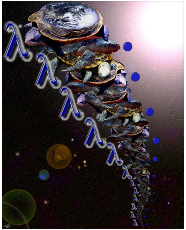
Functions all the way down!

Lambda Calculus and Church Encoding

Ivan Lazar Miljenovic

Maths PhD Journal Club

14 May, 2009



Outline

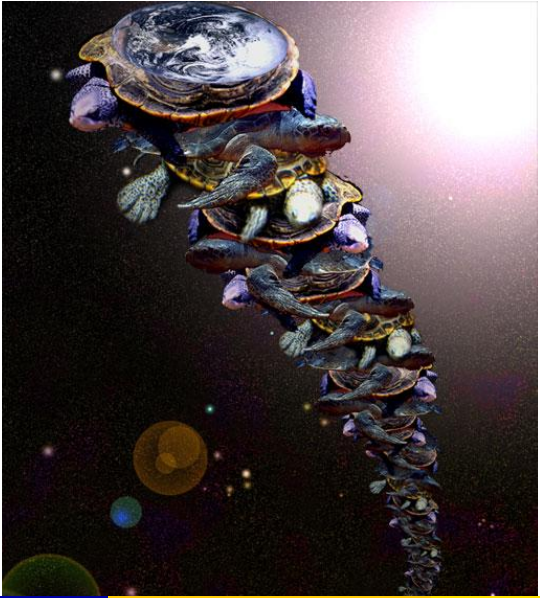
1. Introduction
2. Theory of Computing
3. The λ-Calculus
4. Church Encoding

Introduction

1 Introduction

- Turtles all the way down...
- Programming languages and Functions
- Functions all the way down

Turtles all the way down...



Turtles all the way down...

From Stephen Hawking's *A Brief History of Time*

A well-known scientist (some say it was Bertrand Russell) once gave a public lecture on astronomy. He described how the earth orbits around the sun and how the sun, in turn, orbits around the center of a vast collection of stars called our galaxy.
At the end of the lecture, a little old lady at the back of the room got up and said: "What you have told us is rubbish. The world is really a flat plate supported on the back of a giant tortoise." The scientist gave a superior smile before replying, "What is the tortoise standing on?" "You're very clever, young man, very clever," said the old lady. "But it's *turtles all the way down!*"

Programming languages and Functions

Most "standard" programming languages There is data, and then functions that act on that data.

Programming languages and Functions

Most "standard" programming languages There is data, and then functions that act on that data.

Functional languages Functions are "first class citizens" (Cristopher Strachey, mid-1960s).

Programming languages and Functions

Most "standard" programming languages There is data, and then functions that act on that data.

Functional languages Functions are "first class citizens" (Cristopher Strachey, mid-1960s).

LISP family Everything is data; all data is code.

Programming languages and Functions

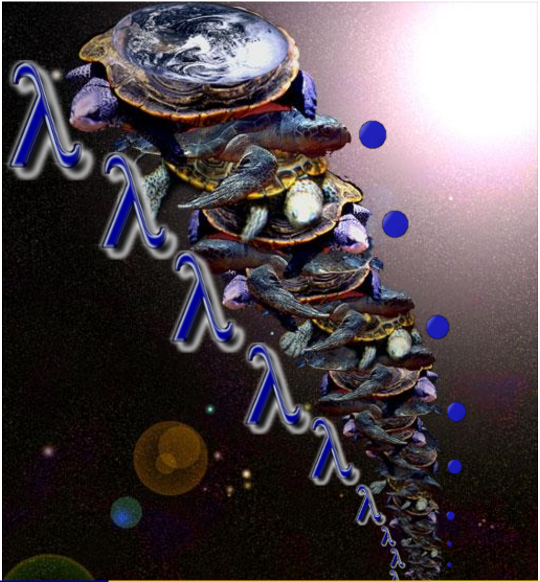
Most "standard" programming languages There is data, and then functions that act on that data.

Functional languages Functions are "first class citizens" (Cristopher Strachey, mid-1960s).

LISP family Everything is data; all data is code.

What about replacing data with functions?

Functions all the way down



Theory of Computing

2 Theory of Computing

- The Entscheidungsproblem
- Solutions to the Entscheidungsproblem

The Entscheidungsproblem

The decidability problem

Posed by David Hilbert in 1928:

The Entscheidungsproblem

The decidability problem

Posed by David Hilbert in 1928:

Definition (The Entscheidungsproblem)

Given a description of a formal language and a mathematical statement in that language, determine if the statement is true or false.

