CS 4110

Programming Languages & Logics

Lecture 17
Fixed-Point Combinators

Review: Church Booleans

We can encode TRUE, FALSE, and IF, as:

$$\begin{aligned} \text{TRUE} &\triangleq \lambda x.\, \lambda y.\, x \\ \text{FALSE} &\triangleq \lambda x.\, \lambda y.\, y \\ \text{IF} &\triangleq \lambda b.\, \lambda t.\, \lambda f.\, b\ t\ f \end{aligned}$$

This way, IF behaves how it ought to:

$$\begin{aligned} \text{IF TRUE}\ v_t\ v_f &\rightarrow^* v_t \\ \text{IF FALSE}\ v_t\ v_f &\rightarrow^* v_f \end{aligned}$$

Review: Church Numerals

Church numerals encode a number n as a function that takes f and x, and applies f to x n times.

$$\begin{aligned}
\overline{0} &\triangleq \lambda f.\, \lambda x.\, x \\
\overline{1} &\triangleq \lambda f.\, \lambda x.\, f\, x \\
\overline{2} &\triangleq \lambda f.\, \lambda x.\, f\,(f\, x)
\end{aligned}$$

We can define other functions on integers:

$$\text{SUCC} \triangleq \lambda n.\, \lambda f.\, \lambda x.\, f\,(n\, f\, x)$$

Review: Church Numerals

Church numerals encode a number n as a function that takes f and x, and applies f to x n times.

$$\begin{array}{rcl} \overline{0} & \triangleq & \lambda f.\, \lambda x.\, x \\ \overline{1} & \triangleq & \lambda f.\, \lambda x.\, f\, x \\ \overline{2} & \triangleq & \lambda f.\, \lambda x.\, f\,(f\, x) \end{array}$$

We can define other functions on integers:

$$\begin{array}{rcl} \mathsf{SUCC} & \triangleq & \lambda n.\, \lambda f.\, \lambda x.\, f\,(n\, f\, x) \\ \mathsf{PLUS} & \triangleq & \lambda n_1.\, \lambda n_2.\, n_1\; \mathsf{SUCC}\; n_2 \end{array}$$

Review: Church Numerals

Church numerals encode a number n as a function that takes f and x, and applies f to x n times.

$$\begin{aligned}
\overline{0} &\triangleq \lambda f.\, \lambda x.\, x \\
\overline{1} &\triangleq \lambda f.\, \lambda x.\, f\, x \\
\overline{2} &\triangleq \lambda f.\, \lambda x.\, f\,(f\, x)
\end{aligned}$$

We can define other functions on integers:

$$\begin{aligned}
\mathsf{SUCC} &\triangleq \lambda n.\, \lambda f.\, \lambda x.\, f\,(n\, f\, x) \\
\mathsf{PLUS} &\triangleq \lambda n_1.\, \lambda n_2.\, n_1\; \mathsf{SUCC}\; n_2 \\
\mathsf{TIMES} &\triangleq \lambda n_1.\, \lambda n_2.\, n_1\; (\mathsf{PLUS}\ \mathsf{n_2})\; \overline{0}
\end{aligned}$$

Review: Church Numerals

Church numerals encode a number n as a function that takes f and x, and applies f to x n times.

$$\begin{aligned}\overline{0} &\triangleq \lambda f.\, \lambda x.\, x\\ \overline{1} &\triangleq \lambda f.\, \lambda x.\, f\, x\\ \overline{2} &\triangleq \lambda f.\, \lambda x.\, f\,(f\, x)\end{aligned}$$

We can define other functions on integers:

$$\begin{aligned}\mathsf{SUCC} &\triangleq \lambda n.\, \lambda f.\, \lambda x.\, f\,(n\, f\, x)\\ \mathsf{PLUS} &\triangleq \lambda n_1.\, \lambda n_2.\, n_1\; \mathsf{SUCC}\; n_2\\ \mathsf{TIMES} &\triangleq \lambda n_1.\, \lambda n_2.\, n_1\; (\mathsf{PLUS}\; \mathsf{n}_2)\; \overline{0}\\ \mathsf{ISZERO} &\triangleq \lambda n.\, n\; (\lambda z.\, \mathsf{FALSE})\; \mathsf{TRUE}\end{aligned}$$

Recursive Functions

How would we write recursive functions like factorial?

