A visualiser for linear lambda-terms as rooted 3-valent maps

George Kaye

University of Birmingham

CLA’2019, July 1

Outline

- **Background**
- Motivation
- Demo
- Future work

The lambda calculus

A model of computation where programs are expressed using three constructs:

x	variable
$\lambda x.t$	abstraction
$t\ u$	application

The lambda calculus

Variables can be **bound** or **free**

$$\lambda x.\ x\ y$$

bound free

The lambda calculus

Terms that differ only by labels of variables are **α-equivalent**
We can rename terms using **α-conversion**

$$\lambda x.\lambda y.\ x\ y \ \rightarrow_{\propto}\ \lambda a.\lambda b.\ a\ b$$

The lambda calculus

Alternatively, we can use **de Bruijn indices** to represent the number of lambdas between a variable and where it was initially abstracted

$$\lambda x.\lambda y.\ x\ y \quad \equiv \quad \lambda\ \lambda\ 1\ 0$$

This eliminates the need for $\propto$-conversion

The lambda calculus

Function application is performed by **β-reduction** on **β-redexes**:

$$(\lambda x.\ x)\ a \quad \to_\beta \quad x\,[x \mapsto a] \quad \equiv \quad a$$

- Repeatedly performing β-reduction is called **normalisation**
- A term with no β-redexes is in its **normal form**

The lambda calculus

- Every term has a single normal form
- But there can be many different ways of reaching it
- These represent different **reduction strategies**
- We can represent this with a **normalisation graph**

The lambda calculus

$$
\begin{tikzcd}
 & (\lambda x.\ x)(\lambda y.\ (\lambda z.\ z)\ y) \arrow[dl] \arrow[dr] & \\
(\lambda y.\ (\lambda z.\ z)\ y) \arrow[dr] & & (\lambda x.\ x)\ (\lambda y.\ y) \arrow[dl] \\
 & \lambda y.\ y &
\end{tikzcd}
$$

The lambda calculus

Some terms do not have a computable normal form

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

$$\to_\beta \quad x\ x\ [x \mapsto (\lambda x.\ x\ x)]$$

$$\equiv (\lambda x.\ x\ x)(\lambda x.\ x\ x)$$

The lambda calculus

Some terms do not have a computable normal form

But they may still have a finite normalisation graph!

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

Fragments of the lambda calculus

- The **pure** lambda calculus contains all terms
- The **linear** lambda calculus contains terms in which each variable is used exactly once
- The **planar** lambda calculus contains linear terms in which each variable is used in the order of abstraction

Fragments of the lambda calculus

$\lambda x.\ x$

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

$\lambda x.\lambda y.\ x\ y$

$\lambda x.\lambda y.\ y\ x$

$\lambda x.\ x\ x$

$\lambda x.\lambda y.\ x$

Fragments of the lambda calculus

λx. x

λx. (λy. y) x

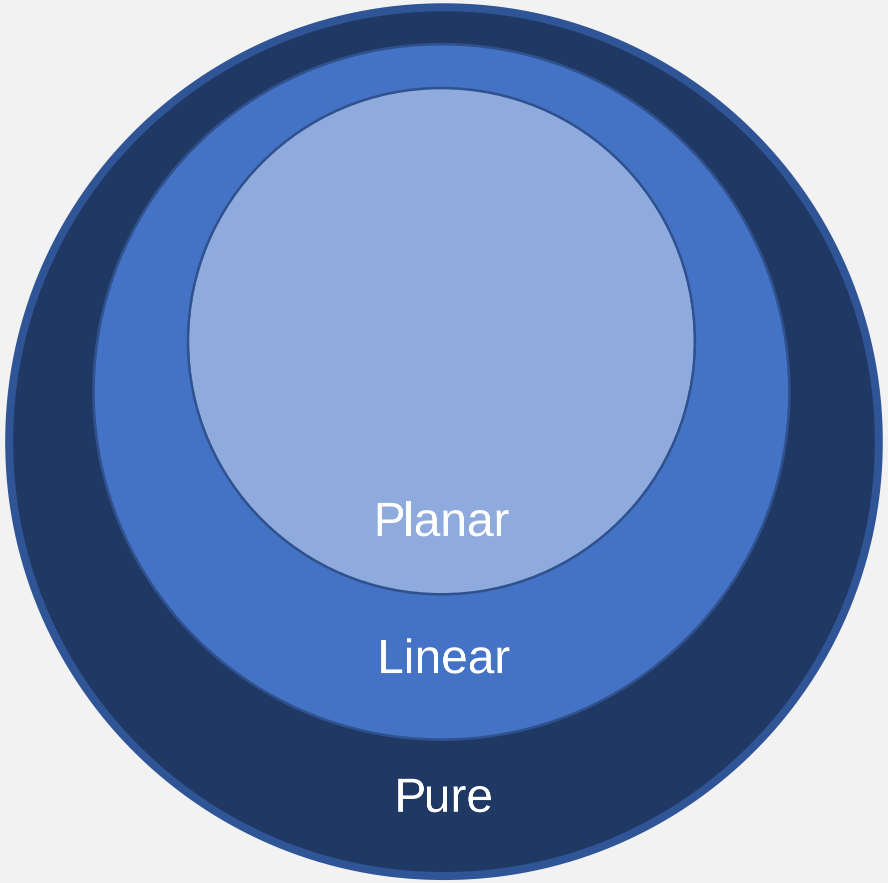
λx.λy. x y

λx.λy. y x

λx. x x

λx.λy. x

Fragments of the lambda calculus



$\lambda x.\ x$

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

$\lambda x.\lambda y.\ x\ y$

$\lambda x.\lambda y.\ y\ x$

$\lambda x.\ x\ x$

$\lambda x.\lambda y.\ x$

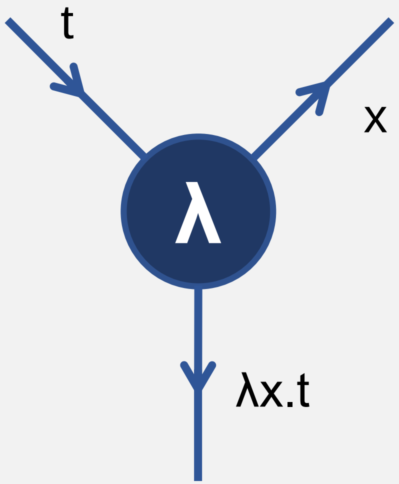
Fragments of the lambda calculus

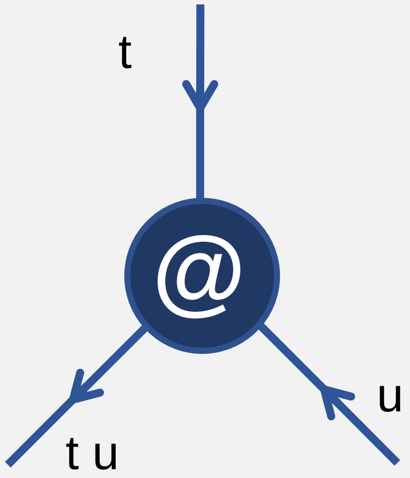
- Linear (and planar) terms have **special properties**
- Linearity and planarity are **preserved** by normalisation
- All linear terms have a computable normal form
- Normalisation of linear terms is **efficient**
- Computing the normal form of a linear term is PTIME-complete

Linear lambda calculus and PTIME-completeness (Mairson, 2004)

- All paths to the normal form of a linear term are the same length

Lambda-terms as rooted maps

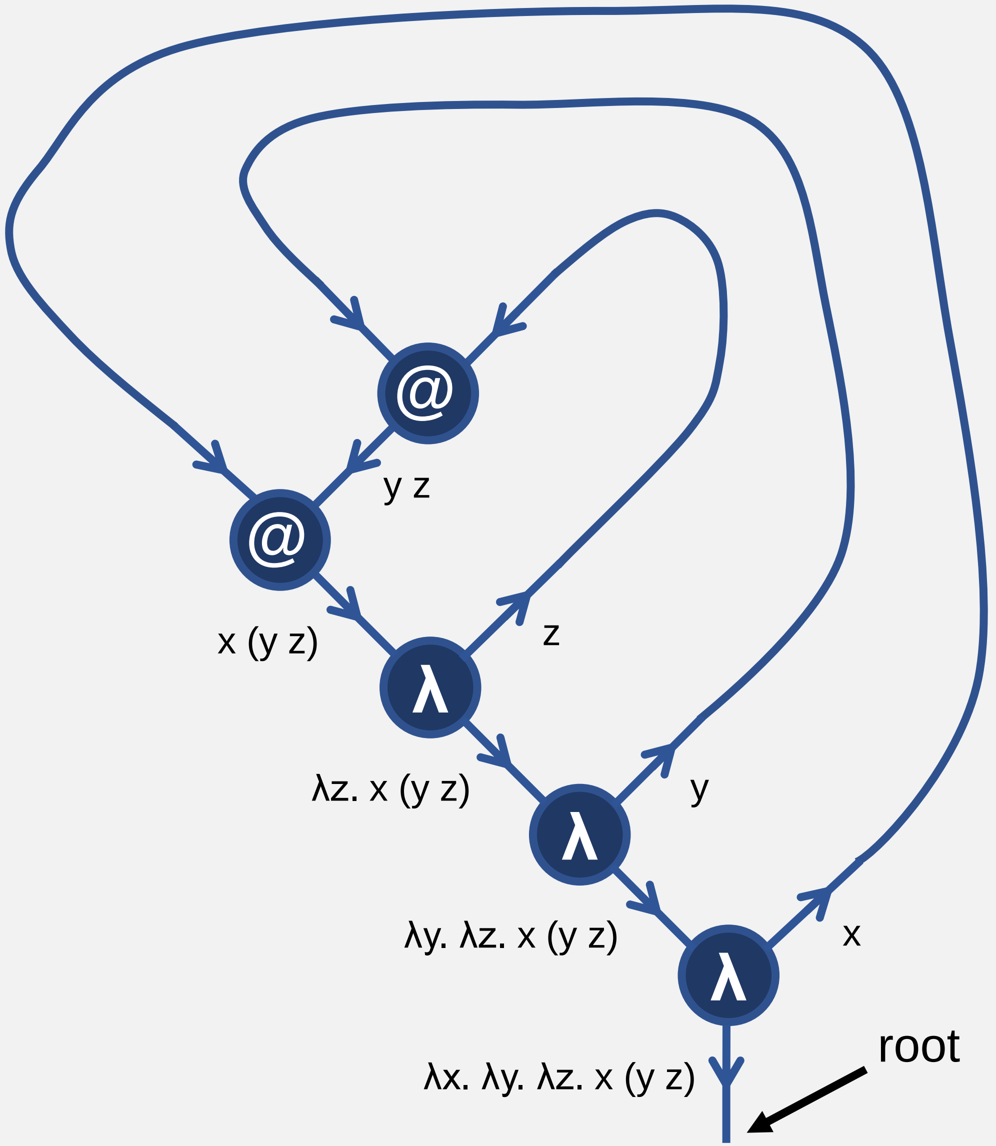




We can build up term maps by combining these nodes and a special node called the **root**, which represents the complete term

