The Y-Combinator

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives

- Use self-application to allow functions to call themselves – even when they don't have names.
- 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 its 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.

$$
\begin{aligned}
&((f_5\ f)\ x) \\
\rightarrow\ & (f\ ((f_4\ f)\ x)) \\
\rightarrow\ & (f\ (f\ ((f_3\ f)\ x))) \\
\rightarrow\ & (f\ (f\ (f\ ((f_2\ f)\ x)))) \\
\rightarrow\ & (f\ (f\ (f\ (f\ (f\ x)))))
\end{aligned}
$$

So ...

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

What would it look like to have an f_∞?

The Y-Combinator

Consider this pattern:

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

- What can you tell about f? About f_∞?
- 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

$$(Y\,f) \;\rightarrow\; f\,(Y\,f)$$

So...

$$Y \;=\; \lambda f.(\lambda y.f\,(y\,y))\;\lambda y.f\,(y\,y))$$

The function f must take $(Y\,f)$ as an argument.

$$\begin{aligned}
(Y\,F) &= (\lambda f.(\lambda y.f\,(y\,y))\;\lambda y.f\,(y\,y))\;F \\
&= (\lambda y.F\,(y\,y))\;\lambda y.F\,(y\,y) \\
&= F\,((\lambda y.F\,(y\,y))\lambda y.F\,(y\,y)) \\
&= F\,(Y\,F)
\end{aligned}$$

Example

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

In λ-calculus:

$$\begin{array}{l}\lambda f.\lambda n.\\ \quad \texttt{if}\ n < 1\ \texttt{then}\ 1\\ \quad\quad \texttt{else}\ n * (f\ (n-1))\end{array}$$

Then we have:

$$Y\ \mathit{fact} \rightarrow \begin{array}{l}\lambda n.\\ \quad \texttt{if}\ n < 1\ \texttt{then}\ 1\\ \quad\quad \texttt{else}\ n * ((Y\ \mathit{fact})\ (n-1))\end{array}$$

Further Reading

- You can use λ-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.