The Y Combinator

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science
Objectives

- Understand how to allow functions to call themselves — even when they don’t have names.
- Understand how to develop a general combinator $Y$ to implement recursion.
Recursion

Suppose we want to implement

\[ f(n) = f(n+1) \]
Step 1

The outline of the function would look like

$$\lambda n. (f (inc \ n))$$

But, how does $f$ get to know itself?
Step 2

Maybe we can tell $f$ by having it take it’s own name as a parameter.

$$\lambda f.\lambda n. (f (inc\ n))$$

So then we pass a copy of $f$ to itself...

$$(\lambda f.\lambda n. (f (inc\ n))) (\lambda f.\lambda n. (f (inc\ n)))$$

But now $f$ must pass itself into itself... so we have

$$(\lambda f.\lambda n. ((f\ f) (inc\ n))) (\lambda f.\lambda n. ((f\ f) (inc\ n)))$$
Expanding a Church Numeral

Consider how this is similar to the operation of Church numerals.

\[
((f_5 f) x) \\
\rightarrow (f ((f_4 f) x)) \\
\rightarrow (f (f ((f_3 f) x)))) \\
\rightarrow (f (f (f ((f_2 f) x)))) \\
\rightarrow (f (f (f (f (f x)))))
\]

So...

\[
((f_n f) x) \rightarrow (f ((f_{n-1} f) x))
\]

What would it look like to have an \( f_\infty \)?
The Y Combinator

Consider this pattern:

$$((f \infty f) x) \rightarrow (f ((f \infty f) x))$$

- What can you tell about $f$? About $f_\infty$?
- Definition: combinator = higher order function that produces its result only though function application.
- The problem with the above function is that there’s no way out. How can we stop the function when we are done?
Coding the Y Combinator

\[(Yf) \rightarrow f(Yf)\]

So...

\[Y = \lambda f. (\lambda y. f(y))\lambda y. f(y)\]

The function \(f\) must take \((Yf)\) as an argument.

\[\begin{align*}
(YF) &= (\lambda f. (\lambda y. f(y))\lambda y. f(y))F \\
&= (\lambda y. F(y))\lambda y. F(y) \\
&= F((\lambda y. F(y))\lambda y. F(y)) \\
&= F(YF)
\end{align*}\]
Example

```
1 fact n =
2    if n < 1 then 1
3    else n * (fact (n-1))
```

In λ-calculus:

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

Then we have:

\[ \lambda n. \]
\[ Y fact \rightarrow \text{if } n < 1 \text{ then } 1 \]
\[ \quad \text{else } n \times ((Y fact)(n-1)) \]
You can use $\lambda$-calculus to represent itself using these techniques. You already have everything you need to do it. You can see the details in Torben Æ. Mogensen’s paper Efficient Self-Interpretations in lambda Calculus, in the Journal of Functional Programming v2 n3.