Lambda-terms as rooted maps

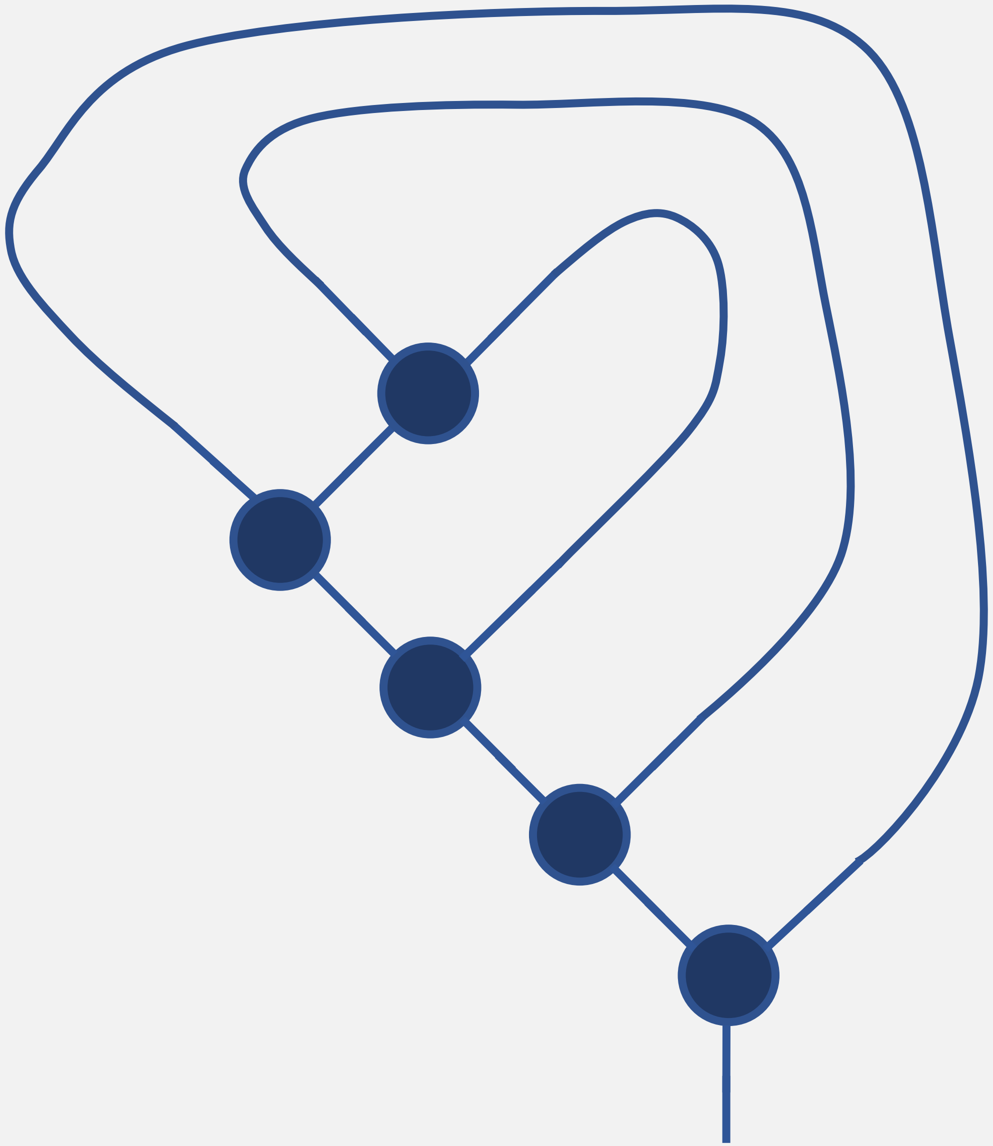
$$\lambda x.\ \lambda y.\ \lambda z.\ x\ (y\ z)$$



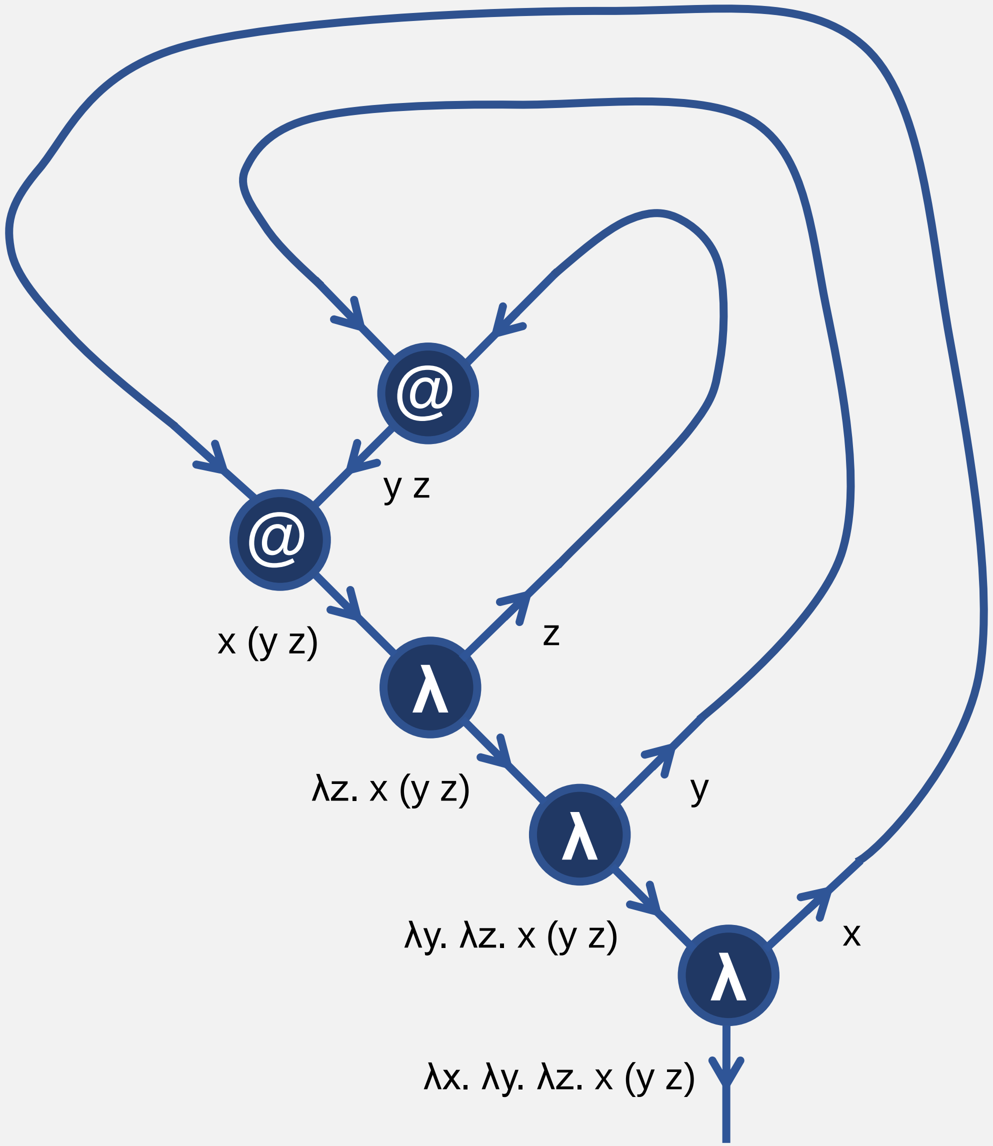
Lambda-terms as rooted maps

λx. λy. λz. x (y z)

Removing the labels and arrows turns this into a **rooted map**



Lambda-terms as rooted maps



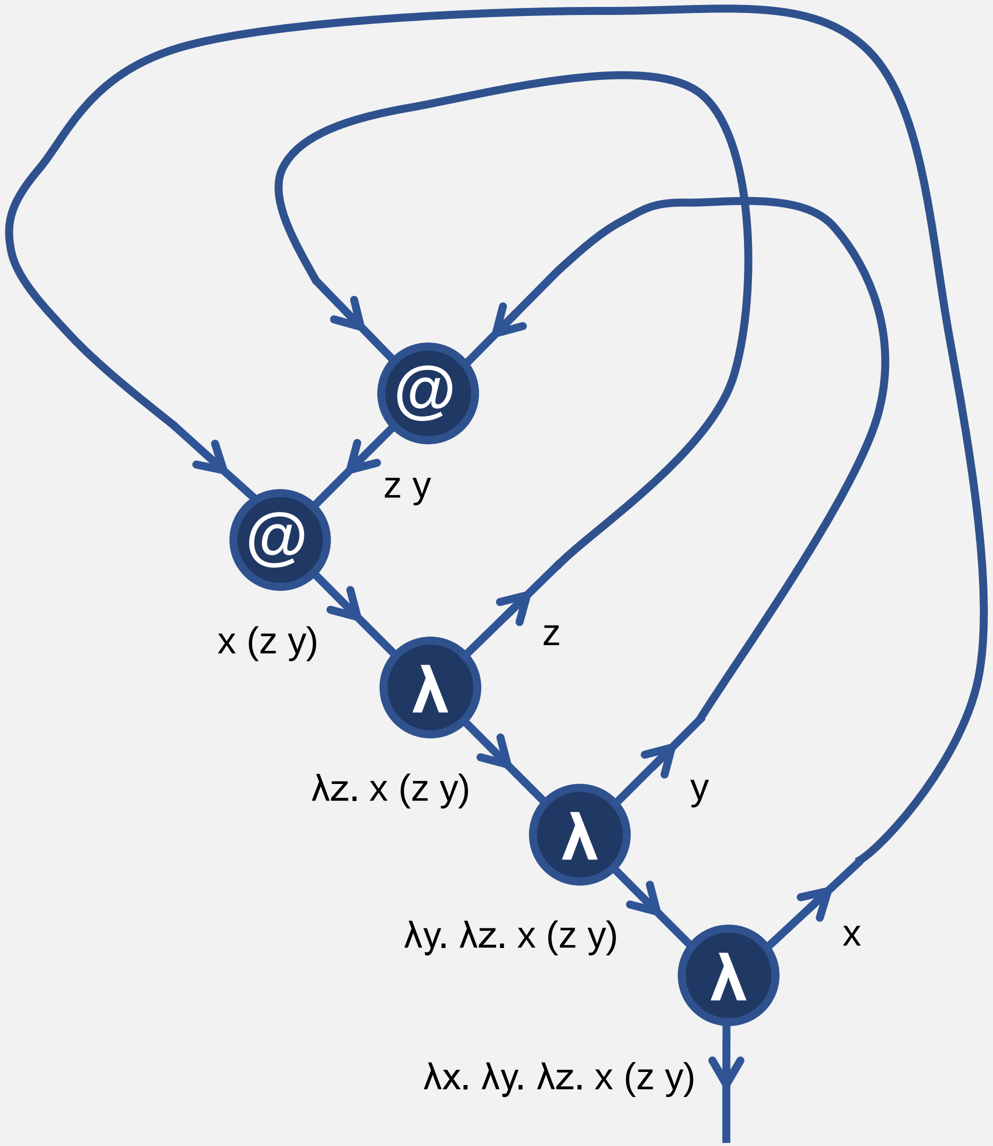
λx. λy. λz. x (y z)

This term is
linear: the map is 3-valent
planar: there are no crossings

Lambda-terms as rooted maps

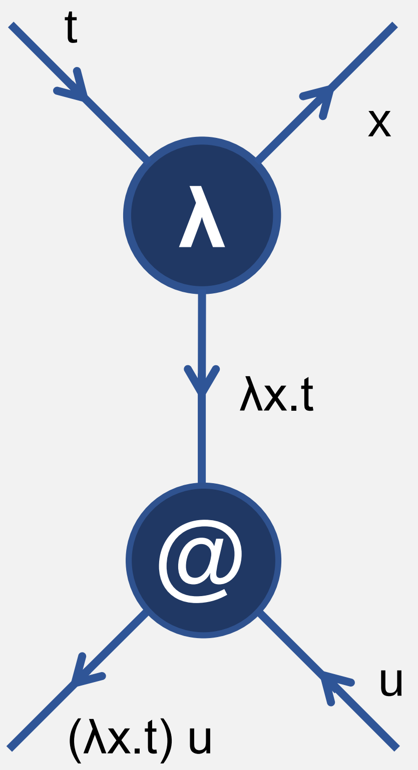
$\lambda x.\ \lambda y.\ \lambda z.\ x\ (z\ y)$

