Lecture 16
Fixed-Point Combinators
Termination in the $\lambda$-calculus

We have encoded lots of useful programming functionality that produces values.

Does every closed $\lambda$-term eventually terminate under CBN evaluation?

$$\forall \text{ closed term } e. \exists e'. e \rightarrow^* e' \land e' \not\rightarrow$$ ?
Termination in the $\lambda$-calculus

We have encoded lots of useful programming functionality that produces values.

Does every closed $\lambda$-term eventually terminate under CBN evaluation?

$$\forall \text{ closed term } e. \exists e'. e \rightarrow^* e' \land e' \not\rightarrow ?$$

No!

$$\Omega \triangleq (\lambda x.x\,x)\,(\lambda x.x\,x)$$
Termination in the $\lambda$-calculus

We have encoded lots of useful programming functionality that produces values.

Does every closed $\lambda$-term eventually terminate under CBN evaluation?

$$\forall \text{ closed term } e. \exists e'. e \rightarrow^* e' \land e' \not\rightarrow \ ?$$

No!

$$\Omega \triangleq (\lambda x. x x) (\lambda x. x x)$$

$$\rightarrow (x x) \{(\lambda x. x x)/x\}$$

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

$$= \Omega$$
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...

\[
\text{FACT} \triangleq \lambda n. \text{IF} \ (\text{ISZERO} \ n) \ 1 \ (\text{TIMES} \ n \ (\text{FACT} \ (\text{PRED} \ n)))
\]
Recursive Functions

How would we write recursive functions, like factorial?

We’d like to write it like this...

\[
FACT \triangleq \lambda n. \text{IF} (\text{ISZERO } n) \ 1 \ \text{(TIMES } n \ (FACT \ (\text{PRED } n)))
\]

In slightly more readable notation this is...

\[
FACT \triangleq \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times \text{FACT } (n - 1)
\]

...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.

\[
FACT' \equiv \lambda f . \lambda n. \begin{cases} 
1, & \text{if } n = 0 \\
 n \times (f (n - 1)), & \text{otherwise}
\end{cases}
\]

Then define FACT as FACT' applied to itself:

\[
FACT \equiv FACT'
\]
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 $ff$:

$$\text{FACT’} \triangleq \lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times ((ff)(n - 1))$$
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 $f f$:

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

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...

FACT 3 = (FACT’ FACT’) 3

= ((\lambda f. \lambda n. \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times ((f f) (n - 1))) \text{ FACT’}) 3
Let’s try evaluating FACT on 3...

\[ \text{FACT } 3 = (\text{FACT’ } \text{FACT’}) \ 3 \]

\[ = \left( \left( \lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (ff) (n - 1) \right) \text{FACT’} \right) 3 \]

\[ \rightarrow \left( \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times ((\text{FACT’ FACT’}) (n - 1)) \right) 3 \]
Example

Let’s try evaluating FACT on 3...

\[
\text{FACT} 3 = (\text{FACT'} \ \text{FACT'}) 3
\]

\[
= ((\lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times ((f f) (n - 1))) \ \text{FACT'}) 3
\]

\[
\rightarrow (\lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times ((\text{FACT'} \ \text{FACT'}) (n - 1))) 3
\]

\[
\rightarrow \text{if } 3 = 0 \text{ then } 1 \text{ else } 3 \times ((\text{FACT'} \ \text{FACT'}) (3 - 1))
\]
Example

Let’s try evaluating FACT on 3...

FACT 3 = (FACT’ FACT’) 3

= ((λf. λn. if n = 0 then 1 else n × ((ff) (n – 1))) FACT’) 3
→ (λn. if n = 0 then 1 else n × ((FACT’ FACT’) (n – 1))) 3
→ if 3 = 0 then 1 else 3 × ((FACT’ FACT’) (3 – 1))
→ 3 × ((FACT’ FACT’) (3 – 1))
Example

Let’s try evaluating FACT on 3...

