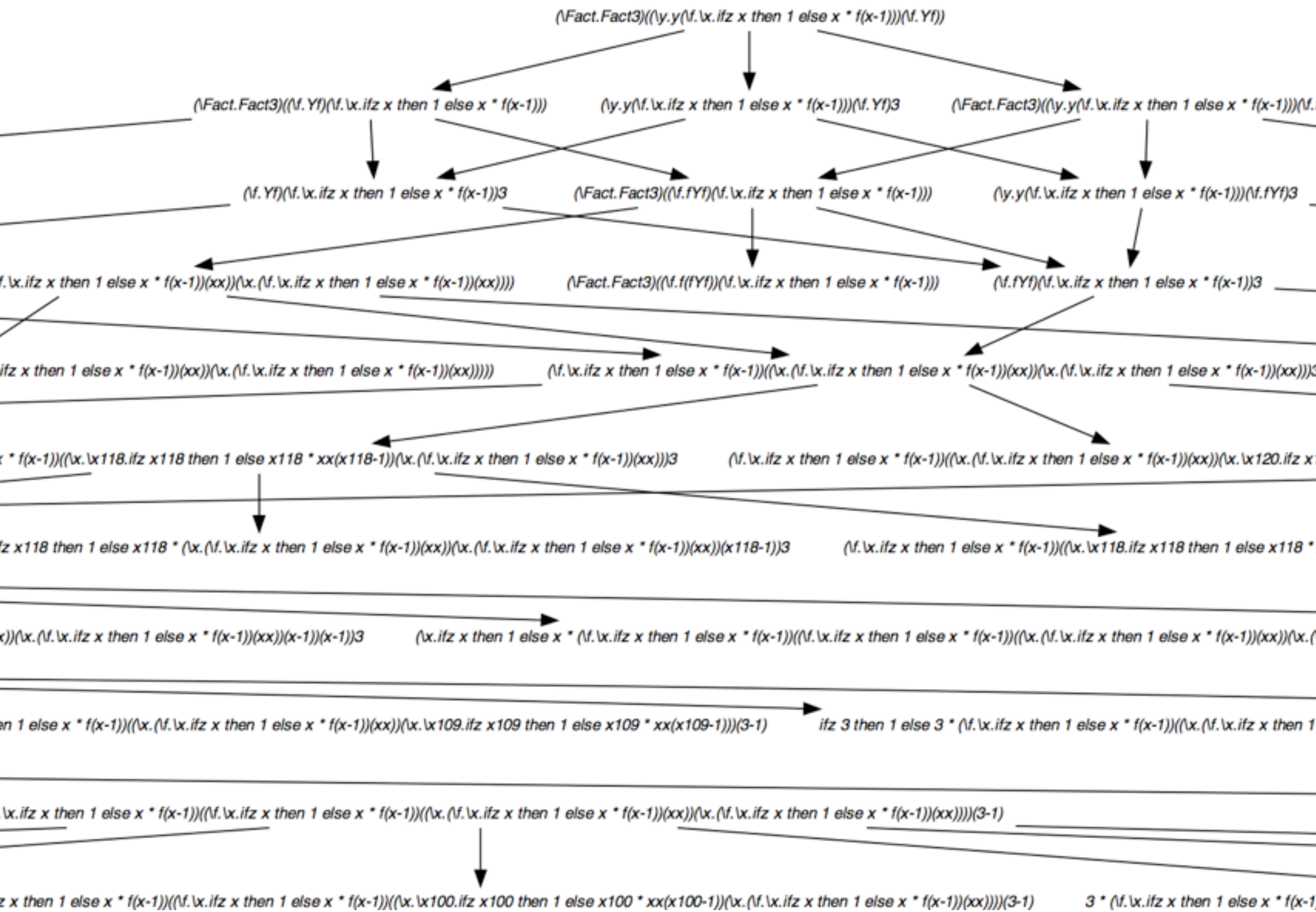
*3 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(3-1)*