This term is
linear: the map is 3-valent
non-planar: there is one crossing



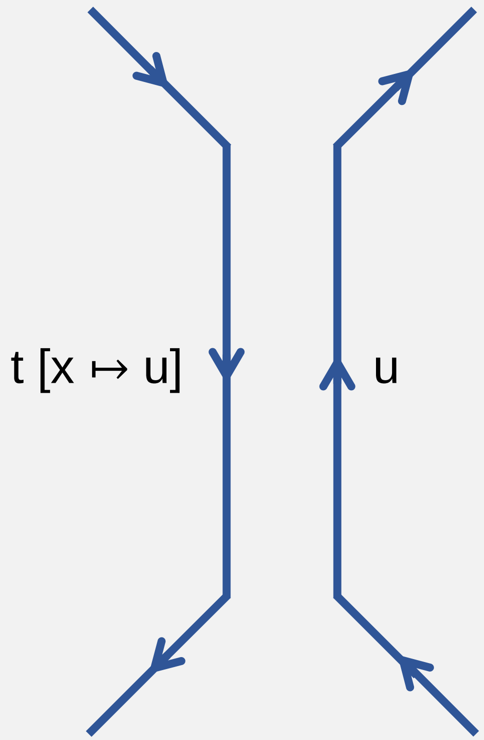
Beta reduction

(λx.t) u



Beta reduction

$t\ [x \mapsto u]$



Outline

- Background
- **Motivation**
- Demo
- Future work

Motivation

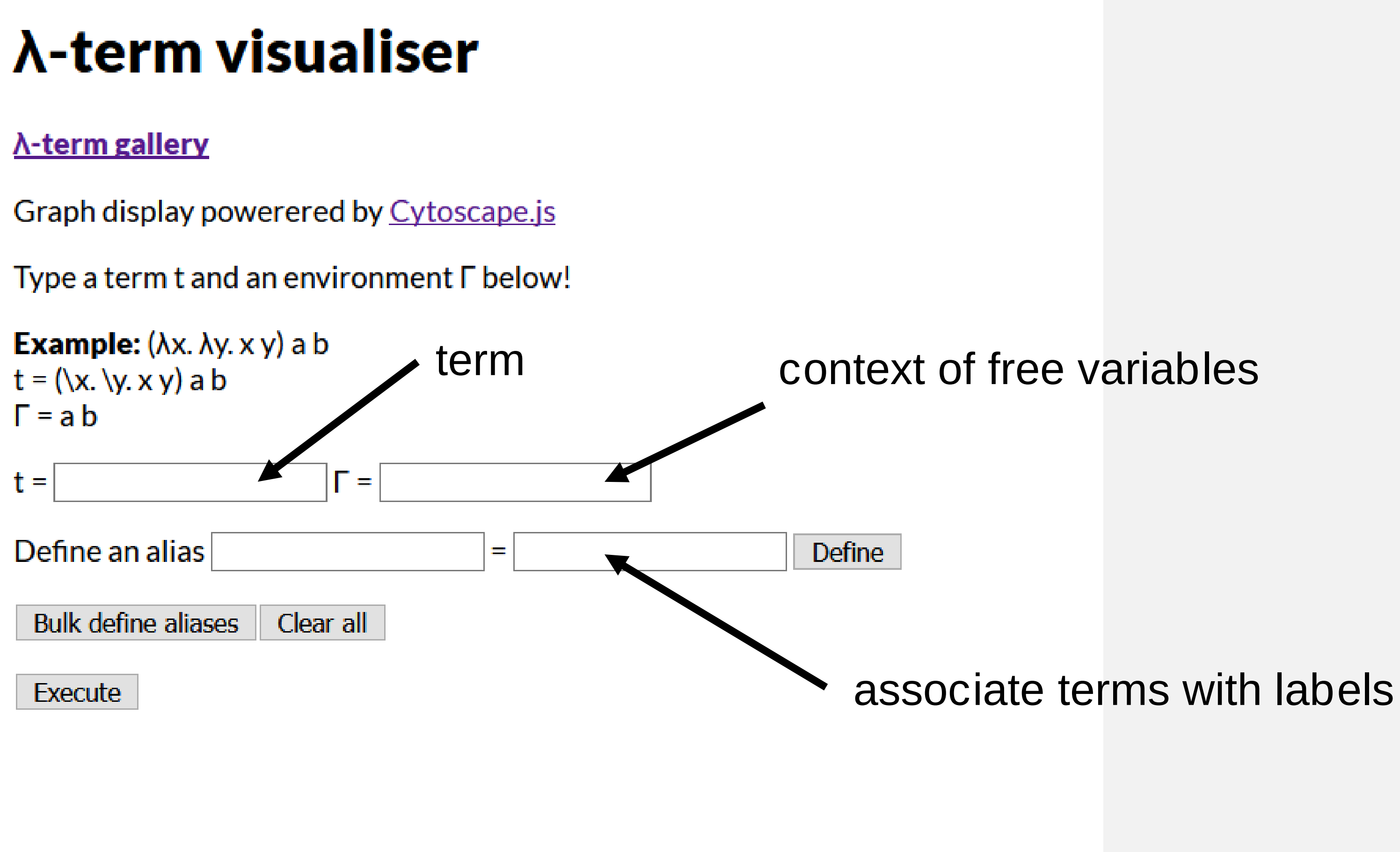
- It can be interesting to examine the different topological properties shared between the maps of terms
- We can perform **experimental mathematics** with these maps
- We want to be able to **test conjectures** about these maps
- But drawing them can be **time-consuming...**
- So why not get something to do it for us!

Outline

- Background
- Motivation
- **Demo**
- Future work

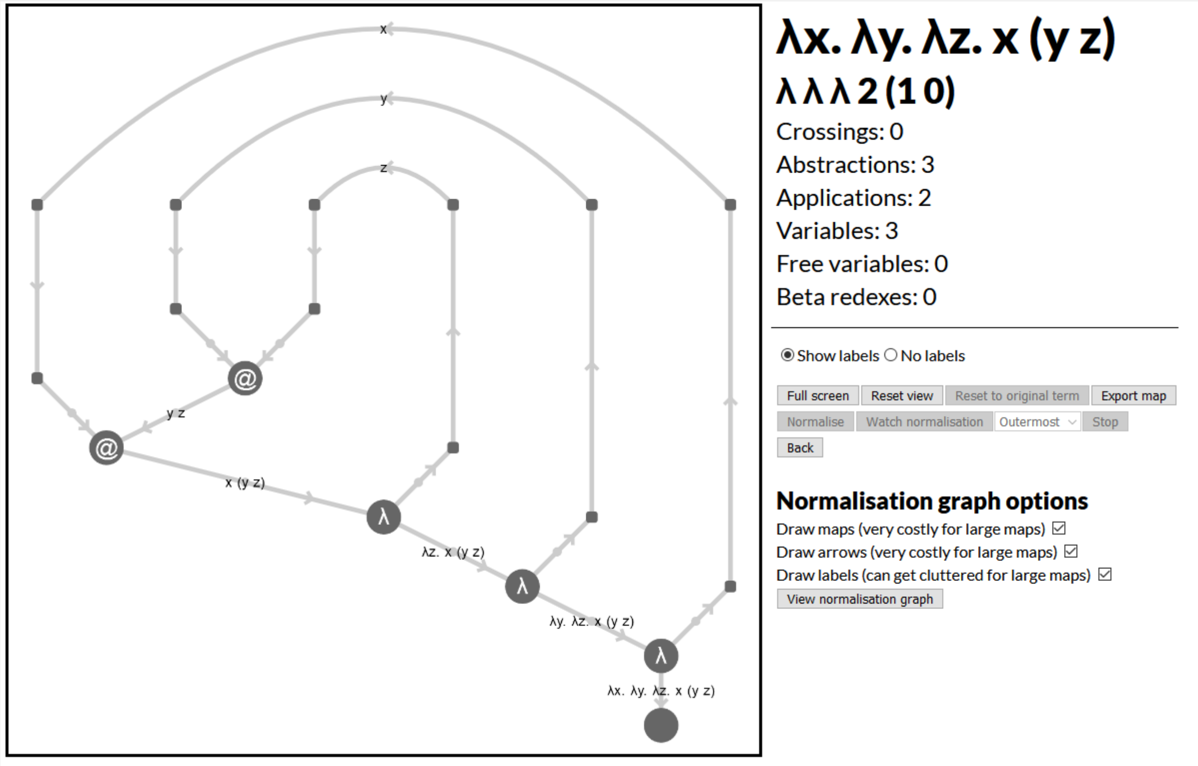
Demo

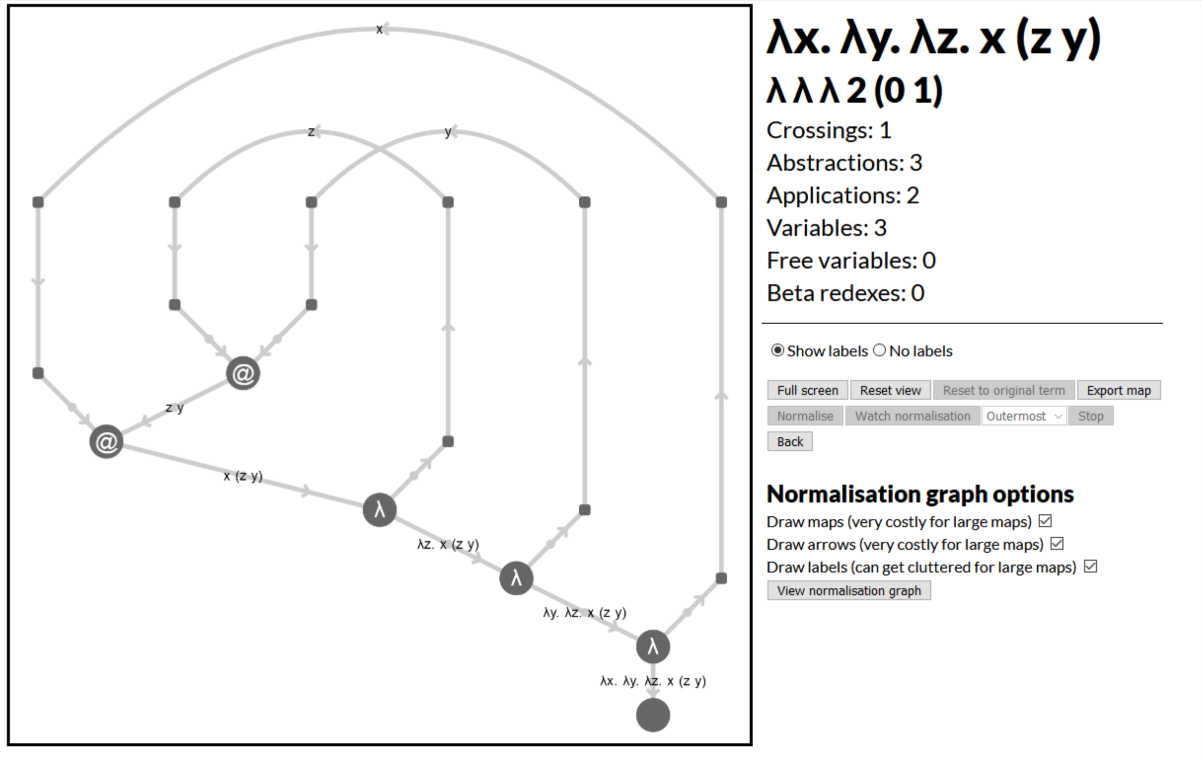
https://www.georgejkaye.com/pages/fyp/visualiser.html