The Entscheidungsproblem

The decidability problem

Posed by David Hilbert in 1928:

Definition (The Entscheidungsproblem)

Given a description of a formal language and a mathematical statement in that language, determine if the statement is true or false.

Original Statement

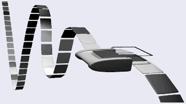
Generalisation of his question "Is mathematics *decidable*".

Solutions to the Entscheidungsproblem

The Halting Problem

January, 1937

On Computable Numbers, with an Application to the Entscheidungsproblem by Alan Turing



Solutions to the Entscheidungsproblem

The Halting Problem

15 April, 1936

An Unsolvable Problem of Elementary Number Theory by Alonzo Church

Solutions to the Entscheidungsproblem

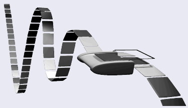
The Halting Problem

15 April, 1936

An Unsolvable Problem of Elementary Number Theory by Alonzo Church

January, 1937

On Computable Numbers, with an Application to the Entscheidungsproblem by Alan Turing



Theorem (Church-Turing Thesis)

Solutions to the Entscheidungsproblem

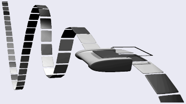
The Halting Problem

15 April, 1936

An Unsolvable Problem of Elementary Number Theory by Alonzo Church

January, 1937

On Computable Numbers, with an Application to the Entscheidungsproblem by Alan Turing



Theorem (Church-Turing Thesis)

- *Every effectively calculable function is a computable function.*

Solutions to the Entscheidungsproblem

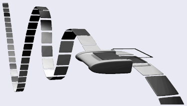
The Halting Problem

15 April, 1936

An Unsolvable Problem of Elementary Number Theory by Alonzo Church

January, 1937

On Computable Numbers, with an Application to the Entscheidungsproblem by Alan Turing



Theorem (Church-Turing Thesis)

- *Every effectively calculable function is a computable function.*
- *λ-Calculus, Turing machines, etc. are equivalent.*

The λ-Calculus

3 The λ-Calculus

- What is the λ-Calculus?
- Definition of the λ-Calculus
- Informal Examples
- Usage of the λ-Calculus

What is the λ-Calculus?

Types and Programming Languages

What is the λ-Calculus?

Types and Programming Languages

- "…a formal system invented in the 1920s by Alonzo Church, in which all computation is reduced to the basic operations of function definition and application."

What is the λ-Calculus?

Types and Programming Languages

- "...a formal system invented in the 1920s by Alonzo Church, in which all computation is reduced to the basic operations of function definition and application."
- "Its importance arises from the fact that it can be viewed simultaneously as a simple programming language *in which* computations can be described and as a mathematical object *about which* rigorous statements can be proved."

What is the λ-Calculus?

A Brief, Incomplete, and Mostly Wrong History of Programming Languages

What is the λ-Calculus?

A Brief, Incomplete, and Mostly Wrong History of Programming Languages

1936 Alonzo Church also invents every language that will ever be but does it better. His lambda calculus is ignored because it is insufficiently C-like. This criticism occurs in spite of the fact that C has not yet been invented.

What is the λ-Calculus?

A Brief, Incomplete, and Mostly Wrong History of Programming Languages

1936 Alonzo Church also invents every language that will ever be but does it better. His lambda calculus is ignored because it is insufficiently C-like. This criticism occurs in spite of the fact that C has not yet been invented.

1970 ...Lambdas are relegated to relative obscurity until Java makes them popular by not having them.

Definition of the λ-Calculus

A term t in the λ-Calculus is one of three things:

Definition of the λ-Calculus

A term t in the λ-Calculus is one of three things:

x A variable.

Definition of the λ-Calculus

A term t in the λ-Calculus is one of three things:

x A variable.

$\lambda\, x.\, t$ The abstraction of a variable x from a term.

Definition of the λ-Calculus

A term t in the λ-Calculus is one of three things:

x	A variable.
$\lambda x. t$	The abstraction of a variable x from a term.
$t\ t$	The application of one term to another.

Definition of the λ-Calculus

A term t in the λ-Calculus is one of three things:

x	A variable.
$\lambda x.\, t$	The abstraction of a variable x from a term.
$t\ t$	The application of one term to another.

Also use parentheses for grouping.