↓

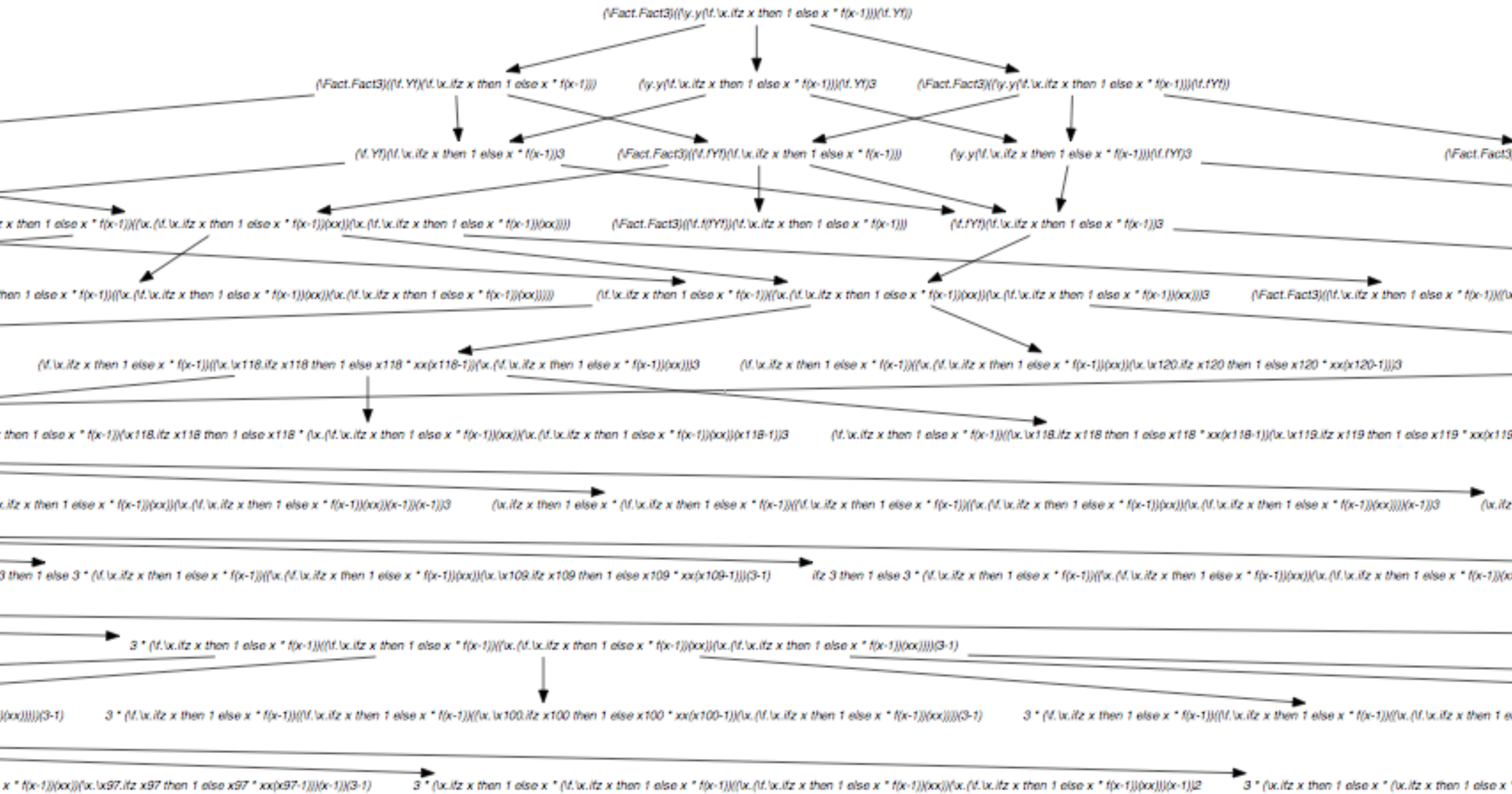
*3 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(3-1)*

