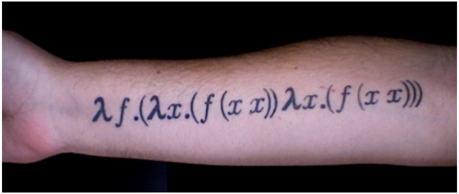
Untyped λ-Calculus, Informally

Alex Vondrak

ajvondrak@csupomona.edu

April 28, 2010



Do-Over!

- Almost a year ago, I did a λ-calculus talk
  - Too long[1]
  - Too fast
  - Too much
- λ-calculus isn't that bad
  - In fact, it's supposed to be *simple*
  - Formalizes function definition and application
  - Last time: formal $\longrightarrow$ fun
  - Full-on formality takes awhile

[1] That's what *she* said!

You Already Know λ-Calculus

Functions

- Remember math?

Examples

$$f(x) = x$$
$$g(x) = x^2$$
$$h(x) = f(x) \cdot g(x)$$

- All of these functions are named
- How do we formalize them?

You Already Know λ-Calculus

Programming

Language	Syntax
λ-calculus	$(\lambda x \,.\, x)$
Common Lisp/Scheme	`(lambda (x) x)`
Python	`lambda x: x`
Ruby	`lambda {|x| x}`
Haskell	`\x -> x`
C#	`x => x`
Javascript	`function(x) {x;}`
OCaml	`fun x -> x`
SML	`fn x => x`
⋮	⋮

First-class functions are supported by even more languages: Perl, PHP, Erlang, Lua, Tcl/Tk, Io, Scala, D, Smalltalk, . . .

λ-Calculus In Theory

Definitions (Syntax & Semantics)

$$
\begin{array}{lll}
\langle term \rangle & \rightarrow & \langle identifier \rangle \\
 & | & (\quad \boldsymbol{\lambda}\langle identifier \rangle \quad \mathbf{.} \quad \langle term \rangle \quad) \\
 & | & (\quad \langle term \rangle \quad \langle term \rangle \quad)
\end{array}
$$

(out) $$\frac{}{(\lambda x\ .\ A)B \quad \longrightarrow_\beta \quad A[x \mapsto B]}$$

(left) $$\frac{A \quad \longrightarrow_\beta \quad A'}{AB \quad \longrightarrow_\beta \quad A'B}$$

(in) $$\frac{A \quad \longrightarrow_\beta \quad A'}{\lambda x\ .\ A \quad \longrightarrow_\beta \quad \lambda x\ .\ A'}$$

(right) $$\frac{B \quad \longrightarrow_\beta \quad B'}{AB \quad \longrightarrow_\beta \quad AB'}$$

λ-Calculus In Practice

Before

$$f(x) = x$$
$$g(x) = x^2$$
$$h(x) = f(x) \cdot g(x)$$

After

$$f = (\lambda x\ .\ x)$$
$$g = (\lambda x\ .\ x^2)$$
$$h = (\lambda x\ .\ (f\ \ x) \cdot (g\ \ x))$$

λ-Calculus In Practice (Basically)

Before

$$f(x) = x$$
$$g(x) = x^2$$
$$h(x) = f(x) \cdot g(x)$$

After

$$f = (\lambda x\ .\ x)$$
$$g = (\lambda x\ .\ x^2)$$
$$h = (\lambda x\ .\ (f\ \ x) \cdot (g\ \ x))$$

Strictly, in the untyped λ-calculus, we don't have names, datatypes, operators, objects, methods, or *anything*; just functions.

Just Functions?

- What can you represent with *just* functions?
- Anything a computer can!
  - Natural numbers
  - Booleans
  - Tuples
  - Linked lists
  - Recursion
  - ...
- λ-calculus is the smallest *interesting* programming language
- *A Correspondence between ALGOL 60 and Church's Lambda-Notation* (Landin, 1965)
- *System F with Type Equality Coercions* (Sulzmann, Chakravarty, Peyton Jones, and Donnelly, 2007)

History

- Early 1930s: Church, Kleene, and Rosser papers
  - 1932: Church formalizes λ-calculus
  - 1933: Church numerals
  - 1935: Kleene-Rosser paradox
- 1936: The Church-Turing thesis
  - Church answers the decision problem
  - Independently and almost immediately afterwards, so does Alan Turing
- 1936-1938: Alan Turing went to Princeton, taking Church as his doctoral advisor. Among other things, this gave us the Turing fixpoint combinator Θ in 1937.
- 1940: Church publishes a reformulation of type theory based on λ-calculus, which is a foundation for type-theoretic work today.