Definition of the λ-Calculus

Functions are:

Definition of the λ-Calculus

Functions are:

- Anonymous

Definition of the λ-Calculus

Functions are:

- Anonymous
- Unary

Definition of the λ-Calculus

Functions are:

- Anonymous
- Unary
- Higher-order

Definition of the λ-Calculus

Functions are:

- Anonymous
- Unary
- Higher-order
- Recursive

Definition of the λ-Calculus

Functions are:

- Anonymous
- Unary
- Higher-order
- Recursive
- Left-associative: $abc = (ab)c$

Definition of the λ-Calculus

Functions are:

- Anonymous
- Unary
- Higher-order
- Recursive
- Left-associative: $abc = (ab)c$
- Extend as far right as possible

Informal Examples

Informal Examples

- Identity function:

$$\lambda\, x.\, x$$

Informal Examples

- Identity function:

$$\lambda\, x.\, x$$

- Mathematics:

$$(\lambda\, x.\, x + 2)\, 3$$

Informal Examples

- Identity function:

$$\lambda\, x.\, x$$

- Mathematics:

$$(\lambda\, x.\, x + 2)\, 3$$

- Multiple arguments:

$$\lambda\, x.\, \lambda\, y.\, x + y$$

Informal Examples

- Identity function:

$$\lambda\, x.\, x$$

- Mathematics:

$$(\lambda\, x.\, x + 2)\; 3$$

- Multiple arguments:

$$\lambda\, x.\, \lambda\, y.\, x + y$$

- Factorials:

$$\begin{aligned} g &= \lambda\, f.\, \lambda\, n.\, \begin{cases} 1 & n = 0 \\ n \times f(n-1) & n > 0 \end{cases} \\ Y &= \lambda\, h.\, (\lambda\, x.\, h\,(x\, x))\, (\lambda\, x.\, h\,(x\, x)) \\ \text{fact} &= \lambda\, n.\, Y\, g\, n \end{aligned}$$

Usage of the λ-Calculus

Usage of the λ-Calculus in programming languages:

Usage of the λ-Calculus

Usage of the λ-Calculus in programming languages:

Functional Languages Based on the λ-Calculus, treat computation as the evaluation of mathematical functions and typically avoids state and mutable data.

Usage of the λ-Calculus

Usage of the λ-Calculus in programming languages:

Functional Languages Based on the λ-Calculus, treat computation as the evaluation of mathematical functions and typically avoids state and mutable data.

LISP family Lisp is the second oldest programming language after FORTRAN; loosely based on the λ-Calculus but technically classed as multi-paradigm.

Usage of the λ-Calculus

Usage of the λ-Calculus in programming languages:

Functional Languages Based on the λ-Calculus, treat computation as the evaluation of mathematical functions and typically avoids state and mutable data.

LISP family Lisp is the second oldest programming language after FORTRAN; loosely based on the λ-Calculus but technically classed as multi-paradigm.

Anonymous Functions Found in many other languages: C#, C++Ox, Matlab, etc.

Church Encoding

4 Church Encoding

- What is Church Encoding?
- Church Numerals
- Peano Arithmetic
  - Isomorphism between Church and Peano
- Other operations on Church Numerals
- Other Church Encodings
- Representation of Church Encodings
- Are Church Encodings practical?

What is Church Encoding?

What is Church Encoding?

Definition (Church Encoding)

A means of embedding data and operations on them by using the λ-Calculus.

What is Church Encoding?

Definition (Church Encoding)

A means of embedding data and operations on them by using the λ-Calculus.

Encodings are available for:

What is Church Encoding?

Definition (Church Encoding)

A means of embedding data and operations on them by using the λ-Calculus.

Encodings are available for:

- Natural numbers

What is Church Encoding?

Definition (Church Encoding)

A means of embedding data and operations on them by using the λ-Calculus.

Encodings are available for:

- Natural numbers
- Booleans

What is Church Encoding?

Definition (Church Encoding)

A means of embedding data and operations on them by using the λ-Calculus.

Encodings are available for:

- Natural numbers
- Booleans
- Pairs

What is Church Encoding?

Definition (Church Encoding)

A means of embedding data and operations on them by using the λ-Calculus.

Encodings are available for:

- Natural numbers
- Booleans
- Pairs
- Lists

Church Numerals

Devised by Alonzo Church in 1941 to represent natural numbers:

Church Numerals

Devised by Alonzo Church in 1941 to represent natural numbers:

Church Numerals

$$\begin{aligned}
\mathbf{0} &\equiv \lambda f.\, \lambda x.\, x \\
\mathbf{1} &\equiv \lambda f.\, \lambda x.\, f\, x \\
\mathbf{2} &\equiv \lambda f.\, \lambda x.\, f\, (f\, x) \\
\mathbf{3} &\equiv \lambda f.\, \lambda x.\, f\, (f\, (f\, x)) \\
\ldots \\
\mathbf{n} &\equiv \lambda f.\, \lambda x.\, f^n\, x
\end{aligned}$$

where $f^0 = id$, $f^{n+1} = f \cdot f^n$.