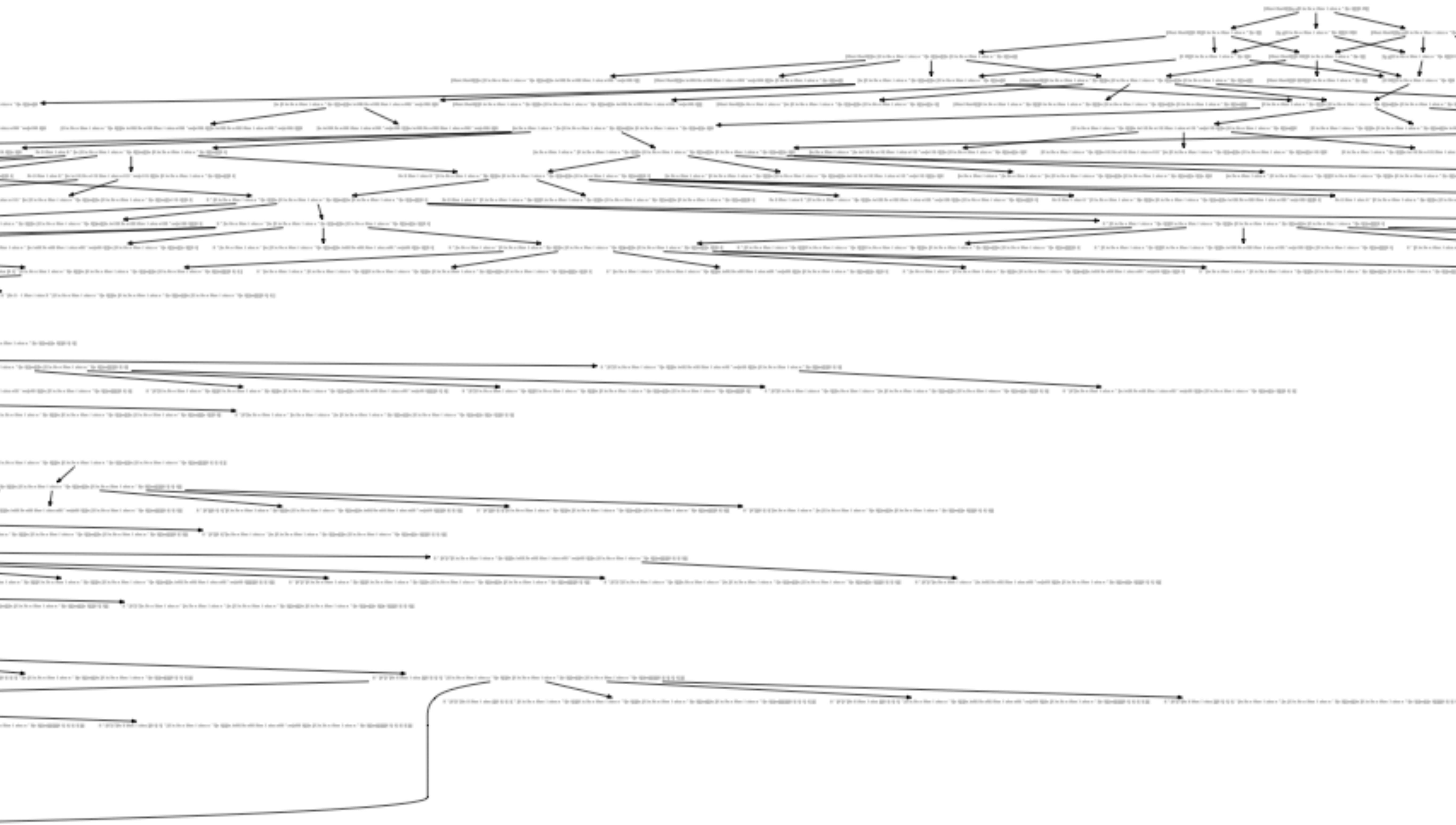
↓

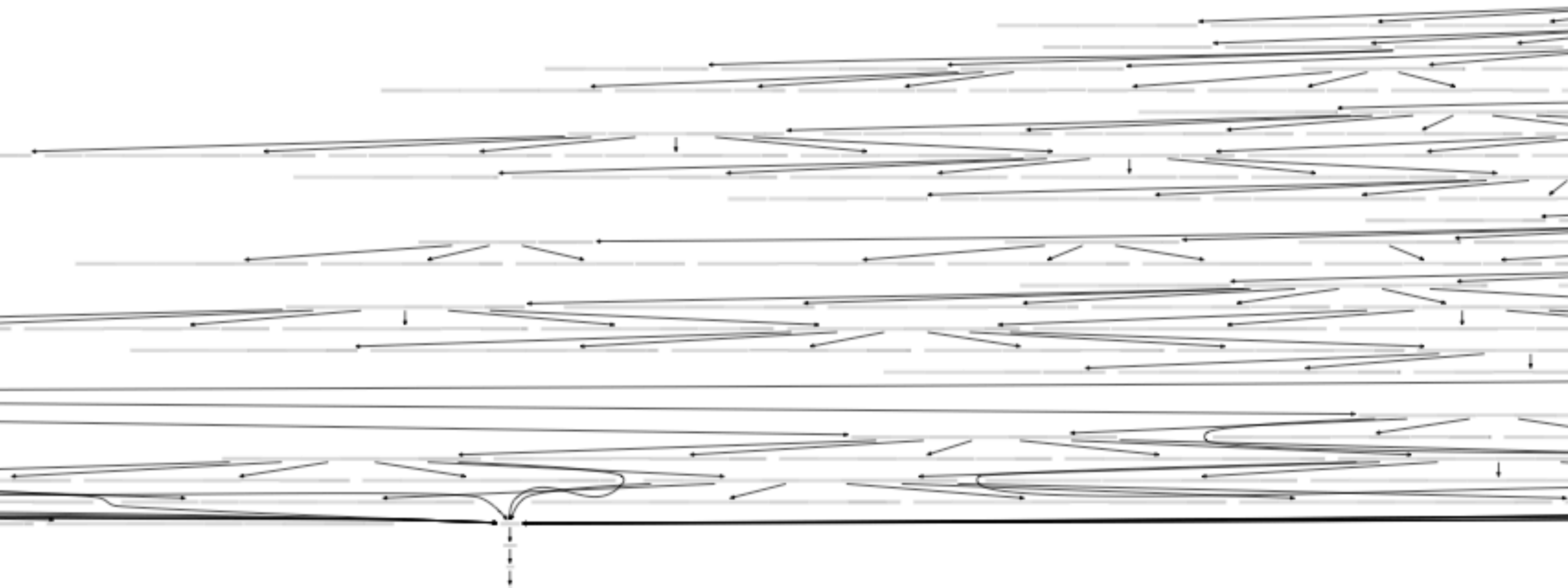

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x.ifz
z x then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then
then 1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