Recursive Functions

How would we write recursive functions like factorial?

We'd like to write it like this...

$$\mathsf{FACT} \triangleq \lambda \mathit{n}.\, \mathsf{IF}\ (\mathsf{ISZERO}\ \mathit{n})\ 1\ (\mathsf{TIMES}\ \mathit{n}\ (\mathsf{FACT}\ (\mathsf{PRED}\ \mathit{n})))$$

Recursive Functions

How would we write recursive functions like factorial?

We'd like to write it like this...

$$\mathsf{FACT} \triangleq \lambda n.\, \mathsf{IF}\ (\mathsf{ISZERO}\ n)\ 1\ (\mathsf{TIMES}\ n\ (\mathsf{FACT}\ (\mathsf{PRED}\ n)))$$

In slightly more readable notation this is...

$$\mathsf{FACT} \triangleq \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times \mathsf{FACT}\ (n - 1)$$

...but this is an equation, not a definition!

Recursion removal trick

We can perform a “trick” to define a function FACT that satisfies the recursive equation on the previous slide.

Recursion removal trick

We can perform a "trick" to define a function FACT that satisfies the recursive equation on the previous slide.

Define a new function FACT$'$ that takes a function f as an argument. Then, for "recursive" calls, it uses $f\, f$:

$$\text{FACT}' \triangleq \lambda f.\, \lambda n.\, \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times ((f\, f)\ (n - 1))$$

Recursion removal trick

We can perform a "trick" to define a function FACT that satisfies the recursive equation on the previous slide.

Define a new function FACT′ that takes a function *f* as an argument. Then, for "recursive" calls, it uses *f f*:

$$\text{FACT}' \triangleq \lambda f.\, \lambda n.\, \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times ((f\,f)\,(n-1))$$

Then define FACT as FACT′ applied to itself:

$$\text{FACT} \triangleq \text{FACT}'\ \text{FACT}'$$

Example

Let's try evaluating FACT on 3...

FACT 3

Example

Let's try evaluating FACT on 3...

$\text{FACT}\ 3 = (\text{FACT}'\ \text{FACT}')\ 3$

Example

Let's try evaluating FACT on 3...

$$\begin{aligned}\mathsf{FACT}\ 3 &= (\mathsf{FACT}'\ \mathsf{FACT}')\ 3 \\ &= ((\lambda f.\ \lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times ((f\, f)\ (n-1)))\ \mathsf{FACT}')\ 3\end{aligned}$$

Example

Let's try evaluating FACT on 3...

$$
\begin{aligned}
\mathsf{FACT}\;3 &= (\mathsf{FACT}'\;\mathsf{FACT}')\;3 \\
&= ((\lambda f.\,\lambda n.\,\textbf{if}\;n = 0\;\textbf{then}\;1\;\textbf{else}\;n \times ((f\,f)\;(n-1)))\;\mathsf{FACT}')\;3 \\
&\to (\lambda n.\,\textbf{if}\;n = 0\;\textbf{then}\;1\;\textbf{else}\;n \times ((\mathsf{FACT}'\;\mathsf{FACT}')\;(n-1)))\;3
\end{aligned}
$$

Example

Let’s try evaluating FACT on 3...

$$
\begin{aligned}
\mathsf{FACT}\;3 &= (\mathsf{FACT}'\;\mathsf{FACT}')\;3 \\
&= ((\lambda f.\,\lambda n.\,\textbf{if}\;n = 0\;\textbf{then}\;1\;\textbf{else}\;n \times ((f\,f)\;(n-1)))\;\mathsf{FACT}')\;3 \\
&\to (\lambda n.\,\textbf{if}\;n = 0\;\textbf{then}\;1\;\textbf{else}\;n \times ((\mathsf{FACT}'\;\mathsf{FACT}')\;(n-1)))\;3 \\
&\to \textbf{if}\;3 = 0\;\textbf{then}\;1\;\textbf{else}\;3 \times ((\mathsf{FACT}'\;\mathsf{FACT}')\;(3-1))
\end{aligned}
$$

Example

Let's try evaluating FACT on 3...