Demo

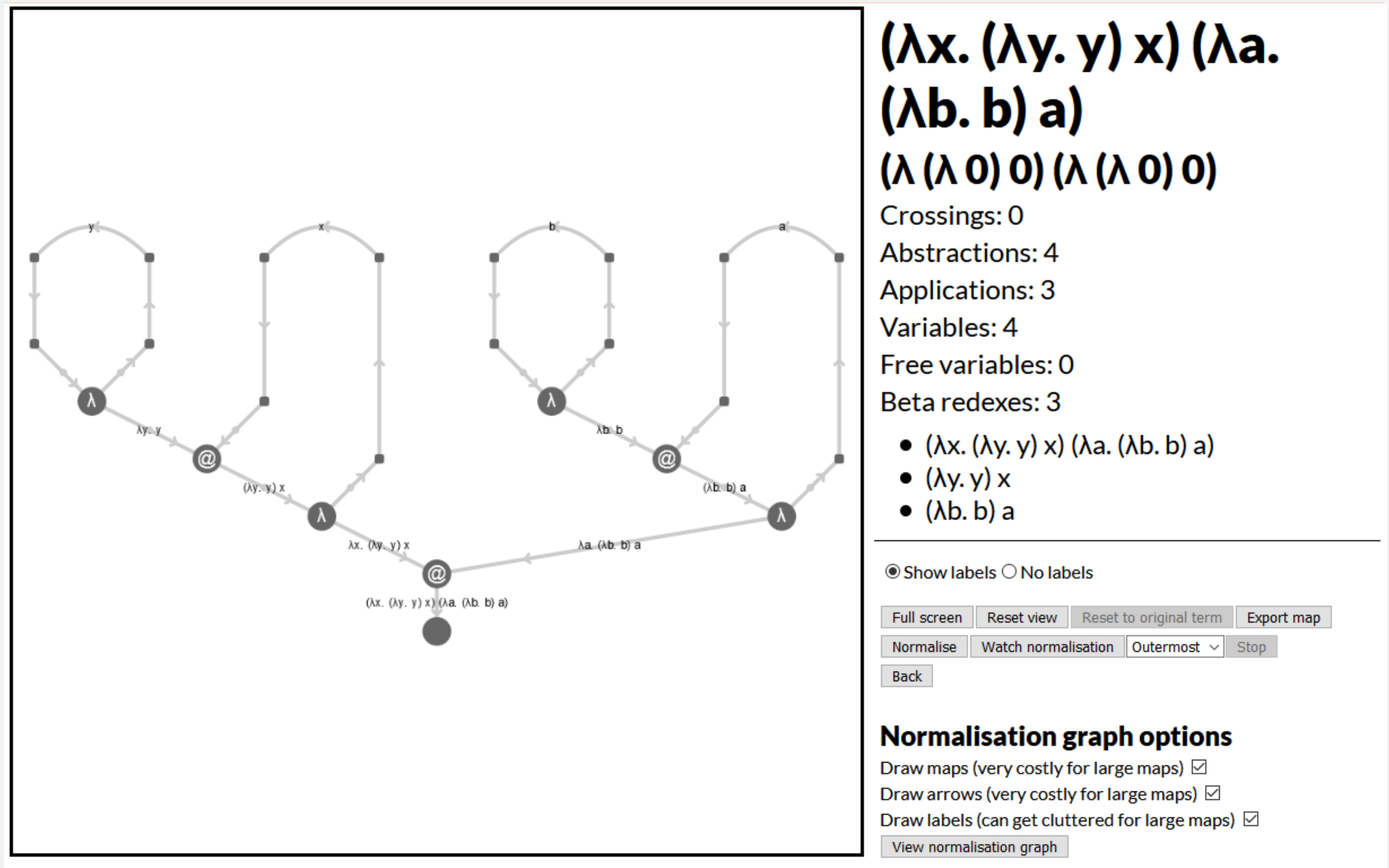
https://www.georgejkaye.com/pages/fyp/visualiser.html





Demo

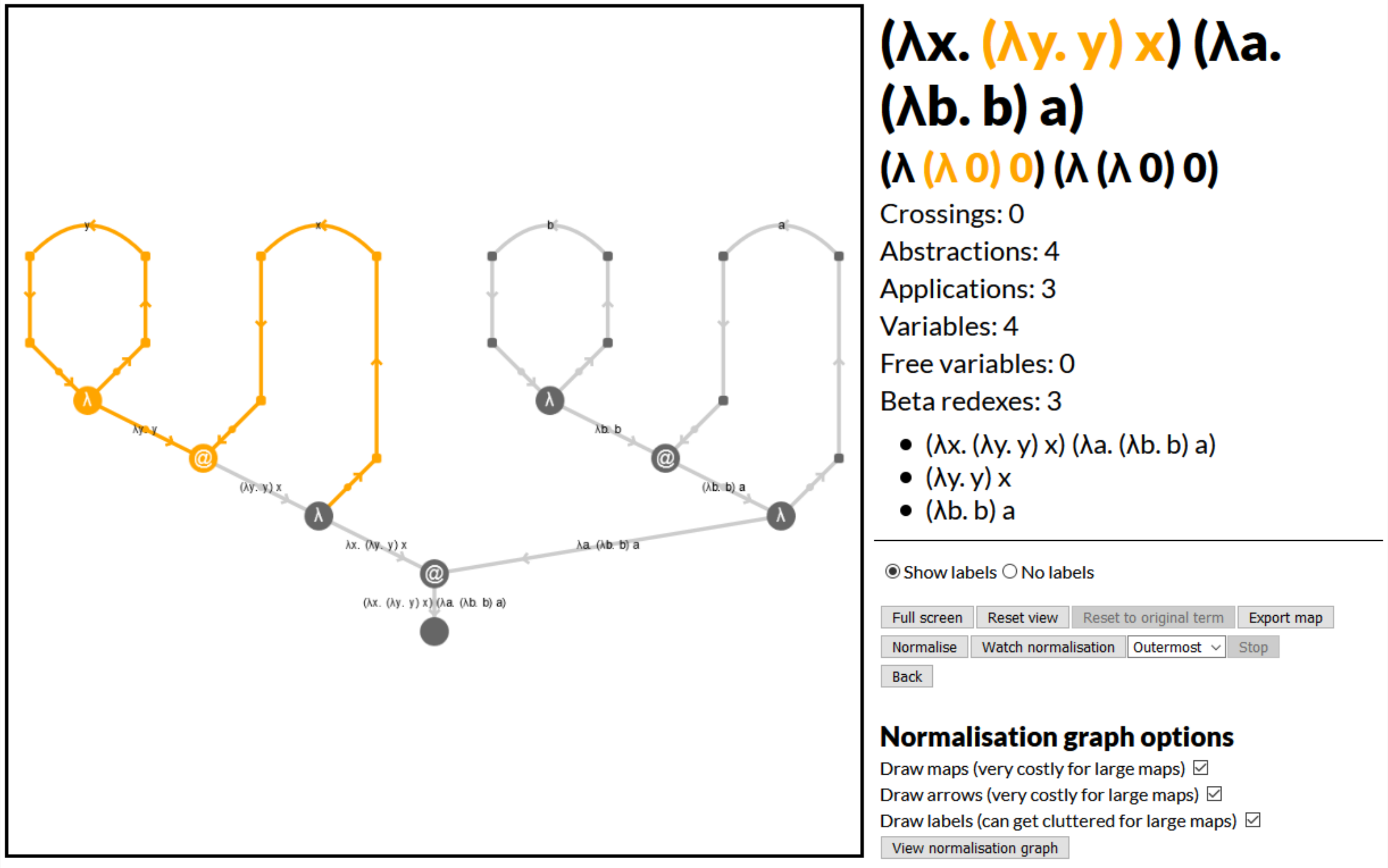
https://www.georgejkaye.com/pages/fyp/visualiser.html



The redexes in the term are listed alongside the map

Demo

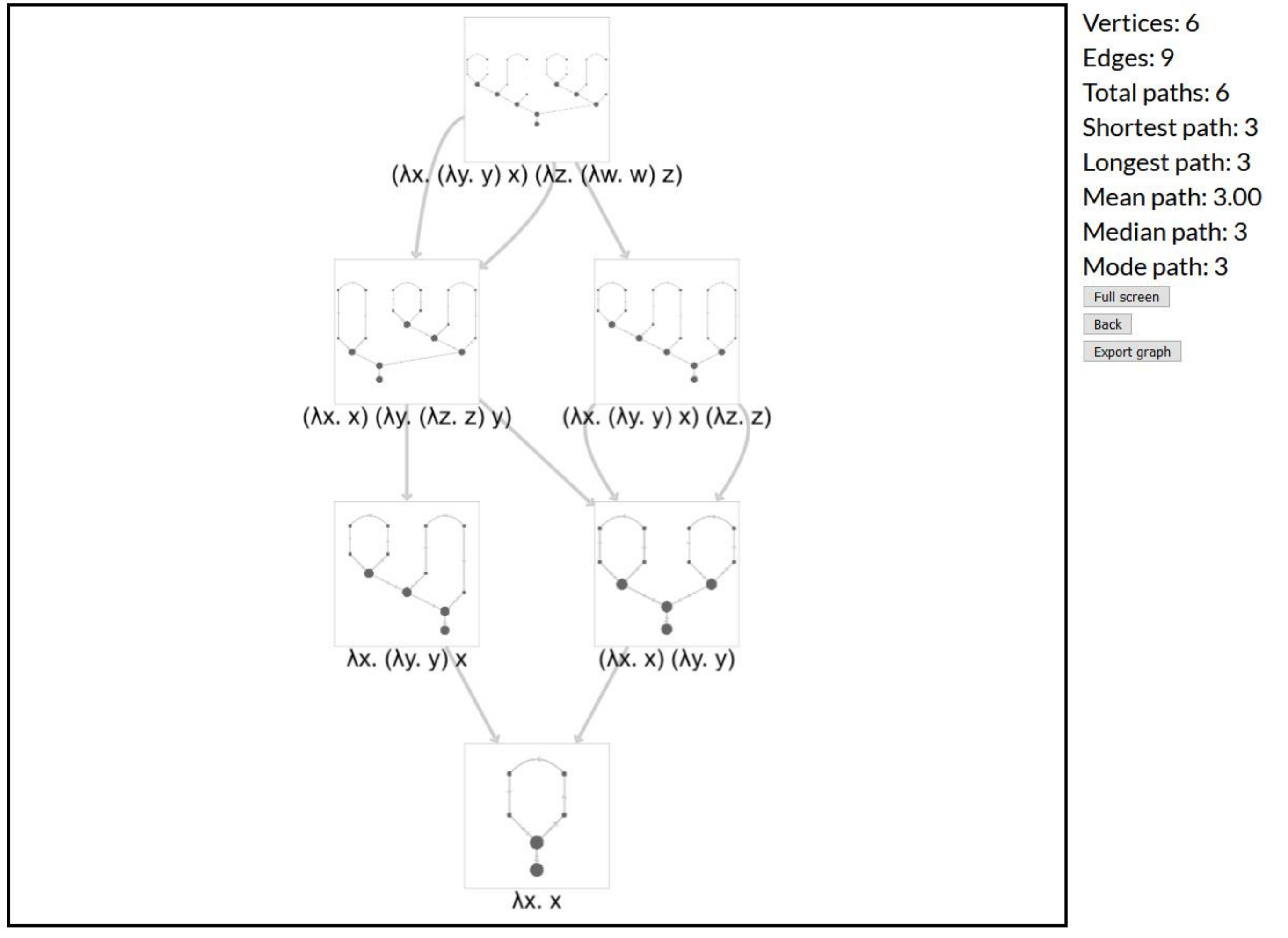
https://www.georgejkaye.com/pages/fyp/visualiser.html



Hovering over a redex in the list will highlight it in the term and the map

Demo

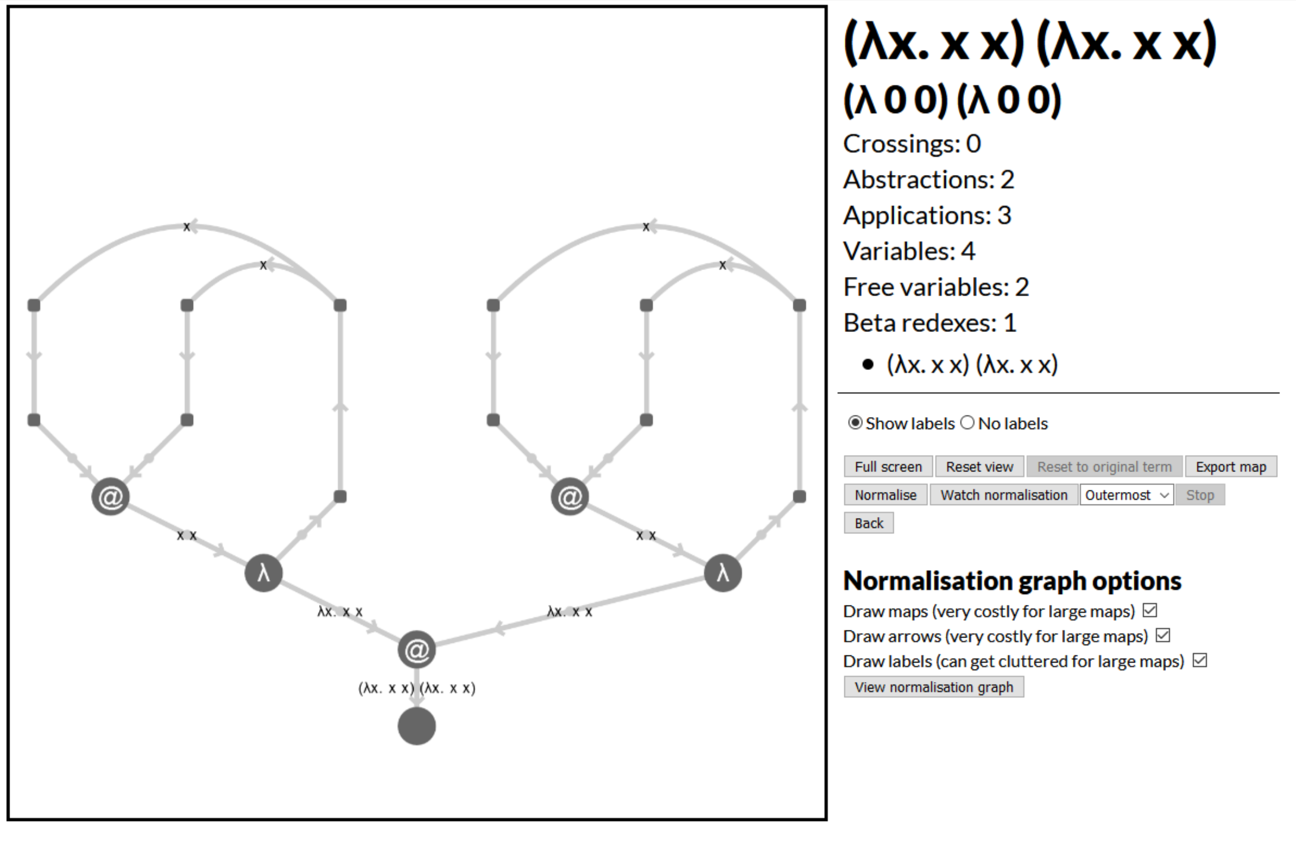
https://www.georgejkaye.com/pages/fyp/visualiser.html