\[ \text{FACT } 3 = (\text{FACT’ FACT’}) 3 \]

\[ = ((\lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times ((ff)(n - 1))) \text{ FACT’}) 3 \]

\[ \rightarrow (\lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times ((\text{FACT’ FACT’})(n - 1))) 3 \]

\[ \rightarrow \text{if } 3 = 0 \text{ then } 1 \text{ else } 3 \times ((\text{FACT’ FACT’})(3 - 1)) \]

\[ \rightarrow 3 \times ((\text{FACT’ FACT’})(3 - 1)) \]

\[ = 3 \times (\text{FACT } (3 - 1)) \]
Let’s try evaluating FACT on 3...

\[
\text{FACT 3} = (\text{FACT'} \ \text{FACT'}) \ 3 \\
= (\lambda f. \lambda n. \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times ((f f) (n - 1))) \ \text{FACT'} \ 3 \\
\rightarrow (\lambda n. \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times ((\text{FACT'} \ \text{FACT'}) (n - 1))) \ 3 \\
\rightarrow \textbf{if } 3 = 0 \textbf{ then } 1 \textbf{ else } 3 \times ((\text{FACT'} \ \text{FACT'}) (3 - 1)) \\
\rightarrow 3 \times ((\text{FACT'} \ \text{FACT'}) (3 - 1)) \\
= 3 \times (\text{FACT} (3 - 1)) \\
\rightarrow \ldots \\
\rightarrow 3 \times 2 \times 1 \times 1
\]
Let’s try evaluating FACT on 3...

\[
\text{FACT } 3 = (\text{FACT}' \text{ FACT}') 3 \\
= (((\lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times ((ff)(n - 1))) \text{ FACT}') 3 \\
\rightarrow (\lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times ((\text{FACT}' \text{ FACT}') (n - 1))) 3 \\
\rightarrow \text{if } 3 = 0 \text{ then } 1 \text{ else } 3 \times ((\text{FACT}' \text{ FACT}') (3 - 1)) \\
\rightarrow 3 \times ((\text{FACT}' \text{ FACT}') (3 - 1)) \\
= 3 \times (\text{FACT} (3 - 1)) \\
\rightarrow \ldots \\
\rightarrow 3 \times 2 \times 1 \times 1 \\
\rightarrow * 6
\]
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:

$$G \equiv \lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (f (n - 1))$$

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

$$g 5 \rightarrow^* 5 \times (g 4) = 5 \times ((G g) 4)$$
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:

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

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

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

$$g 5 = (G g) 5$$
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:

\[
G \triangleq \lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (f\ (n - 1))
\]

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:

\[
g\ 5 = (G\ g)\ 5
\to^* 5 \times (g\ 4)
\]
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:

\[
G \triangleq \lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (f(n - 1))
\]

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:

\[
g \ 5 = (G \ g) \ 5 \\
\rightarrow^* 5 \times (g \ 4) \\
= 5 \times ((G \ g) \ 4)
\]
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 $\lambda$-calculus, we just need a suitable combinator...
Y Combinator

The (infamous) Y combinator is defined as

\[ Y \triangleq \lambda f. (\lambda x. f(x x)) (\lambda x. f(x x)) \]

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

\[ Y \triangleq \lambda f. (\lambda x. f(xx)) (\lambda x. f(xx)) \]

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.

\[
Z \triangleq \lambda f. (\lambda x. f (\lambda y. x x y)) (\lambda x. f (\lambda y. x x y))
\]
Example

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

FACT
Example

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

\[
\text{FACT} = ZG
\]
Example

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

\[
\text{FACT} = Z \ G
\]

\[
= (\lambda f. (\lambda x. f (\lambda y. x \ x \ y)) (\lambda x. f (\lambda y. x \ x \ y))) \ G
\]
Example

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

$$
\text{FACT} = Z G \\
= (\lambda f. (\lambda x. f (\lambda y. x x y)) (\lambda x. f (\lambda y. x x y))) G \\
\rightarrow (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y))
$$
Example

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

\[
\text{FACT} = Z \ G = (\lambda f. (\lambda x. f (\lambda y. x x y)) (\lambda x. f (\lambda y. x x y))) \ G \\
\rightarrow (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) \\
\rightarrow G (\lambda y. (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) y)
\]
Example