$$
\begin{aligned}
\mathsf{FACT}\ 3 &= (\mathsf{FACT}'\ \mathsf{FACT}')\ 3 \\
&= ((\lambda f.\ \lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times ((f\ f)\ (n-1)))\ \mathsf{FACT}')\ 3 \\
&\to (\lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times ((\mathsf{FACT}'\ \mathsf{FACT}')\ (n-1)))\ 3 \\
&\to \textbf{if}\ 3 = 0\ \textbf{then}\ 1\ \textbf{else}\ 3 \times ((\mathsf{FACT}'\ \mathsf{FACT}')\ (3-1)) \\
&\to 3 \times ((\mathsf{FACT}'\ \mathsf{FACT}')\ (3-1))
\end{aligned}
$$

Example

Let's try evaluating FACT on 3...

$$
\begin{aligned}
\mathsf{FACT}\;3 &= (\mathsf{FACT}'\;\mathsf{FACT}')\;3 \\
&= ((\lambda f.\,\lambda n.\,\textbf{if}\;n = 0\;\textbf{then}\;1\;\textbf{else}\;n \times ((f\,f)\;(n-1)))\;\mathsf{FACT}')\;3 \\
&\to (\lambda n.\,\textbf{if}\;n = 0\;\textbf{then}\;1\;\textbf{else}\;n \times ((\mathsf{FACT}'\;\mathsf{FACT}')\;(n-1)))\;3 \\
&\to \textbf{if}\;3 = 0\;\textbf{then}\;1\;\textbf{else}\;3 \times ((\mathsf{FACT}'\;\mathsf{FACT}')\;(3-1)) \\
&\to 3 \times ((\mathsf{FACT}'\;\mathsf{FACT}')\;(3-1)) \\
&= 3 \times (\mathsf{FACT}\;(3-1))
\end{aligned}
$$

Example

Let's try evaluating FACT on 3...

$$
\begin{aligned}
\mathsf{FACT}\;3 &= (\mathsf{FACT}'\;\mathsf{FACT}')\;3 \\
&= ((\lambda f.\,\lambda n.\,\textbf{if}\;n = 0\;\textbf{then}\;1\;\textbf{else}\;n \times ((f\,f)\;(n-1)))\;\mathsf{FACT}')\;3 \\
&\to (\lambda n.\,\textbf{if}\;n = 0\;\textbf{then}\;1\;\textbf{else}\;n \times ((\mathsf{FACT}'\;\mathsf{FACT}')\;(n-1)))\;3 \\
&\to \textbf{if}\;3 = 0\;\textbf{then}\;1\;\textbf{else}\;3 \times ((\mathsf{FACT}'\;\mathsf{FACT}')\;(3-1)) \\
&\to 3 \times ((\mathsf{FACT}'\;\mathsf{FACT}')\;(3-1)) \\
&= 3 \times (\mathsf{FACT}\;(3-1)) \\
&\to \ldots \\
&\to 3 \times 2 \times 1 \times 1
\end{aligned}
$$

Example

Let's try evaluating FACT on 3...

$$
\begin{aligned}
\mathsf{FACT}\ 3 &= (\mathsf{FACT}'\ \mathsf{FACT}')\ 3 \\
&= ((\lambda f.\, \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times ((f\, f)\ (n-1)))\ \mathsf{FACT}')\ 3 \\
&\to (\lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times ((\mathsf{FACT}'\ \mathsf{FACT}')\ (n-1)))\ 3 \\
&\to \textbf{if}\ 3 = 0\ \textbf{then}\ 1\ \textbf{else}\ 3 \times ((\mathsf{FACT}'\ \mathsf{FACT}')\ (3-1)) \\
&\to 3 \times ((\mathsf{FACT}'\ \mathsf{FACT}')\ (3-1)) \\
&= 3 \times (\mathsf{FACT}\ (3-1)) \\
&\to \ldots \\
&\to 3 \times 2 \times 1 \times 1 \\
&\to^{*} 6
\end{aligned}
$$

Fixed point combinators

Our "trick" requires following human-readable instructions.
Write a different function f' that takes itself as an argument and uses self-application for recursive calls, and then define f as $f'\ f'$.

Fixed point combinators

Our “trick” requires following human-readable instructions. Write a different function f' that takes itself as an argument and uses self-application for recursive calls, and then define f as $f'\ f'$.

There is another way: fixed points!

Fixed point combinators

Our "trick" requires following human-readable instructions. Write a different function f' that takes itself as an argument and uses self-application for recursive calls, and then define f as $f'\ f'$.