We can generate normalisation graphs

Demo

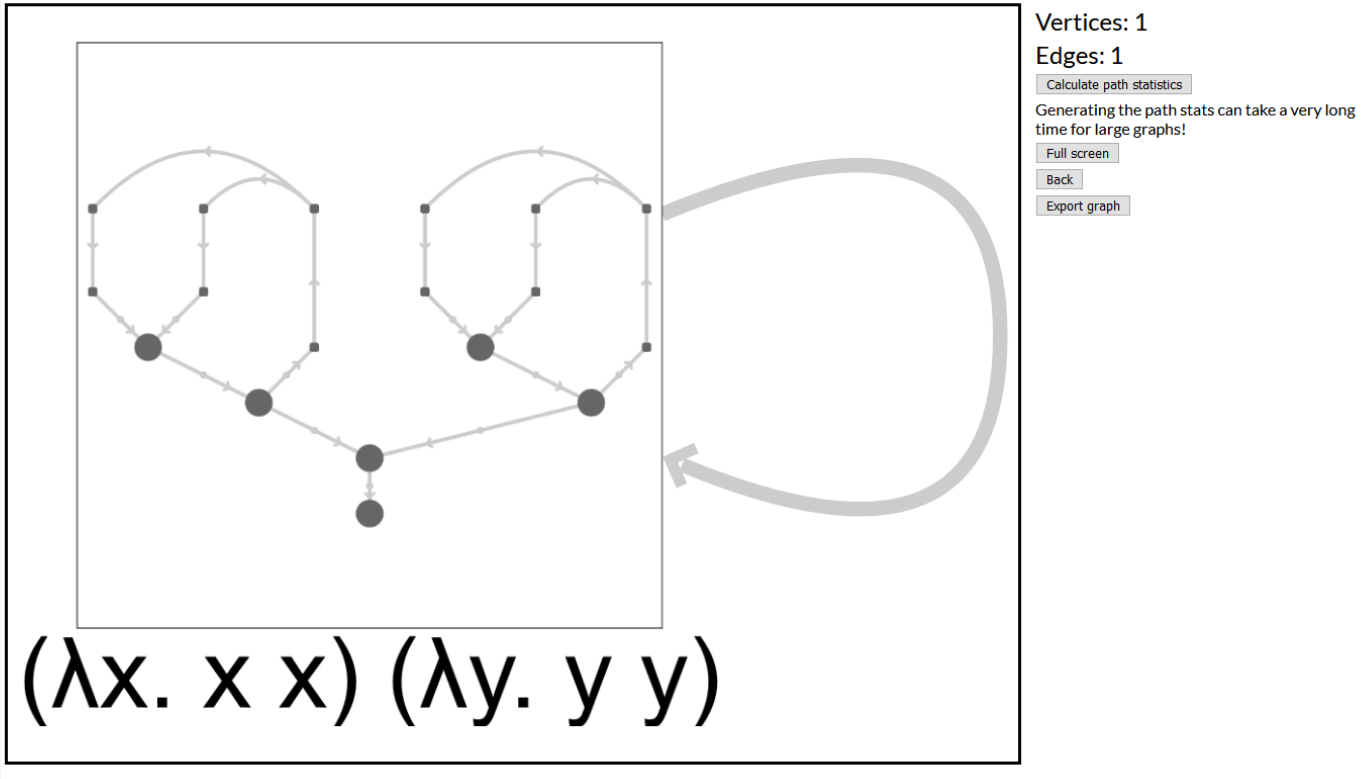
https://www.georgejkaye.com/pages/fyp/visualiser.html



We can visualise pure terms too!

Demo

https://www.georgejkaye.com/pages/fyp/visualiser.html



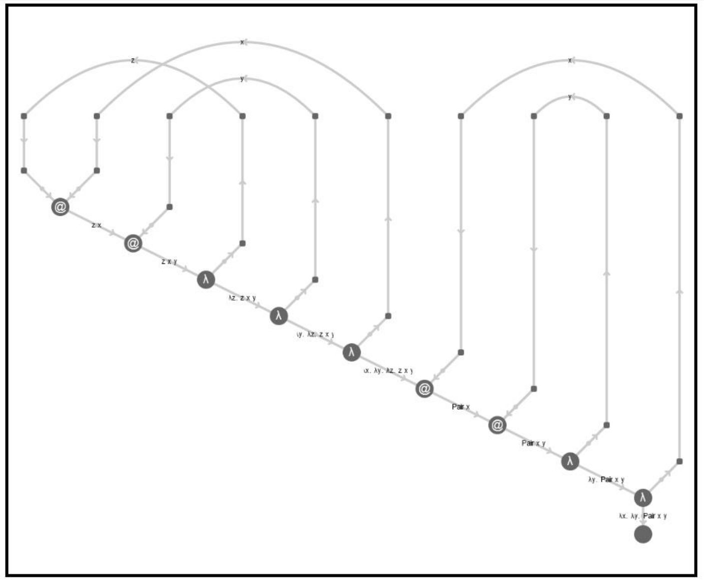
And their normalisation graphs if they're finite

(infinite graphs will give up after ~100 reductions)

Demo

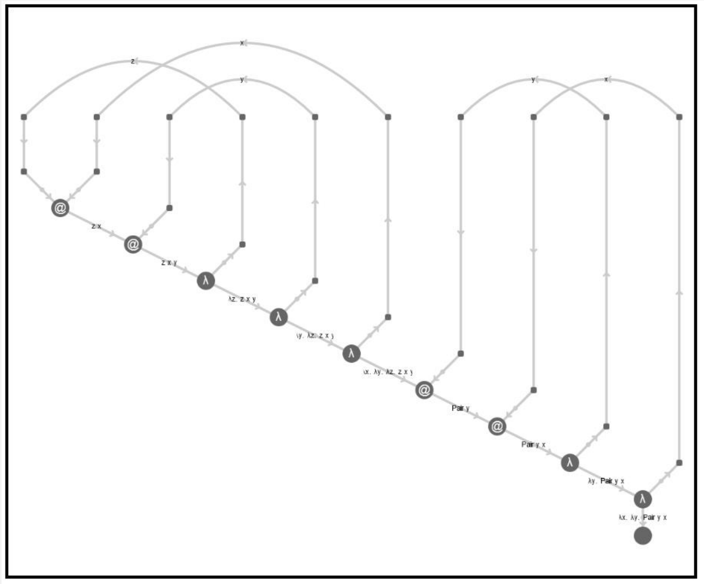
https://www.georgejkaye.com/pages/fyp/visualiser.html

Examples using Mairson’s Boolean circuit encodings



True

λ λ (λ λ λ 0 2 1) 1 0

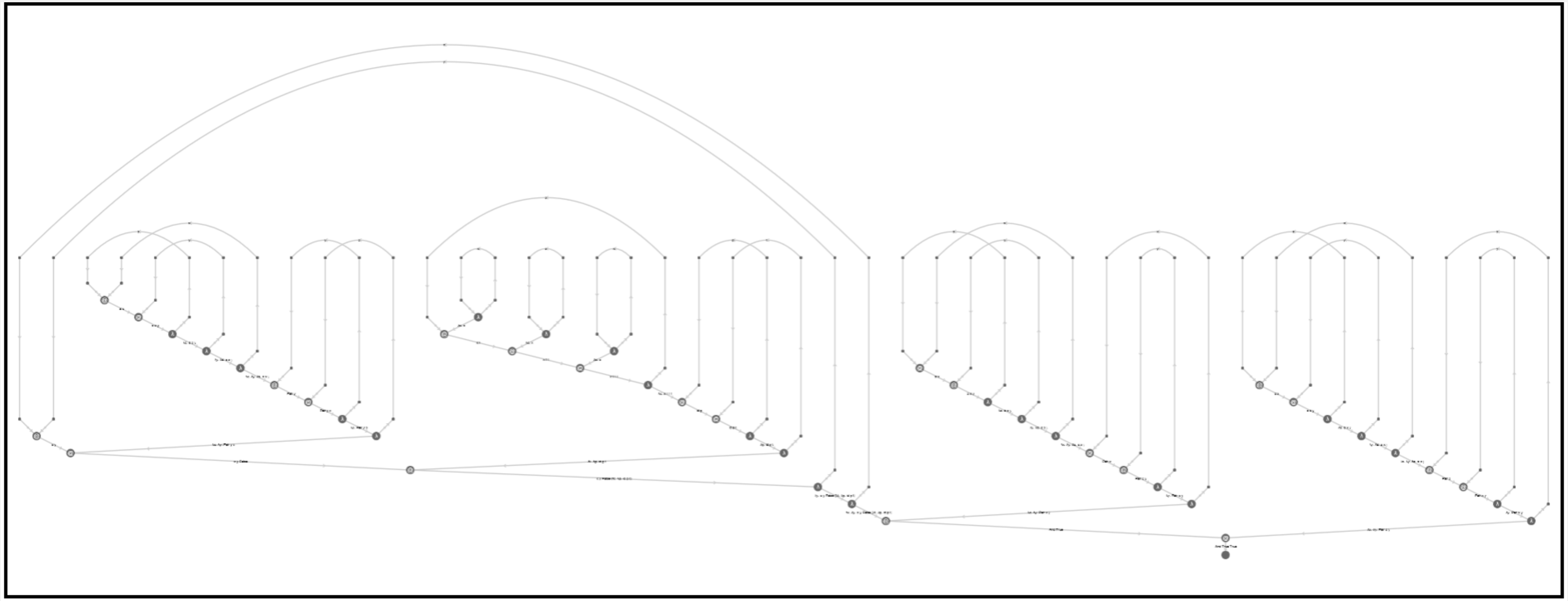


False

λ λ (λ λ λ 0 2 1) 0 1

Demo

https://www.georgejkaye.com/pages/fyp/visualiser.html

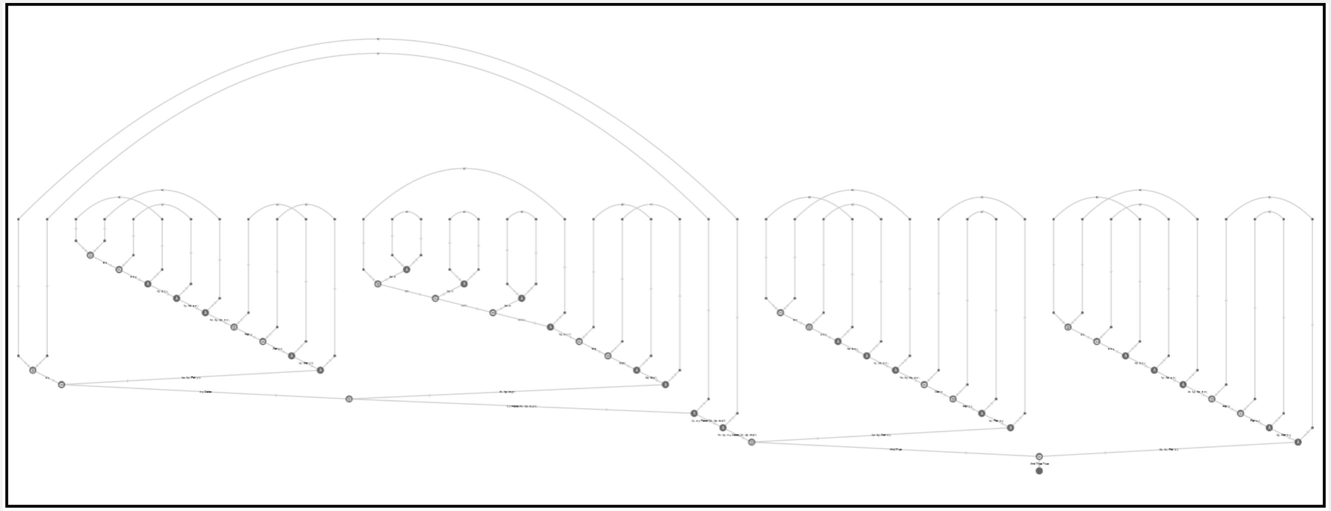


And True True

$(\lambda\,\lambda\,1\,0\,(\lambda\,\lambda\,(\lambda\,\lambda\,\lambda\,0\,2\,1)\,0\,1)\,(\lambda\,\lambda\,(\lambda\,0\,(\lambda\,0)\,(\lambda\,0)\,(\lambda\,0))\,0\,1))\,(\lambda\,\lambda\,(\lambda\,\lambda\,\lambda\,0\,2\,1)\,1\,0)\,(\lambda\,\lambda\,(\lambda\,\lambda\,\lambda\,0\,2\,1)\,1\,0)$