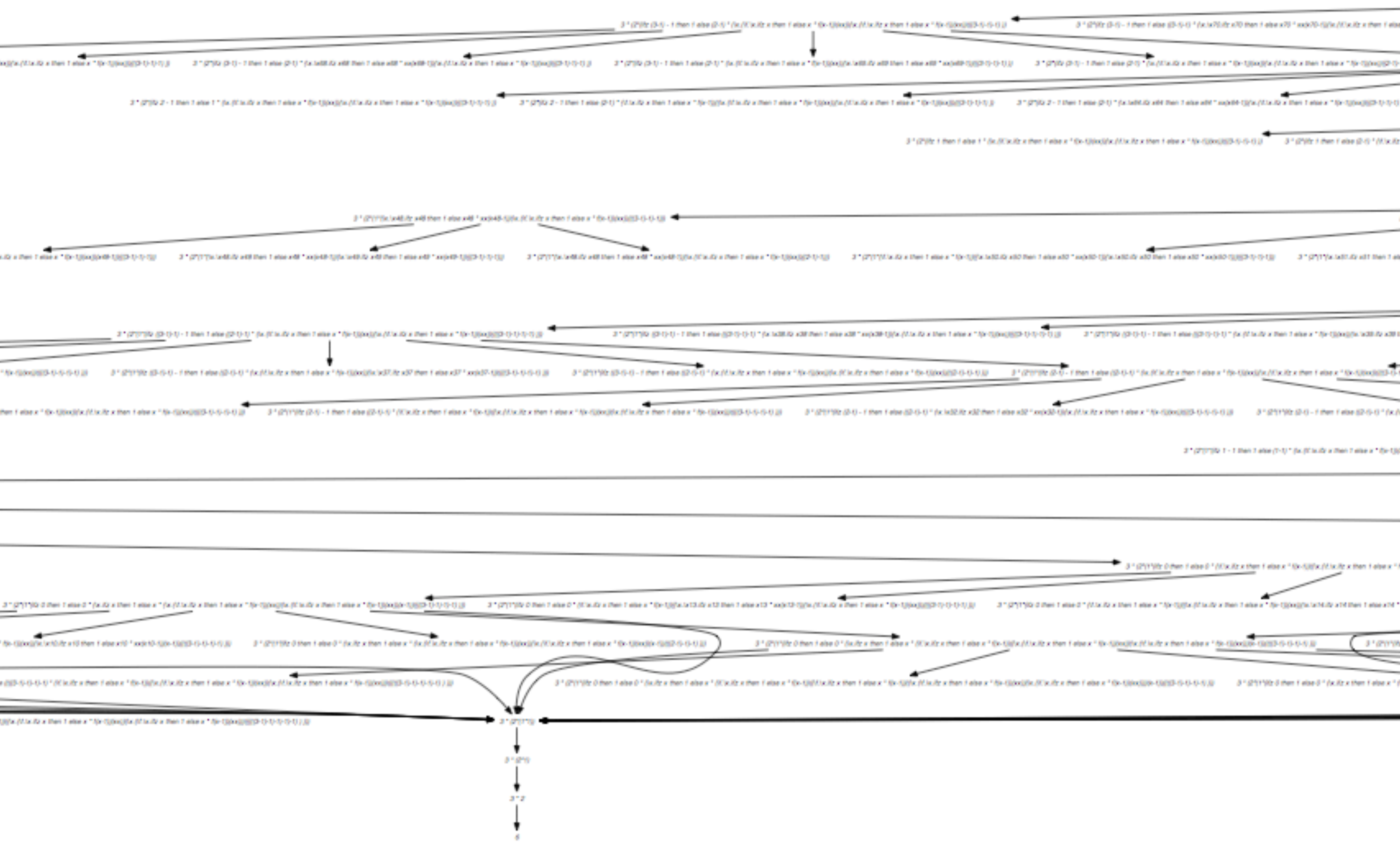


(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
(\Fact.Fact3)((\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then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