Extending λ-Calculus

- Though theoretically we can make everything a function, in practice we "cheat"
- To make our lives easier, assume we have
  - Natural numbers ($\lambda_{\mathbb{N}}$-calculus)
  - Booleans ($\lambda_{\mathbb{NB}}$-calculus)
- We use these as shorthand — it's still untyped
- Even more extensions are possible
  - Annotate each program variable with a type variable — ($\lambda x : T\ .\ etc$)
  - Give semantic rules that handle types soundly
  - This is the foundation of type theory

Case Study: Linked Lists

- Linked lists are ordered sequences of `cons` cells
- End each list with `nil`

Definitions

$$\texttt{cons} = (\lambda h\ .\ (\lambda t\ .\ (\lambda f\ .\ ((f\ h)\ t))))$$
$$\texttt{head} = (\lambda c\ .\ (c\ (\lambda h\ .\ (\lambda t\ .\ h))))$$
$$\texttt{tail} = (\lambda c\ .\ (c\ (\lambda h\ .\ (\lambda t\ .\ t))))$$
$$\texttt{nil} = (\lambda f\ .\ \texttt{T})$$
$$\texttt{null} = (\lambda c\ .\ (c\ (\lambda h\ .\ (\lambda t\ .\ \texttt{F}))))$$

Case Study: Linked Lists

Variable Length

Examples

$$
\begin{aligned}
((\texttt{cons}\ 1)\ \texttt{nil}) &\longrightarrow_\beta (\lambda f\ .\ ((f\ 1)\ \texttt{nil})) \\
(\texttt{cons}\ 2\ (\texttt{cons}\ 1\ \texttt{nil})) &\longrightarrow_\beta \left(\lambda f\ .\ f\ 2\ \underset{\sim\sim\sim}{(\lambda f\ .\ f\ 1\ \texttt{nil})}\right) \\
(\texttt{cons}\ 3\ (\texttt{cons}\ 2\ (\texttt{cons}\ 1\ \texttt{nil}))) &\longrightarrow_\beta \ldots
\end{aligned}
$$

Case Study: Linked Lists

Accessing Elements

- Since linked lists are just nested pairs, we can access elements by individually accessing the head or tail of each sublist

Example

$$
\begin{aligned}
& (\texttt{head}\ (\texttt{cons}\ 2\ (\texttt{cons}\ 1\ \texttt{nil}))) \\
\equiv\ & \underline{((\lambda c\ .\ c\ (\lambda h\ t\ .\ h))\ (\texttt{cons}\ 2\ (\texttt{cons}\ 1\ \texttt{nil})))} \\
\longrightarrow_\beta\ & ((\texttt{cons}\ 2\ (\texttt{cons}\ 1\ \texttt{nil}))\ (\lambda h\ t\ .\ h)) \\
\equiv\ & \underline{((\lambda f\ .\ f\ 2\ (\lambda f\ .\ f\ 1\ \texttt{nil}))\ (\lambda h\ t\ .\ h))} \\
\longrightarrow_\beta\ & \underline{\underline{(\lambda h\ t\ .\ h)\ 2}\ (\lambda f\ .\ f\ 1\ \texttt{nil})} \\
\longrightarrow_\beta\ & 2
\end{aligned}
$$

Case Study: Linked Lists

The Empty List

Example

$$
\begin{aligned}
& (\texttt{null nil}) \\
\equiv\ & \underline{((\lambda c\ .\ c\ (\lambda h\ t\ .\ \texttt{F}))\ \ (\lambda f\ .\ \texttt{T}))} \\
\longrightarrow_\beta\ & \underline{((\lambda f\ .\ \texttt{T})\ \ (\lambda h\ t\ .\ \texttt{F}))} \\
\longrightarrow_\beta\ & \texttt{T}
\end{aligned}
$$