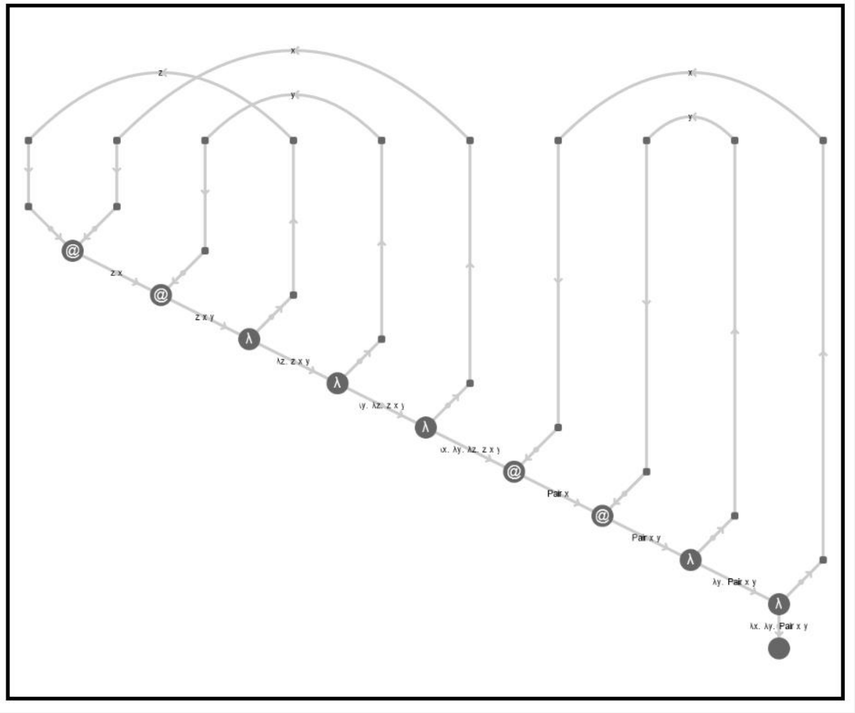
Demo

https://www.georgejkaye.com/pages/fyp/visualiser.html



And True True

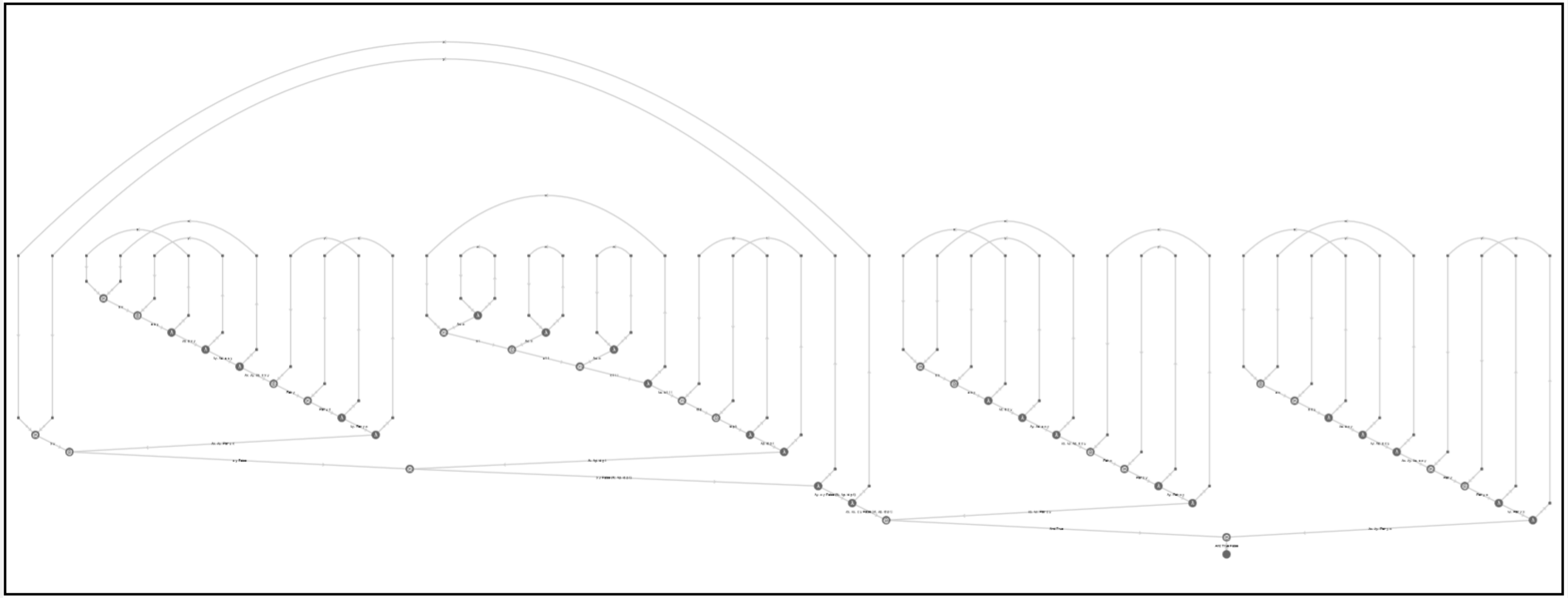
normalises to



True

Demo

https://www.georgejkaye.com/pages/fyp/visualiser.html

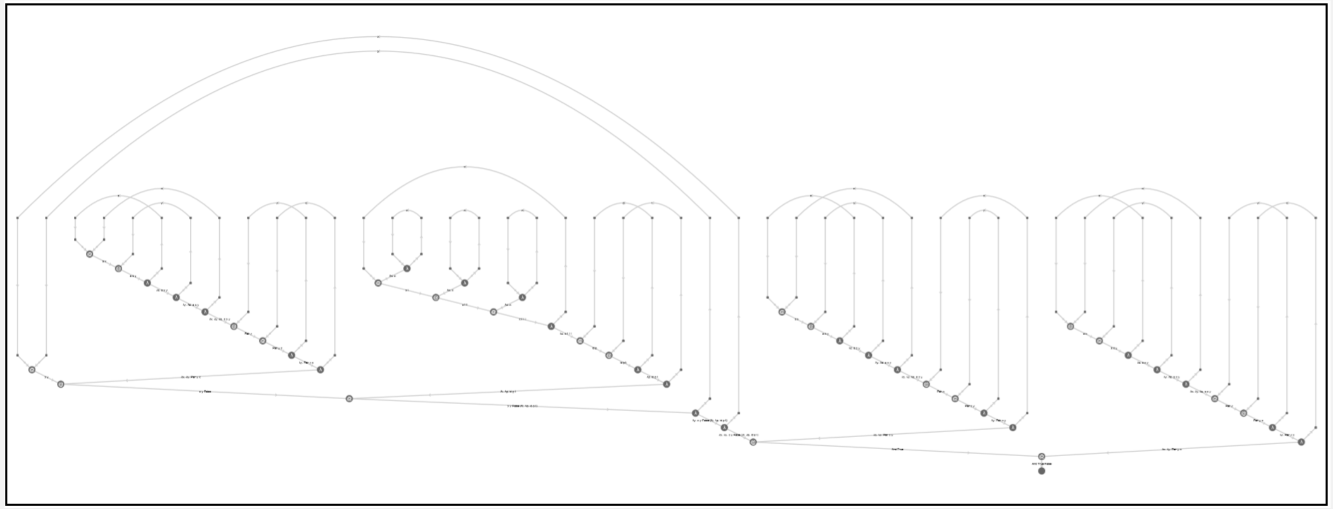


And True False

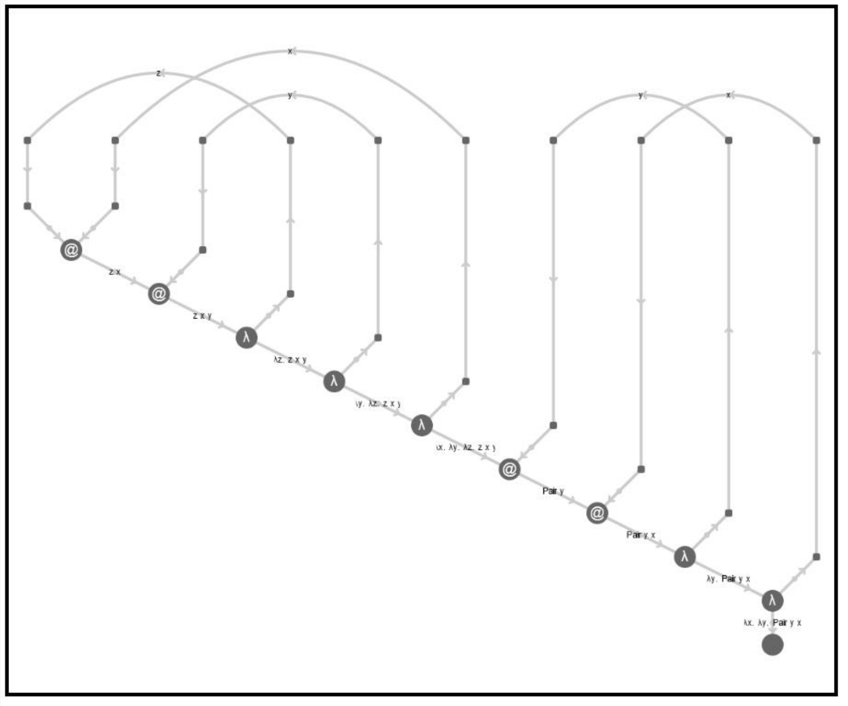
(λ λ 1 0 (λ λ (λ λ λ 0 2 1) 0 1) (λ λ (λ 0 (λ 0) (λ 0) (λ 0)) 0 1)) (λ λ (λ λ λ 0 2 1) 1 0) (λ λ (λ λ λ 0 2 1) 1 0)