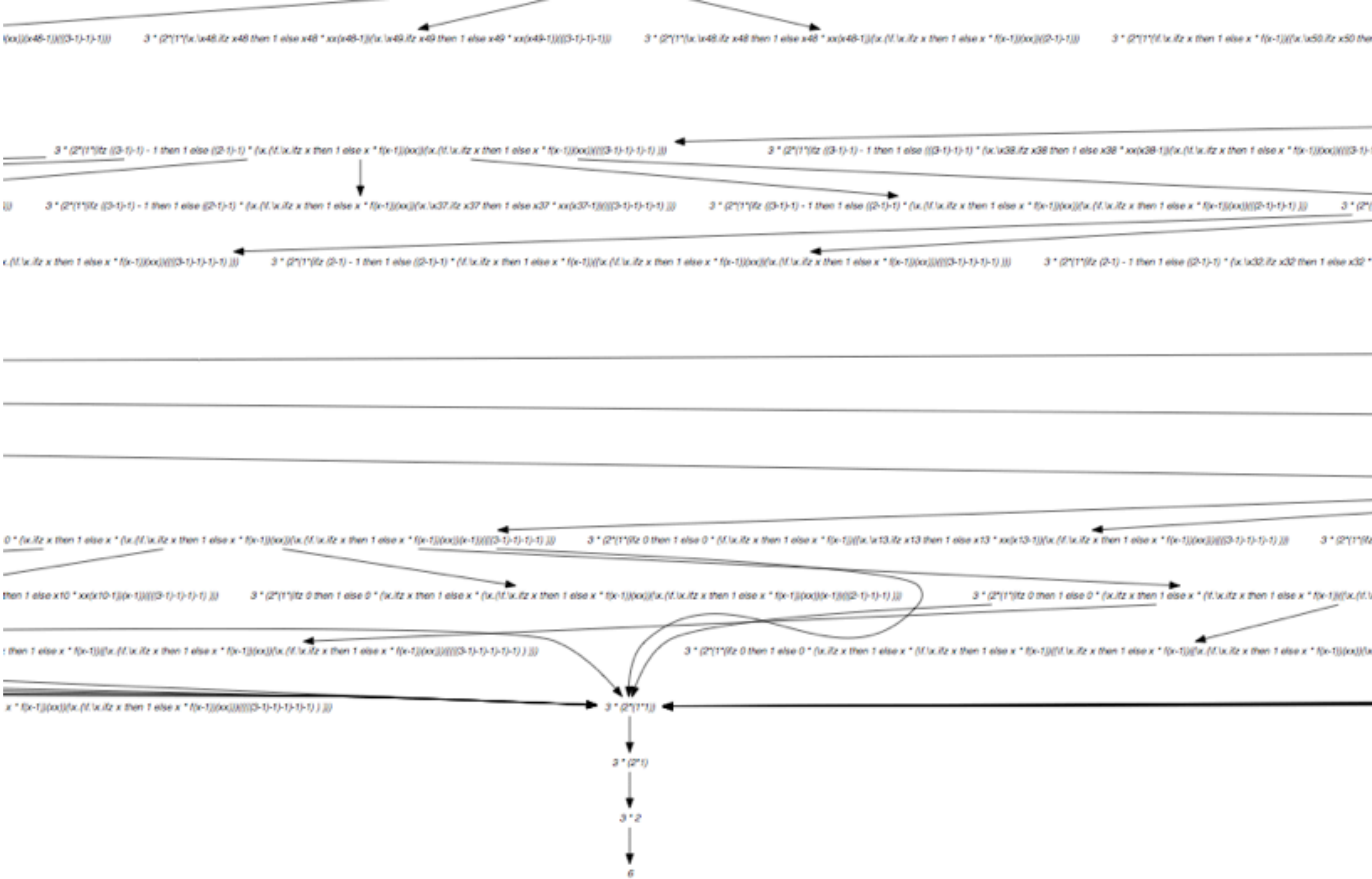

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
(\Fact.Fact3)((\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.fYf))
(\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz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