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

FACT

\[ Z \rightarrow Z \ G \]

\[ \rightarrow (\lambda f. (\lambda x. f (\lambda y. x x y)) (\lambda x. f (\lambda y. x x y))) \ G \]

\[ \rightarrow (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) \]

\[ \rightarrow G (\lambda y. (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) y) \]

\[ = (\lambda f. \lambda n. \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \times (f(n - 1))) \]

\[ (\lambda y. (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) y) \]
Example

Let’s see \( Z \) in action, on our function \( G \).

\[
\begin{align*}
\text{FACT} & = Z \ G \\
& = (\lambda f. (\lambda x. f (\lambda y. x x y)) (\lambda x. f (\lambda y. x x y))) \ G \\
& \to (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) \\
& \to G (\lambda y. (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) y) \\
& = (\lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (f(n - 1))) \\
& \quad \quad (\lambda y. (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) y) \\
& \to \lambda n. \text{if } n = 0 \text{ then } 1 \\
& \quad \quad \text{else } n \times ((\lambda y. (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) y) (n - 1))
\end{align*}
\]
Example

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

\[
\text{FACT} = Z G
\]

\[
\Rightarrow (\lambda x. G (\lambda y. x y)) (\lambda x. G (\lambda y. x y))
\]

\[
\Rightarrow G (\lambda y. (\lambda x. G (\lambda y. x y)) (\lambda x. G (\lambda y. x y)) y)
\]

\[
= (\lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (f (n - 1)))
\]

\[
(\lambda y. (\lambda x. G (\lambda y. x y)) (\lambda x. G (\lambda y. x y)) y)
\]

\[
\Rightarrow \lambda n. \text{if } n = 0 \text{ then } 1
\]

\[
\text{else } n \times ((\lambda y. (\lambda x. G (\lambda y. x y)) (\lambda x. G (\lambda y. x y)) y) (n - 1))
\]

\[
= \beta \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (\lambda y. (Z G) y) (n - 1)
\]
Let’s see $Z$ in action, on our function $G$.

\[
\text{FACT} = Z \ G
\]

\[
\rightarrow (\lambda x. \ G \ (\lambda y. \ x \ y)) \ (\lambda x. \ G \ (\lambda y. \ x \ y))
\]

\[
\rightarrow G \ (\lambda y. \ (\lambda x. \ G \ (\lambda y. \ x \ y)) \ (\lambda x. \ G \ (\lambda y. \ x \ y)) \ y)
\]

\[
\rightarrow (\lambda f. \ \lambda n. \ \text{if} \ n = 0 \ \text{then} \ 1 \ \text{else} \ n \times (f \ (n - 1)))
\]

\[
\rightarrow \lambda n. \ \text{if} \ n = 0 \ \text{then} \ 1 \ \text{else} \ n \times ((\lambda y. \ (\lambda x. \ G \ (\lambda y. \ x \ y)) \ (\lambda x. \ G \ (\lambda y. \ x \ y)) \ y) \ (n - 1))
\]

\[
\rightarrow \lambda n. \ \text{if} \ n = 0 \ \text{then} \ 1 \ \text{else} \ n \times (\lambda y. \ (Z \ G) \ y) \ (n - 1)
\]

\[
\rightarrow \lambda n. \ \text{if} \ n = 0 \ \text{then} \ 1 \ \text{else} \ n \times ((Z \ G) \ (n - 1))
\]
Example