Demo

https://www.georgejkaye.com/pages/fyp/visualiser.html



normalises to



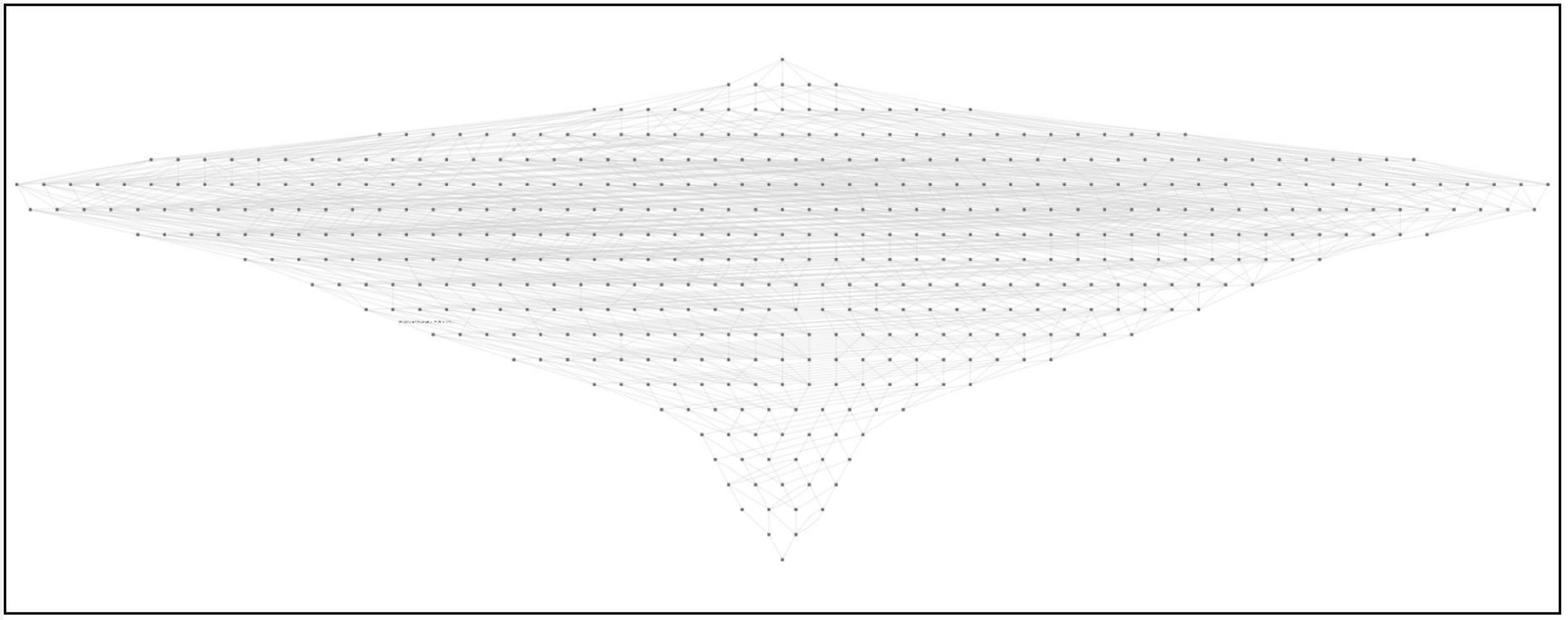
And True False

False

Demo

https://www.georgejkaye.com/pages/fyp/visualiser.html

Normalisation graph of **And True False**



Demo

https://www.georgejkaye.com/pages/fyp/gallery.html

λ-term gallery

[λ-term visualiser]

Graph display powerered by Cytoscape.js

The underlying algorithms behind the term generators can be found here (in Haskell!).

λ term generators

n [ ] k [ ] Pure Linear Planar

Number of subterms

Number of free variables

You might need to be patient for larger values of n and k while the maps are drawn. For n > 10 be prepared to wait a while, or until the universe collapses!

For larger terms, you may wish to disable map generation to speed up the process a bit.

Generation options

Draw maps (costly) ☑

Use de Bruijn notation ☐

Filtering options

Crossings [ ] Abstractions [ ] Applications [ ] Variables [ ] β-redexes [ ]

Demo

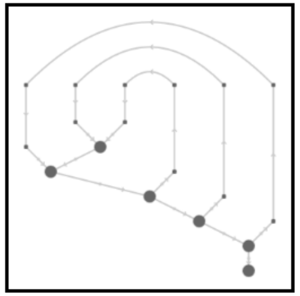
https://www.georgejkaye.com/pages/fyp/gallery.html

λ term generators

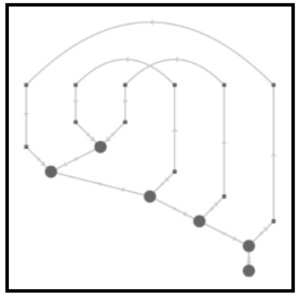
n 8 k 0 Pure Linear Planar

There are 60 linear terms for n = 8 and k = 0
60/60 terms match the filtering criteria: 100.00%

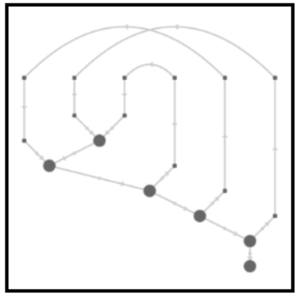
Click on a term to learn more about it. Clear all



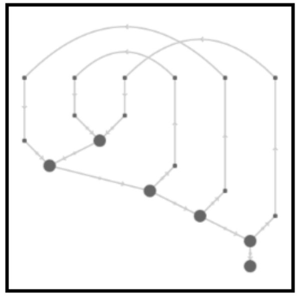
λx. λy. λz. x (y z)
0 crossings



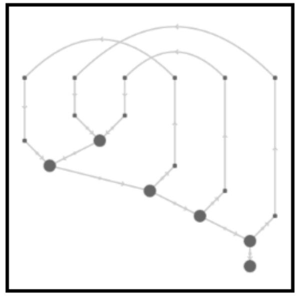
λx. λy. λz. x (z y)
1 crossings



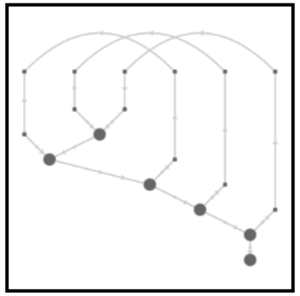
λx. λy. λz. y (x z)
1 crossings



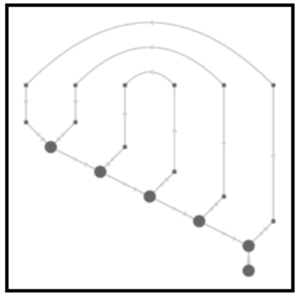
λx. λy. λz. y (z x)
2 crossings



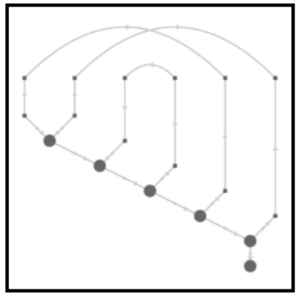
λx. λy. λz. z (x y)
2 crossings



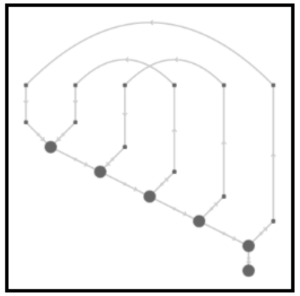
λx. λy. λz. z (y x)
3 crossings



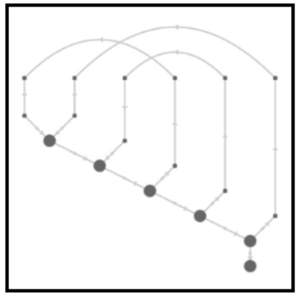
λx. λy. λz. x y z
0 crossings



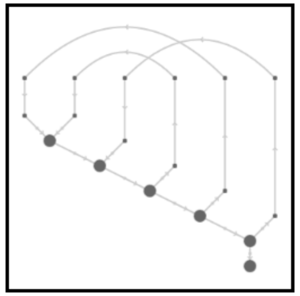
λx. λy. λz. y x z
1 crossings



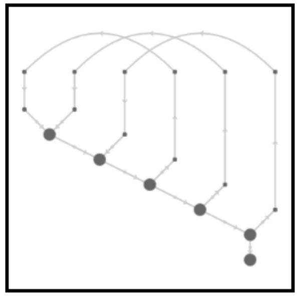
λx. λy. λz. x z y
1 crossings



λx. λy. λz. z x y
2 crossings



λx. λy. λz. y z x
2 crossings



λx. λy. λz. z y x
3 crossings

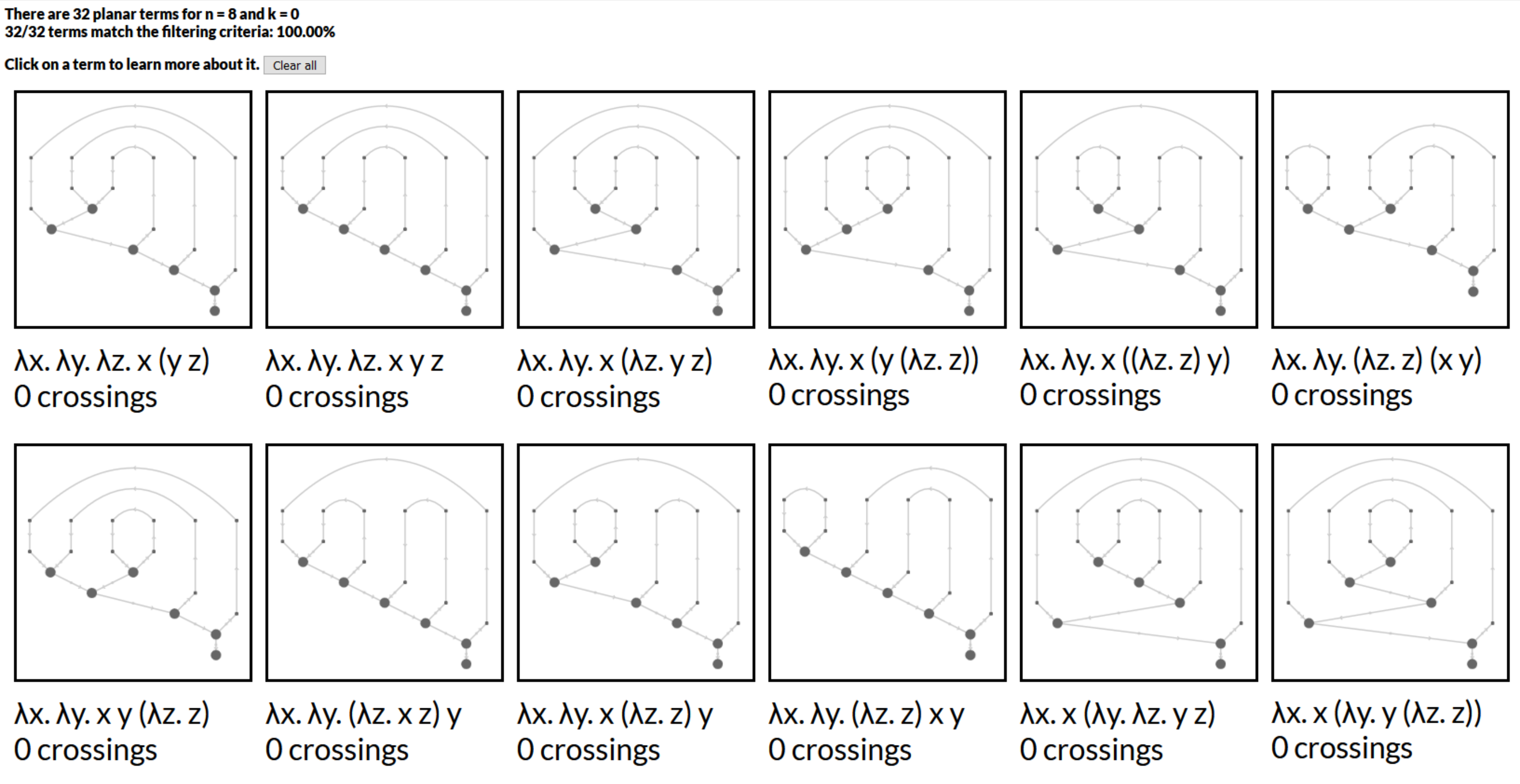
(this is only the first twelve)

Demo

https://www.georgejkaye.com/pages/fyp/gallery.html

λ term generators

n 8 k 0 Pure Linear Planar



(this is only the first twelve)