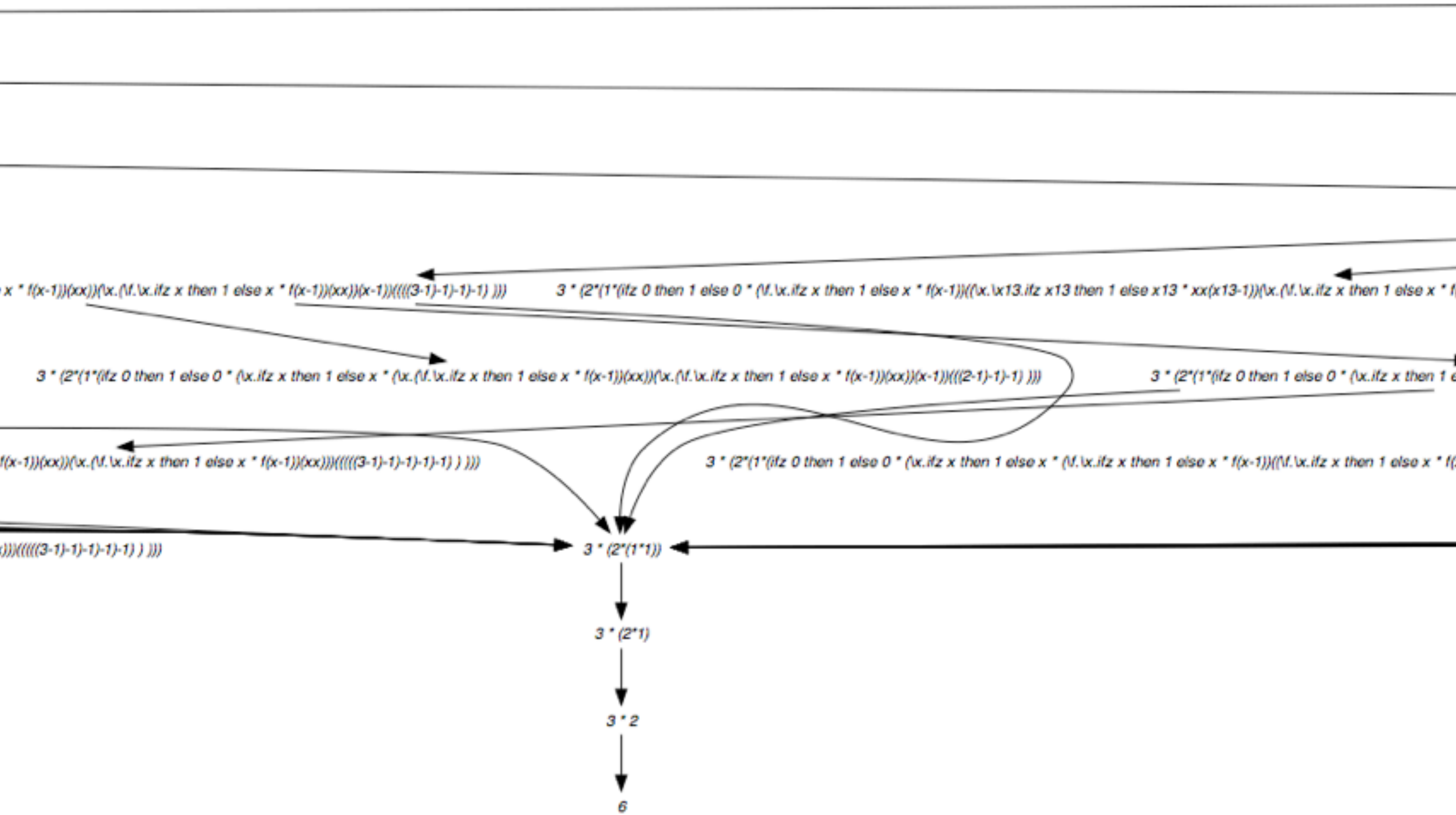


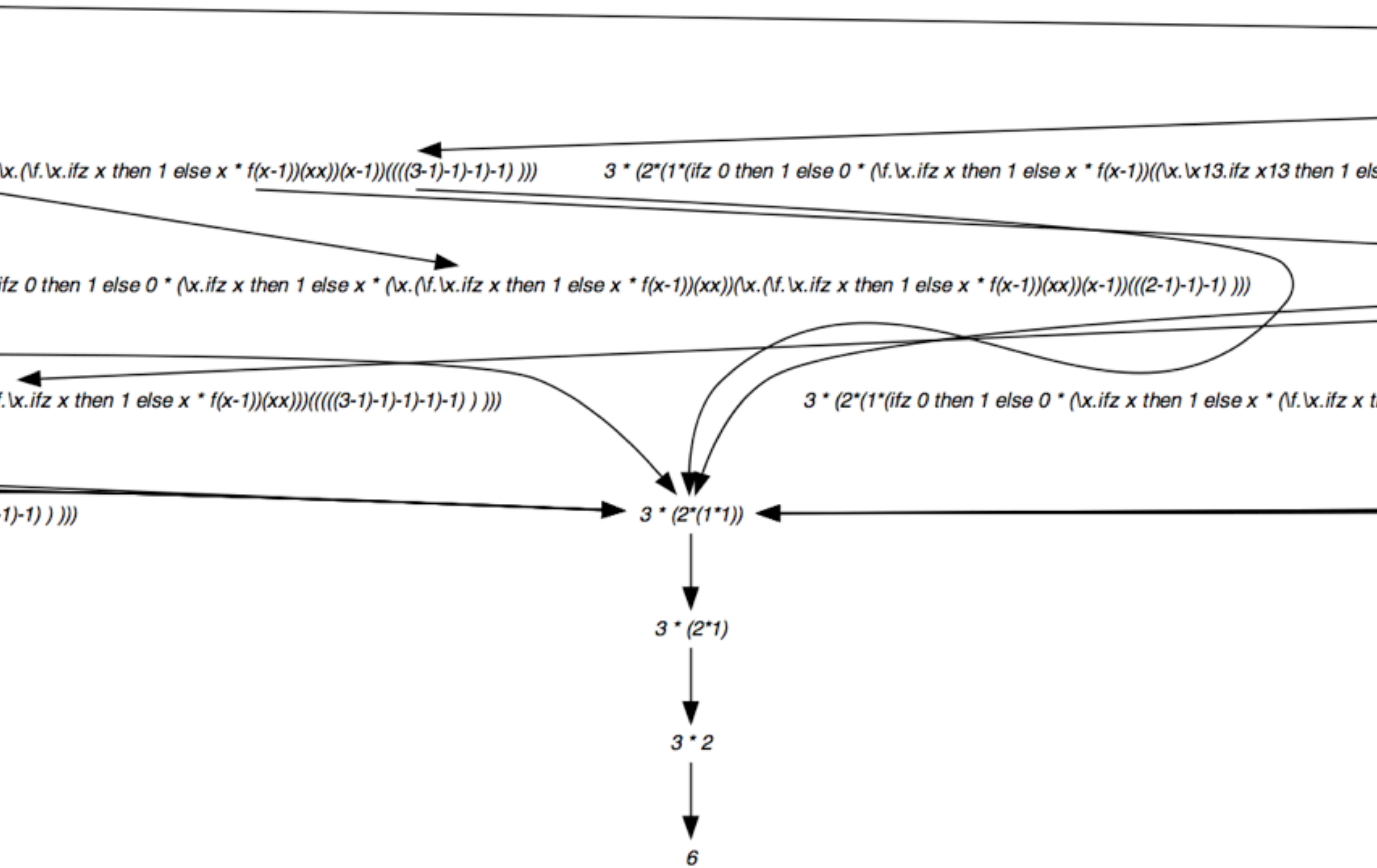




3 * (2*1)
3 * 2
6


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


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 else x * f
3 * (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 then 1 else x * (\f.\x.ifz x then 1 else x * f(x-1))((\f.\x.ifz x then 1 else x * f
3 * (2*(1*1))
3 * (2*1)
3 * 2
6