Church Numerals

Alternatively:

Church Numerals

Alternatively:

Church Numerals

$$
\begin{aligned}
\mathbf{0} &\equiv \lambda\, f.\, id \\
\mathbf{1} &\equiv \lambda\, f.\, f \\
\mathbf{2} &\equiv \lambda\, f.\, f \cdot f \\
\mathbf{3} &\equiv \lambda\, f.\, f \cdot f \cdot f \\
\ldots & \\
\mathbf{n} &\equiv \lambda\, f.\, f^n
\end{aligned}
$$

where $f^0 = id$, $f^{n+1} = f \cdot f^n$.

Church Numerals

Operations on Church Numerals (due to Rosser):

Church Numerals

Operations on Church Numerals (due to Rosser):

Successor function: $succ\ n = \lambda f.\, f \cdot n\, f$

Church Numerals

Operations on Church Numerals (due to Rosser):

Successor function: $succ\ n = \lambda f.\, f \cdot n\, f$

Addition: $plus\ m\ n = \lambda f.\, m\, f \cdot n\, f$

Church Numerals

Operations on Church Numerals (due to Rosser):

Successor function: $succ\, n = \lambda f.\, f \cdot n\, f$

Addition: $plus\, m\, n = \lambda f.\, m\, f \cdot n\, f$

Multiplication: $mult\, m\, n = m \cdot n$

Church Numerals

Operations on Church Numerals (due to Rosser):

Successor function: $succ\; n = \lambda f.\, f \cdot n\, f$

Addition: $plus\; m\; n = \lambda f.\, m\, f \cdot n\, f$

Multiplication: $mult\; m\; n = m \cdot n$

Exponentiation: $pow\; m\; n = n\; m$

Peano Arithmetic

We can derive the Church Numerals from the unary representation of the natural numbers, also known as the Peano numeral system.

Peano Arithmetic

We can derive the Church Numerals from the unary representation of the natural numbers, also known as the Peano numeral system.

Definition (Peano Numbers)

```
data Nat = Zero | Succ Nat

one :: Nat
one = Succ Zero
```

Peano Arithmetic

We can derive the Church Numerals from the unary representation of the natural numbers, also known as the Peano numeral system.

Definition (Peano Numbers)

```
data Nat = Zero | Succ Nat

one :: Nat
one = Succ Zero
```

We can operate on Peano numbers using what is known as a *fold* function:

Peano Arithmetic

We can derive the Church Numerals from the unary representation of the natural numbers, also known as the Peano numeral system.

Definition (Peano Numbers)

```
data Nat = Zero | Succ Nat

one :: Nat
one = Succ Zero
```

We can operate on Peano numbers using what is known as a *fold* function:

Definition (Fold on Peano Numbers)

```
fold                   :: (a → a) → a → Nat → a
fold succ zero Zero     = zero
fold succ zero (Succ n) = succ (fold succ zero n)
```

Peano Arithmetic

Operations on Nat

```
plus, mult, pow :: Nat → Nat → Nat

plus m n = fold Succ n m

mult m n = fold (add n) Zero m

pow  m n = fold (mult m) one n
```

Peano Arithmetic

Isomorphism between Church and Peano

We can make an isormorphism between Church Numerals and Peano Numbers:

Peano Arithmetic

Isomorphism between Church and Peano

We can make an isormorphism between Church Numerals and Peano Numbers:

Converting between the two

```
type Church a = (a → a) → a → a

nat    :: Church Nat → Nat
nat c = c Succ Zero

church    :: Nat → Church a
church n = λ succ → λ zero → fold succ zero n
```

Peano Arithmetic

Isomorphism between Church and Peano

This leads to a new formulation for operations on Church Numerals:

Peano Arithmetic

Isomorphism between Church and Peano

This leads to a new formulation for operations on Church Numerals:

Successor function: $succ\ c = \lambda\, s.\, \lambda\, z.\, s\,(c\, s\, z)$

Peano Arithmetic

Isomorphism between Church and Peano

This leads to a new formulation for operations on Church Numerals:

Successor function: $succ\ c = \lambda\, s.\ \lambda\, z.\ s\,(c\, s\, z)$

Addition: $plus\ m\ n = m\ succ\ n$

Peano Arithmetic

Isomorphism between Church and Peano

This leads to a new formulation for operations on Church Numerals:

Successor function: $succ\ c = \lambda\, s.\, \lambda\, z.\, s\, (c\, s\, z)$

Addition: $plus\ m\ n = m\ succ\ n$

Multiplication: $mult\ m\ n = m\, (n+)\, \mathbf{0}$

Peano Arithmetic

Isomorphism between Church and Peano

This leads to a new formulation for operations on Church Numerals:

Successor function: $succ\ c = \lambda\, s.\, \lambda\, z.\, s\,(c\, s\, z)$

Addition: $plus\ m\ n = m\, succ\ n$

Multiplication: $mult\ m\ n = m\,(n+)\,\mathbf{0}$

Exponentiation: $pow\ m\ n = n\,(m\times)\,\mathbf{1}$

Peano Arithmetic

Isomorphism between Church and Peano

This leads to a new formulation for operations on Church Numerals:

Successor function: $succ\ c = \lambda\, s.\, \lambda\, z.\, s\,(c\, s\, z)$

Addition: $plus\ m\ n = m\ succ\ n$

Multiplication: $mult\ m\ n = m\,(n+)\,\mathbf{0}$

Exponentiation: $pow\ m\ n = n\,(m\times)\,\mathbf{1}$

Completely different formulation from before!

Other operations on Church Numerals

We can also define operations such as *equal*, *pred* and *subtract* on Church Numerals . . .

Other operations on Church Numerals

We can also define operations such as *equal*, *pred* and *subtract* on Church Numerals . . .

. . . but they get very messy very quickly.

Other Church Encodings

Other Church Encodings of interest:

Other Church Encodings

Other Church Encodings of interest:

Church Booleans:

$$\textbf{true} \equiv \lambda\, t.\, \lambda\, f.\, t$$
$$\textbf{false} \equiv \lambda\, t.\, \lambda\, f.\, f$$

Other Church Encodings

Other Church Encodings of interest:

Church Booleans:

$$\mathbf{true} \equiv \lambda\, t.\, \lambda\, f.\, t$$
$$\mathbf{false} \equiv \lambda\, t.\, \lambda\, f.\, f$$

Church Pairs:

$$\mathbf{pair} \equiv \lambda\, f.\, \lambda\, s.\, \lambda\, b.\, b\, f\, s$$
$$\mathbf{fst} \equiv \lambda\, p.\, p\, \mathbf{true}$$
$$\mathbf{snd} \equiv \lambda\, p.\, p\, \mathbf{false}$$

Other Church Encodings

Other Church Encodings of interest:

Church Booleans:

$$\mathbf{true} \equiv \lambda\, t.\, \lambda\, f.\, t$$
$$\mathbf{false} \equiv \lambda\, t.\, \lambda\, f.\, f$$

Church Pairs:

$$\mathbf{pair} \equiv \lambda\, f.\, \lambda\, s.\, \lambda\, b.\, b\, f\, s$$
$$\mathbf{fst} \equiv \lambda\, p.\, p\,\mathbf{true}$$
$$\mathbf{snd} \equiv \lambda\, p.\, p\,\mathbf{false}$$

Church Lists: Too complicated to define here.

Representation of Church Encodings

What do the different Church Encodings *represent*?

Representation of Church Encodings

What do the different Church Encodings *represent*?

Church Numerals: Apply the function f on z a total of n times.

Representation of Church Encodings

What do the different Church Encodings *represent*?

Church Numerals: Apply the function f on z a total of n times.

Church Booleans: Selector functions (one-line `if` statement, etc.).

Representation of Church Encodings

What do the different Church Encodings *represent*?

Church Numerals: Apply the function f on z a total of n times.

Church Booleans: Selector functions (one-line `if` statement, etc.).

Church Pairs: Hard-coded Church Booleans.

Representation of Church Encodings

What do the different Church Encodings *represent*?

Church Numerals: Apply the function f on z a total of n times.

Church Booleans: Selector functions (one-line `if` statement, etc.).

Church Pairs: Hard-coded Church Booleans.

Church Lists: Combine elements using a combining function.

Are Church Encodings practical?

Are Church Encodings practical?

Not really, as you're keeping too many functions in memory . . .

Are Church Encodings practical?

Not really, as you're keeping too many functions in memory . . .

But keep them in mind and try to use functions if you don't need intermediary data.

Are Church Encodings practical?

Not really, as you're keeping too many functions in memory . . .

But keep them in mind and try to use functions if you don't need intermediary data.

OK, I lied: some optimizers, etc. internally use Church Encodings . . . but you should really know what you're doing!

Conclusion