Demo

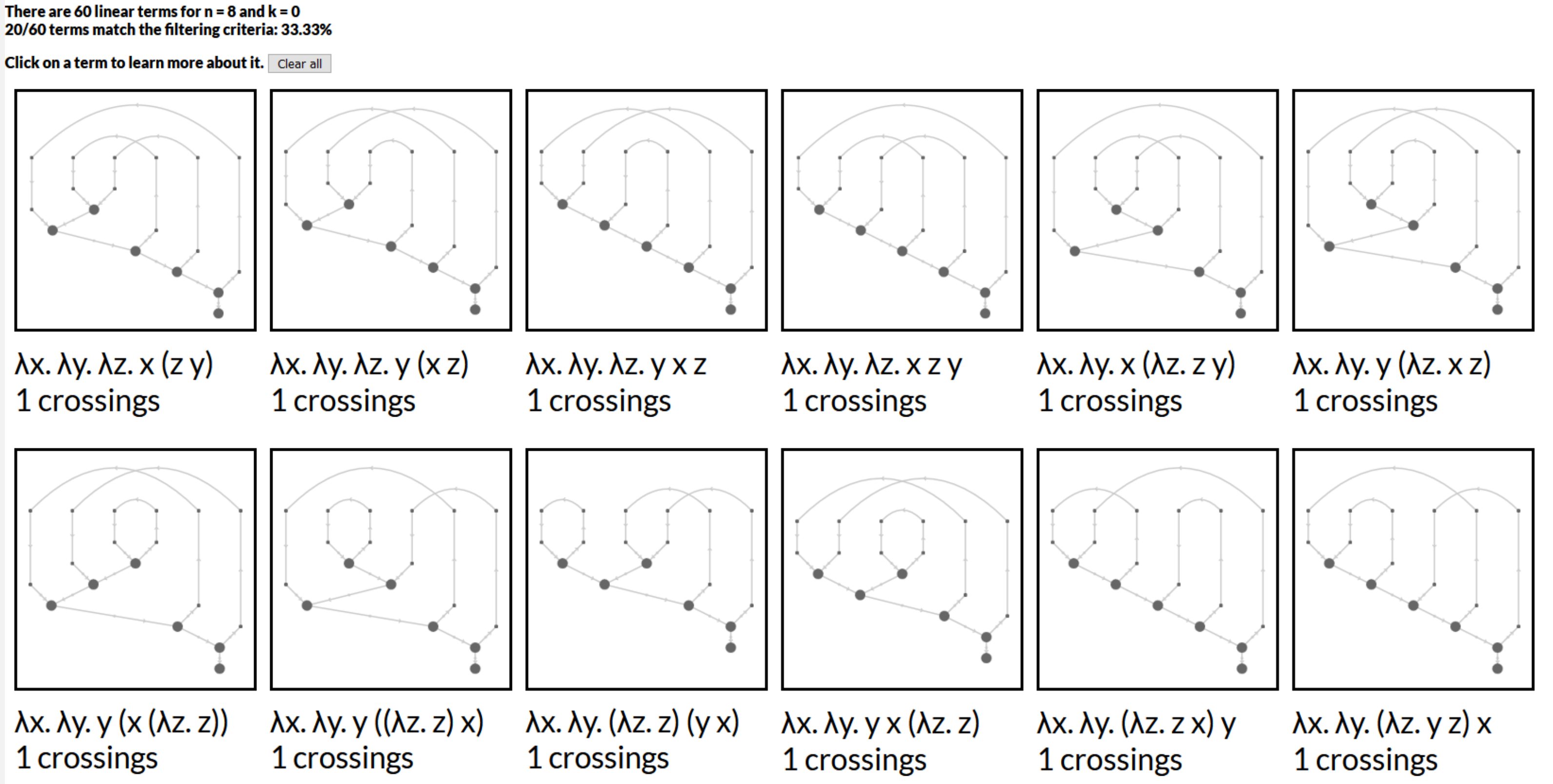
https://www.georgejkaye.com/pages/fyp/gallery.html

Filtering options

Crossings 1 Abstractions Applications Variables β-redexes

λ term generators

n 8 k 0 Pure Linear Planar



Demo

https://www.georgejkaye.com/pages/fyp/gallery.html

Filtering options

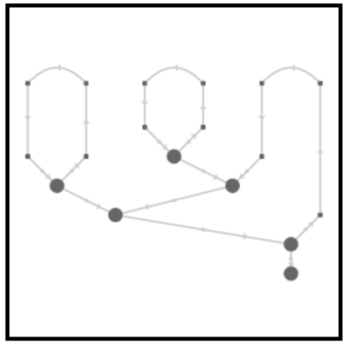
Crossings [ ] Abstractions [ ] Applications [ ] Variables [ ] β-redexes [2]

λ term generators

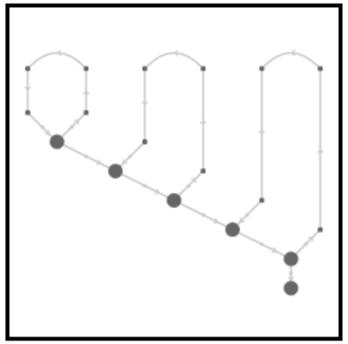
n [8] k [0] Pure Linear Planar

There are 60 linear terms for n = 8 and k = 0
5/60 terms match the filtering criteria: 8.33%

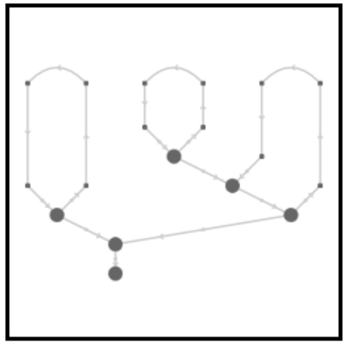
Click on a term to learn more about it. Clear all



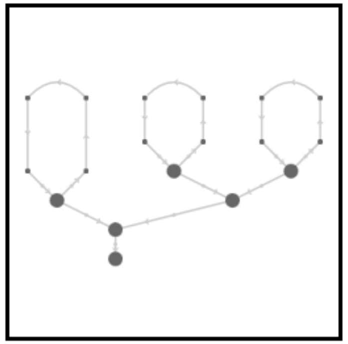
λx. (λy. y) ((λz. z) x)
0 crossings



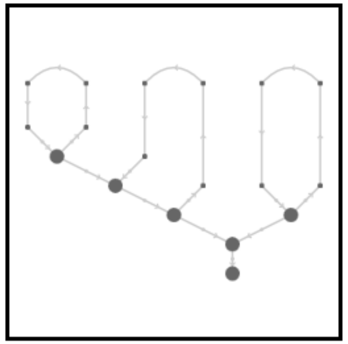
λx. (λy. (λz. z) y) x
0 crossings



(λx. x) (λy. (λz. z) y)
0 crossings



(λx. x) ((λy. y) (λz. z))
0 crossings

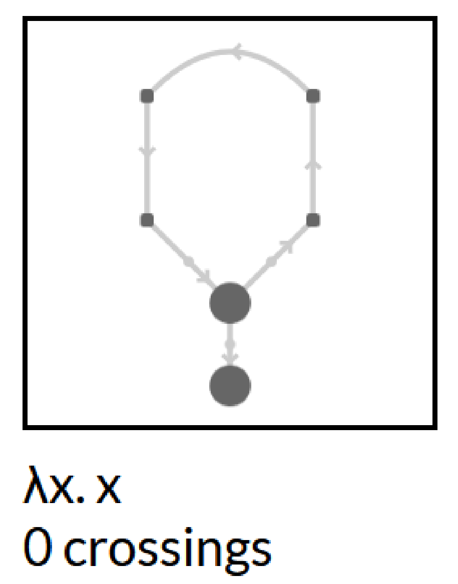


(λx. (λy. y) x) (λz. z)
0 crossings

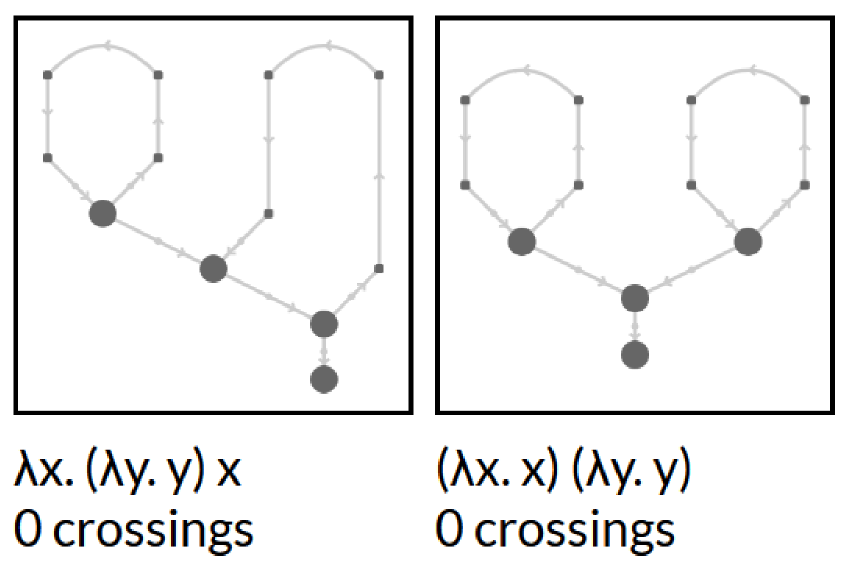
These are all planar… can we make a conjecture here?

Demo

https://www.georgejkaye.com/pages/fyp/gallery.html



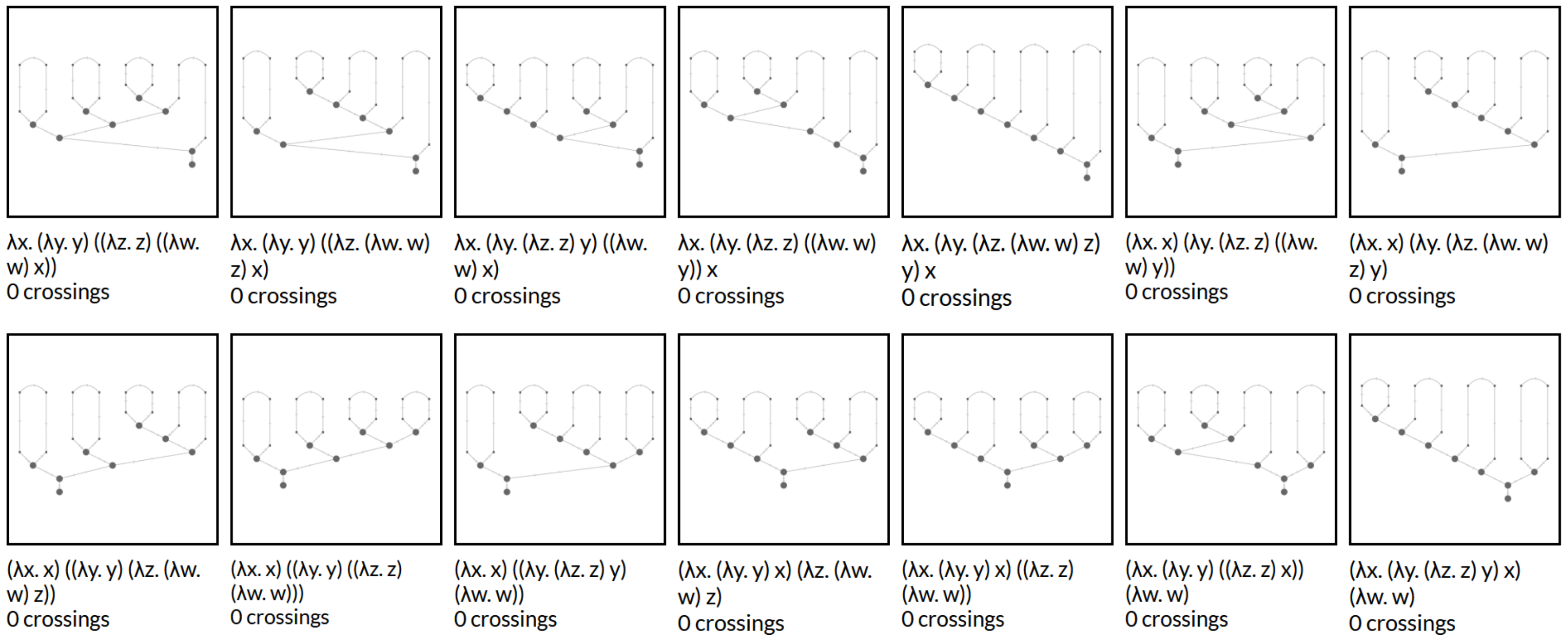
n = 2, k = 0, β = 0



n = 5, k = 0, β = 1

Demo

https://www.georgejkaye.com/pages/fyp/gallery.html



n = 11, k = 0, β = 3

So far so good…

Demo

https://www.georgejkaye.com/pages/fyp/gallery.html

Conjecture:

All closed linear lambda terms of size n with $\frac{n-2}{3}$ redexes are planar

Outline

- Background
- Motivation
- Demo
- **Future work**

Future work

- Efficient generation of subsets
- Alternative ways of visualising terms

georgejkaye.com/fyp