\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 els
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))))
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 t
-1)-1)))))
3 * (2*(1*1))
3 * (2*1)
3 * 2
6

λ-calculus

CENTRE DE RECHERCHE COMMUN

INRIA MICROSOFT RESEARCH

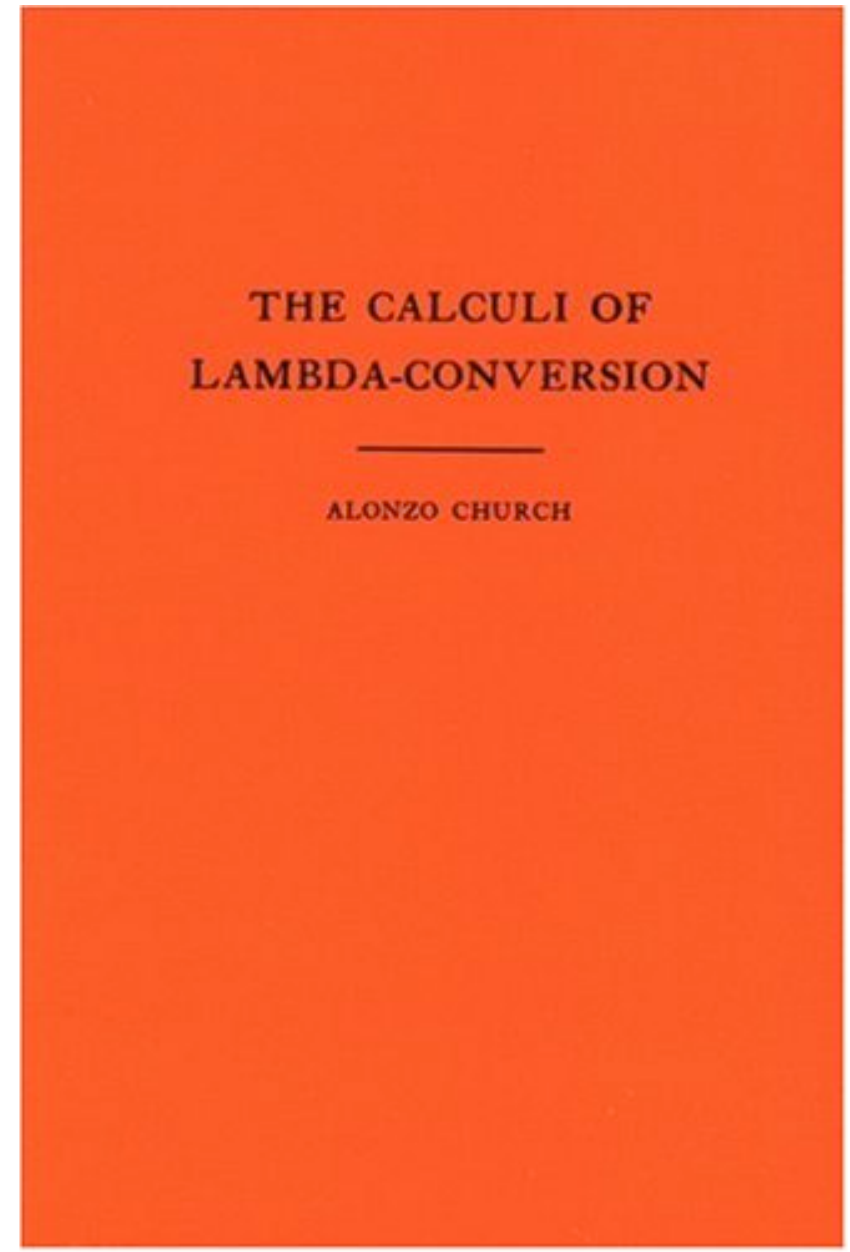
Pure lambda-calculus

- 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 (2/2)