There is another way: fixed points!

Consider factorial again. It is a fixed point of the following:

$$G \triangleq \lambda f.\, \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (f\,(n-1))$$

Fixed point combinators

Our "trick" requires following human-readable instructions. Write a different function f' that takes itself as an argument and uses self-application for recursive calls, and then define f as $f'\ f'$.

There is another way: fixed points!

Consider factorial again. It is a fixed point of the following:

$$G \triangleq \lambda f.\, \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (f\,(n-1))$$

Recall that if g if a fixed point of G, then $G\,g = g$. To see that any fixed point g is a real factorial function, try evaluating it:

$$g\,5 = (G\,g)\,5$$

Fixed point combinators

Our "trick" requires following human-readable instructions. Write a different function f' that takes itself as an argument and uses self-application for recursive calls, and then define f as $f'\ f'$.

There is another way: fixed points!

Consider factorial again. It is a fixed point of the following:

$$G \triangleq \lambda f.\, \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (f\,(n-1))$$

Recall that if g if a fixed point of G, then $G\,g = g$. To see that any fixed point g is a real factorial function, try evaluating it:

$$\begin{aligned} g\,5 &= (G\,g)\,5 \\ &\to^* 5 \times (g\,4) \end{aligned}$$

Fixed point combinators

Our "trick" requires following human-readable instructions. Write a different function f' that takes itself as an argument and uses self-application for recursive calls, and then define f as $f'\ f'$.

There is another way: fixed points!

Consider factorial again. It is a fixed point of the following:

$$G \triangleq \lambda f.\, \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (f\,(n-1))$$

Recall that if g if a fixed point of G, then $G\,g = g$. To see that any fixed point g is a real factorial function, try evaluating it:

$$\begin{aligned} g\,5 &= (G\,g)\,5 \\ &\to^* 5 \times (g\,4) \\ &= 5 \times ((G\,g)\,4) \end{aligned}$$

Fixed point combinators

How can we generate the fixed point of G?

In denotational semantics, finding fixed points took a lot of math. In the λ-calculus, we just need a suitable combinator...

Y Combinator

The (infamous) Y combinator is defined as

$$\mathsf{Y} \triangleq \lambda f.\, (\lambda x.\, f\,(x\,x))\; (\lambda x.\, f\,(x\,x))$$

We say that Y is a *fixed point combinator* because Y f is a fixed point of f (for any lambda term f).

Y Combinator

The (infamous) Y combinator is defined as

$$\mathsf{Y} \triangleq \lambda f.\, (\lambda x.\, f\,(x\,x))\,(\lambda x.\, f\,(x\,x))$$

We say that Y is a *fixed point combinator* because Y f is a fixed point of f (for any lambda term f).

What happens when we evaluate Y G under CBV?

Z Combinator

To avoid this issue, we'll use a slight variant of the Y combinator, called Z, which is easier to use under CBV.

Z Combinator

To avoid this issue, we'll use a slight variant of the Y combinator, called Z, which is easier to use under CBV.

$$\mathsf{Z} \triangleq \lambda f.\,(\lambda x.\,f\,(\lambda y.\,x\,x\,y))\,(\lambda x.\,f\,(\lambda y.\,x\,x\,y))$$

Example

Let's see Z in action, on our function G.

FACT

Example

Let's see Z in action, on our function G.

$$\begin{aligned} & \mathrm{FACT} \\ = \quad & \mathrm{Z}\, G \end{aligned}$$

Example

Let's see Z in action, on our function G.

$$
\begin{array}{ll}
 & \text{FACT} \\
= & \text{Z}\, G \\
= & (\lambda f.\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y)))\, G
\end{array}
$$

Example

Let's see Z in action, on our function G.

$$
\begin{array}{cl}
 & \mathrm{FACT} \\
= & \mathrm{Z}\, G \\
= & (\lambda f.\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y)))\, G \\
\to & (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))
\end{array}
$$

Example

Let's see Z in action, on our function G.

$$
\begin{array}{rl}
 & \text{FACT} \\
= & \mathrm{Z}\, G \\
= & (\lambda f.\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y)))\, G \\
\rightarrow & (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y)) \\
\rightarrow & G\, (\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y)
\end{array}
$$

Example

Let's see Z in action, on our function G.