Case Study: Linked Lists

The Empty List

Example

$$
\begin{aligned}
& (\texttt{null}\ (\texttt{cons}\ 1\ \texttt{nil})) \\
\equiv\ & \underline{((\lambda c\ .\ c\ (\lambda h\ t\ .\ \texttt{F}))\ \ (\lambda f\ .\ f\ 1\ \texttt{nil}))} \\
\longrightarrow_\beta\ & \underline{((\lambda f\ .\ f\ 1\ \texttt{nil})\ \ (\lambda h\ t\ .\ \texttt{F}))} \\
\longrightarrow_\beta\ & ((\underline{(\lambda h\ t\ .\ \texttt{F})\ \ 1})\ \texttt{nil}) \\
\longrightarrow_\beta\ & \underline{((\lambda t\ .\ \texttt{F})\ \ \texttt{nil})} \\
\longrightarrow_\beta\ & \texttt{F}
\end{aligned}
$$

Case Study: Recursion

- Recursion is a natural way to iterate through a linked list

Example

$$\begin{aligned} \texttt{sum} = \lambda l \,.\, \texttt{if}\ & (\texttt{null}\ l) \\ & 0 \\ & (l\ \ (\lambda h\ .\ (\lambda t\ .\ h + (\texttt{sum}\ t)))) \end{aligned}$$

- But without naming, how can we make functions recursive?

Case Study: Recursion

Fixpoints

Definition

A *fixpoint* of a function f is a value fix_f such that

$$f(\mathrm{fix}_f) = \mathrm{fix}_f$$

Example

Consider the algebraic function $f(x) = x^2$. f has the fixpoints 0 and 1:

$$f(0) = 0^2 = 0 \qquad\qquad f(1) = 1^2 = 1$$

But -1 is not a fixpoint, because

$$f(-1) = (-1)^2 = 1 \neq -1$$

Case Study: Recursion

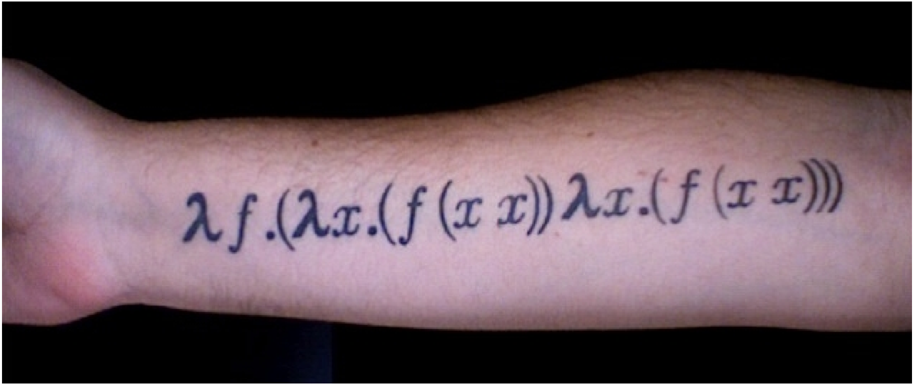
Fixpoint Combinators

- In the untyped λ-calculus, every term has a fixpoint
- Fixpoints can be calculated with fixpoint combinators
  - Y Combinator
  - Alan Turing's fixpoint combinator, Θ
  - Others
- Therefore, given a λ-calculus function f, we have

$$(f\ (\mathtt{Y}\ f)) =_\beta (\mathtt{Y}\ f)$$
$$(f\ (\Theta\ f)) =_\beta (\Theta\ f)$$

Case Study: Recursion

Y Combinator



Case Study: Recursion

Using Fixpoints

- How do fixpoints help us with recursion?
- Let f take a parameter in order to refer to itself, then use a fixpoint combinator

Example

Let

$$f = (\lambda rec\ .\ \lambda l\ .\ \texttt{if}\ (\texttt{null}\ l)\\ 0\\ (l\ (\lambda h\ .\ (\lambda t\ .\ h + (rec\ t)))))$$

Then $\texttt{sum} = (\texttt{Y}\ f) =_\beta (f\ \underset{\sim}{(\texttt{Y}\ f)}) = (f\ \texttt{sum})$

Summary

- λ-calculus is a tiny, axiomatic tool used in
  - computability
  - compilers
  - formal semantics
  - programming language theory
  - type theory
  - logic
  - math
  - ...
- Extensions to the untyped λ-calculus make it much richer
- Simple Yet Effective™