- 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 λ-term

→ Church's thesis. [Church 41]

Fathers of computability

Alonzo Church

Stephen Kleene

CENTRE DE RECHERCHE COMMUN
INRIA MICROSOFT RESEARCH

The Giants of computability

Hilbert → Gödel → Church → Turing

Kleene

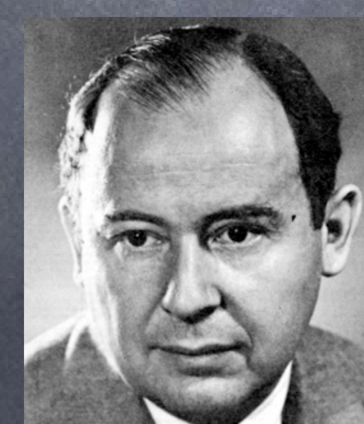
Post Curry

von Neumann










Typed lambda-calculus (1/5)

- In Coq, all λ-terms are typed

- In Coq, following λ-terms are typable

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

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

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

$$(\lambda x.\lambda y.x+y)3\ 2 =$$

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

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

these terms are allowed

Typed lambda-calculus (2/5)

- In Coq, all λ-terms have only finite reductions

`(strong normalization property)`

- In Coq, all λ-terms have a (unique) normal form.

- In Coq, the following λ-terms are not typable

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

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

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

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

these terms are not allowed

Typed lambda-calculus (3/5)

- The Coq laws for typing terms are quite complex [Coquand-Huet 1985]

- In first approximation, they are the following (1st-order) rules:

  Basic types: $\mathcal{N}$ (nat), $\mathcal{B}$ (bool), $\mathcal{Z}$ (int), ...

  If x has type α, then $(\lambda x.M)$ has type $\alpha \to \beta$

  If M has type $\alpha \to \beta$, then $M(N)$ has type β

Example

$1 : \mathtt{nat}$

$x : \mathtt{nat}$ implies $x + 1 : \mathtt{nat}$

$(\lambda x.\, x + 1) : \mathtt{nat} \to \mathtt{nat}$

$3 : \mathtt{nat}$

$(\lambda x.\, x + 1)3 : \mathtt{nat}$

Typed lambda-calculus (4/5)

Example

$$x : \texttt{nat} \vdash x : \texttt{nat}$$

$$\frac{x : \texttt{nat} \vdash x : \texttt{nat} \qquad 1 : \texttt{nat}}{x : \texttt{nat} \vdash x + 1 : \texttt{nat}}$$

$$\frac{x : \texttt{nat} \vdash x + 1 : \texttt{nat}}{\vdash (\lambda x.\, x + 1) : \texttt{nat} \to \texttt{nat}}$$

$$\frac{\vdash (\lambda x.\, x + 1) : \texttt{nat} \to \texttt{nat} \qquad 3 : \texttt{nat}}{\vdash (\lambda x.\, x + 1)3 : \texttt{nat}}$$

Typed lambda-calculus (5/5)

Example with currying and function as result

λ-calculus in Coq

CENTRE DE RECHERCHE COMMUN
INRIA MICROSOFT RESEARCH

lambda-terms (1/3)

three equivalent definitions:

```
Definition plusOne (x: nat) : nat := x + 1.
Check plusOne.

Definition plusOne := fun (x: nat) => x + 1.
Check plusOne.

Definition plusOne := fun x => x + 1.
Check plusOne.
```

```
Compute (fun x:nat => x + 1) 3.
```

higher-order definitions:

```
Definition plusTwo (x: nat) : nat := x + 2.

Definition twice := fun f => fun (x:nat) => f (f x).

Compute twice plusTwo 3.
```

lambda-terms (2/3)

- Coq tries to guess the type, but could fail.

`(type inference)`

- but always possible to give explicit types.

- Types can be higher-order

`(see later with polymorphic functions)`

- Types can also depend on values

`(see later the constructor cases)`

lambda-terms (3/3)

- Coq treats with an extention of the λ-calculus with inductive data types. It’s a programming language.

- the typed λ-calculus is also used as a trick to make a correspondance between proofs and λ-terms and propositions and types for constructive logics (see other lectures).

`(Curry-Howard correspondance)`