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

\[
\text{FACT} = Z \ G
\]
\[
\rightarrow \ (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y))
\]
\[
\rightarrow \ G (\lambda y. (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) y)
\]
\[
\rightarrow \ (\lambda f. \lambda n. \text{if } n = 0 \text{ \textbf{then} } 1 \text{ \textbf{else} } n \times (f (n - 1)))
\]
\[
\quad (\lambda y. (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) y)
\]
\[
\rightarrow \ \lambda n. \text{if } n = 0 \text{ \textbf{then} } 1
\]
\[
\quad \text{else } n \times ((\lambda y. (\lambda x. G (\lambda y. x x y)) (\lambda x. G (\lambda y. x x y)) y) (n - 1))
\]
\[
\Rightarrow \ \lambda n. \text{if } n = 0 \text{ \textbf{then} } 1 \text{ \textbf{else} } n \times (\lambda y. (Z \ G) y) (n - 1)
\]
\[
\Rightarrow \ \lambda n. \text{if } n = 0 \text{ \textbf{then} } 1 \text{ \textbf{else} } n \times ((Z \ G) (n - 1))
\]
\[
= \ \lambda n. \text{if } n = 0 \text{ \textbf{then} } 1 \text{ \textbf{else} } n \times (\text{FACT} (n - 1))
\]
Other fixed point combinators

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

\[ Y_k \triangleq (L L L L L L L L L L L L L L L L L L L L L L L) \]

where

\[ L \triangleq \lambda abcdefghijklmnopqrstuvwxyzr. (r (t h i s i s a f i x e d p o i n t c o m b i n a t o r)) \]
To gain some more intuition for fixed point combinators, let’s derive a combinator $\Theta$ originally discovered by Turing.

Let $\Theta$ be 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:

$$\Theta' = \lambda t. \lambda f. f(t tf)$$

$$\Theta = \Theta' \Theta'$$
Turing’s Fixed Point Combinator

To gain some more intuition for fixed point combinators, let’s derive a combinator \( \Theta \) 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 $\Theta$ 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:

$$\Theta = \lambda f. f (\Theta f)$$
To gain some more intuition for fixed point combinators, let’s derive a combinator $\Theta$ 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:

$$\Theta' \triangleq \lambda t. \lambda f. f (t t f)$$

$$\Theta \triangleq \Theta' \Theta'$$
Example

\[ \text{FACT} = \Theta G \]
\[ \text{FACT} = \Theta G \]

\[ = ((\lambda t. \lambda f. f(t t f)) (\lambda t. \lambda f. f(t t f))) G \]
Example

\[ \text{FACT} = \Theta G \]
\[ = (\lambda t. \lambda f. f(t t f)) (\lambda t. \lambda f. f(t t f)) G \]
\[ \rightarrow (\lambda f. f((\lambda t. \lambda f. f(t t f)) (\lambda t. \lambda f. f(t t f)) f)) G \]
$\theta$ Example

\[
\text{FACT} \equiv \Theta G
\]
\[
= ((\lambda t. \lambda f. f (t t f)) (\lambda t. \lambda f. f (t t f))) G
\]
\[
\rightarrow (\lambda f. f ((\lambda t. \lambda f. f (t t f)) (\lambda t. \lambda f. f (t t f))) f)) G
\]
\[
\rightarrow G ((\lambda t. \lambda f. f (t t f)) (\lambda t. \lambda f. f (t t f))) G)
\]
\( \theta \) Example

FACT \( = \Theta G \)

\[ = ((\lambda t. \lambda f. f (t t f)) (\lambda t. \lambda f. f (t t f))) G \]

\[ \rightarrow (\lambda f. f ((\lambda t. \lambda f. f (t t f)) (\lambda t. \lambda f. f (t t f)) f)) G \]

\[ \rightarrow G ((\lambda t. \lambda f. f (t t f)) (\lambda t. \lambda f. f (t t f)) G) \]

\[ = G (\Theta G) \]
$\theta$ Example

$$\text{FACT} = \Theta G$$

$$= \left((\lambda t. \lambda f. f (ttf)) (\lambda t. \lambda f. f (ttf))\right) G$$

$$\rightarrow (\lambda f. f ((\lambda t. \lambda f. f (ttf)) (\lambda t. \lambda f. f (ttf)) f)) G$$

$$\rightarrow G ((\lambda t. \lambda f. f (ttf)) (\lambda t. \lambda f. f (ttf)) G)$$

$$= G (\Theta G)$$

$$= (\lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (f(n - 1))) (\Theta G)$$

$$\rightarrow \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times ((\Theta G)(n - 1))$$

$$= \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (\text{FACT}(n - 1))$$