$$
\begin{array}{rl}
& \text{FACT} \\
= & \mathsf{Z}\, G \\
= & (\lambda f.\, (\lambda x.\, f\,(\lambda y.\, x\,x\,y))\,(\lambda x.\, f\,(\lambda y.\, x\,x\,y)))\; G \\
\to & (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\,(\lambda x.\, G\,(\lambda y.\, x\,x\,y)) \\
\to & G\,(\lambda y.\, (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\,(\lambda x.\, G\,(\lambda y.\, x\,x\,y))\; y) \\
= & (\lambda f.\, \lambda n.\, \textbf{if}\; n = 0\; \textbf{then}\; 1\; \textbf{else}\; n \times (f\,(n-1))) \\
& \qquad (\lambda y.\, (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\,(\lambda x.\, G\,(\lambda y.\, x\,x\,y))\; y)
\end{array}
$$

Example

Let's see Z in action, on our function G.

$$
\begin{array}{ll}
& \mathsf{FACT} \\
= & \mathsf{Z}\, G \\
= & (\lambda f.\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y)))\, G \\
\to & (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y)) \\
\to & G\, (\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y) \\
= & (\lambda f.\, \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (f\, (n-1))) \\
& \qquad (\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y) \\
\to & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1 \\
& \qquad \textbf{else}\ n \times ((\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y)\, (n-1))
\end{array}
$$

Example

Let's see Z in action, on our function G.

$$
\begin{array}{rl}
& \mathsf{FACT} \\
= & \mathsf{Z}\, G \\
= & (\lambda f.\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y)))\, G \\
\rightarrow & (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y)) \\
\rightarrow & G\, (\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y) \\
= & (\lambda f.\, \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (f\, (n-1))) \\
& \qquad (\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y) \\
\rightarrow & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1 \\
& \quad \textbf{else}\ n \times ((\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y)\, (n-1)) \\
=_\beta & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (\lambda y.\, (\mathsf{Z}\, G)\, y)\, (n-1)
\end{array}
$$

Example

Let's see Z in action, on our function G.

$$
\begin{array}{ll}
 & \mathsf{FACT} \\
= & \mathsf{Z}\, G \\
= & (\lambda f.\, (\lambda x.\, f\,(\lambda y.\, x\,x\,y))\, (\lambda x.\, f\,(\lambda y.\, x\,x\,y)))\, G \\
\to & (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\, (\lambda x.\, G\,(\lambda y.\, x\,x\,y)) \\
\to & G\,(\lambda y.\, (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\, (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\, y) \\
= & (\lambda f.\, \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (f\,(n-1))) \\
 & \qquad (\lambda y.\, (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\, (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\, y) \\
\to & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1 \\
 & \qquad \textbf{else}\ n \times ((\lambda y.\, (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\, (\lambda x.\, G\,(\lambda y.\, x\,x\,y))\, y)\, (n-1)) \\
=_\beta & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (\lambda y.\, (\mathsf{Z}\, G)\, y)\, (n-1) \\
=_\beta & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times ((\mathsf{Z}\, G)\, (n-1))
\end{array}
$$

Example

Let's see Z in action, on our function G.

$$
\begin{array}{ll}
 & \mathsf{FACT} \\
= & \mathsf{Z}\, G \\
= & (\lambda f.\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, f\, (\lambda y.\, x\, x\, y)))\, G \\
\rightarrow & (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y)) \\
\rightarrow & G\, (\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y) \\
= & (\lambda f.\, \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (f\, (n-1))) \\
 & \qquad (\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y) \\
\rightarrow & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1 \\
 & \quad \textbf{else}\ n \times ((\lambda y.\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, (\lambda x.\, G\, (\lambda y.\, x\, x\, y))\, y)\, (n-1)) \\
=_\beta & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (\lambda y.\, (\mathsf{Z}\, G)\, y)\, (n-1) \\
=_\beta & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times ((\mathsf{Z}\, G)\, (n-1)) \\
= & \lambda n.\, \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n \times (\mathsf{FACT}\, (n-1))
\end{array}
$$

Other fixed point combinators

There are many (indeed infinitely many) fixed-point combinators. Here's a cute one:

$$\mathsf{Y}_k \triangleq (\mathsf{L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L\,L})$$

where

$$\begin{aligned}\mathsf{L} \triangleq\ & \lambda abcdefghijklmnopqstuvwxyzr. \\ & (r\,(t\,h\,i\,s\,i\,s\,a\,f\,i\,x\,e\,d\,p\,o\,i\,n\,t\,c\,o\,m\,b\,i\,n\,a\,t\,o\,r))\end{aligned}$$

Turing’s Fixed Point Combinator

To gain some more intuition for fixed point combinators, let’s derive a combinator Θ originally discovered by Turing.

Turing's Fixed Point Combinator

To gain some more intuition for fixed point combinators, let's derive a combinator Θ originally discovered by Turing.

We know that $\Theta\, f$ is a fixed point of f, so we have

$$\Theta\, f = f\,(\Theta\, f).$$

Turing's Fixed Point Combinator

To gain some more intuition for fixed point combinators, let's derive a combinator Θ originally discovered by Turing.

We know that $\Theta\, f$ is a fixed point of f, so we have

$$\Theta\, f = f\,(\Theta\, f).$$

We can write the following recursive equation:

$$\Theta = \lambda f.\, f\,(\Theta\, f)$$

Turing's Fixed Point Combinator

To gain some more intuition for fixed point combinators, let's derive a combinator Θ originally discovered by Turing.

We know that $\Theta\, f$ is a fixed point of f, so we have

$$\Theta\, f = f\,(\Theta\, f).$$

We can write the following recursive equation:

$$\Theta = \lambda f.\, f\,(\Theta\, f)$$

Now use the recursion removal trick:

$$\begin{aligned} \Theta' &\triangleq \lambda t.\, \lambda f.\, f\,(t\,t\,f) \\ \Theta &\triangleq \Theta'\,\Theta' \end{aligned}$$

θ Example

$\mathsf{FACT} = \Theta\, G$

θ Example

$$
\begin{aligned}
\mathsf{FACT} &= \Theta\, G \\
&= ((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\,(\lambda t.\, \lambda f.\, f\,(t\,t\,f)))\, G
\end{aligned}
$$

θ Example

$$
\begin{aligned}
\mathsf{FACT} &= \Theta\, G \\
&= ((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\,(\lambda t.\, \lambda f.\, f\,(t\,t\,f)))\, G \\
&\to (\lambda f.\, f\,((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\,(\lambda t.\, \lambda f.\, f\,(t\,t\,f))\, f))\, G
\end{aligned}
$$

θ Example

$$\begin{aligned}
\mathsf{FACT} &= \Theta\, G \\
&= ((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\;(\lambda t.\, \lambda f.\, f\,(t\,t\,f)))\; G \\
&\to (\lambda f.\, f\,((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\;(\lambda t.\, \lambda f.\, f\,(t\,t\,f))\; f))\; G \\
&\to G\,((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\;(\lambda t.\, \lambda f.\, f\,(t\,t\,f))\; G)
\end{aligned}$$

θ Example

$$
\begin{aligned}
\mathsf{FACT} &= \Theta\, G \\
&= ((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\; (\lambda t.\, \lambda f.\, f\,(t\,t\,f)))\; G \\
&\rightarrow (\lambda f.\, f\,((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\; (\lambda t.\, \lambda f.\, f\,(t\,t\,f))\; f))\; G \\
&\rightarrow G\,((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\; (\lambda t.\, \lambda f.\, f\,(t\,t\,f))\; G) \\
&= G\,(\Theta\, G)
\end{aligned}
$$

θ Example

$$
\begin{aligned}
\mathsf{FACT} &= \Theta\, G \\
&= ((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\;(\lambda t.\, \lambda f.\, f\,(t\,t\,f)))\; G \\
&\to (\lambda f.\, f\,((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\;(\lambda t.\, \lambda f.\, f\,(t\,t\,f))\; f))\; G \\
&\to G\,((\lambda t.\, \lambda f.\, f\,(t\,t\,f))\;(\lambda t.\, \lambda f.\, f\,(t\,t\,f))\; G) \\
&= G\,(\Theta\, G) \\
&= (\lambda f.\, \lambda n.\, \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times (f\,(n-1)))\;(\Theta\, G) \\
&\to \lambda n.\, \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times ((\Theta\, G)\;(n-1)) \\
&= \lambda n.\, \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times (\mathsf{FACT}\;(n-1))
\end{aligned}
$$