COMP2022: Formal Languages and Logic

2018, Semester 2, Week 3

Joseph Godbehere

16th August, 2018

THE UNIVERSITY OF SYDNEY

Outline

- **Revision - Lambda Calculus**
- Y Combinator

- Encodings
  - numbers (a different way)
  - pairs
  - lists

- Functional Programming

When are α-reductions required?

If they never change the meaning, why bother?

- Readability
- β-reduction assumes all variables have different labels
  - Usually it doesn't matter...
  - ... except when it does!
    - ... even worse – it's not enough to only look at the subformula being reduced

WRONG

$$
\begin{aligned}
& \lambda x.(\lambda y x.y)x \\
&= \lambda x.(\lambda x.x) \\
&= \lambda x.(\lambda y.y) \\
&= \lambda x y.y \\
&= \mathit{FALSE}
\end{aligned}
$$

- Where is the error?

WRONG

$$
\begin{aligned}
& \lambda x.(\lambda yx.y)x \\
&= \lambda x.(\lambda x.x) && \text{(mistake!)} \\
&= \lambda x.(\lambda y.y) \\
&= \lambda xy.y \\
&= \mathit{FALSE}
\end{aligned}
$$

- Where is the error?
- Why is it a mistake?

WRONG

$$
\begin{aligned}
& \lambda x.(\lambda yx.y)x \\
&= \lambda x.(\lambda x.x) && \text{(mistake!)} \\
&= \lambda x.(\lambda y.y) \\
&= \lambda xy.y \\
&= \mathit{FALSE}
\end{aligned}
$$

- Where is the error?
- Why is it a mistake?
- x was bound to the *first* λ, but on line 2 it is not! Free variables in N should not become bound in $M[x := N]$

CORRECT

$$
\begin{aligned}
&\lambda x.(\lambda yx.y)x \\
&= \lambda x.(\lambda yz.y)x && (\alpha) \\
&= \lambda x.(\lambda z.x) && (\beta) \\
&= \lambda xz.x \\
&= \mathit{TRUE}
\end{aligned}
$$

Rule of thumb: always perform α reductions before β reductions.

- sometimes it's necessary
- usually makes the formula easier to read too

η-REDUCTION (ETA)

If x is not free in M, then we can write:

$$\lambda x.\, M x \equiv M$$

Idea: any input applied to this will simply be applied to M

η-reduction (Eta)

If x is not free in M, then we can write:

$$\lambda x.\, M x = M$$

Idea: any input applied to this will simply be applied to M

- If x is not free in M, then $(\lambda x.\, M x) N = M x[x := N] = MN$
  - Identical to applying N to M directly.

η-reduction (Eta)

If x is not free in M, then we can write:

$$\lambda x.Mx = M$$

Idea: any input applied to this will simply be applied to M

- If x is not free in M, then $(\lambda x.Mx)N = Mx[x := N] = MN$
  - Identical to applying N to M directly.

Uses:

- It can simplify some arguments a little
  - e.g. $\lambda x.(\lambda y.y)x = \lambda y.y$
- It can help to convert expressions to 'point free' form (where they do not label their variables).
  - Point-free programs can be easier to reason about, but are often difficult to read.

Outline

- Revision – Lambda Calculus

- **Y Combinator**

- Encodings
  - numbers (a different way)
  - pairs
  - lists

- Functional Programming

Necessary Logical Notation: Quantifiers

- "for all possible values of X ..." denoted $\forall X$
- "there exists a value of X such that ..." denoted $\exists X$

Examples (for all positive rational numbers)

- "$\forall x\ (x * 1 = x)$" is true
- "$\exists x\ (x + 1 = 4)$" is true (choose $x = 3$)
- "$\forall x\ (x + 1 = 4)$" is false (e.g. false on $x = 1$)
- "$\exists x \forall y\ (xy = 0)$" is true (choose $x = 0$)
- "$\exists x \forall y\ (xy = 1)$" is false (whatever we choose for x, we'll be able to find a y that doesn't work)
- "$\forall x \exists y\ (xy = 1)$" is true (for any x, we can choose $y = \frac{1}{x}$)

Combinators

A *combinator* is any expression M which contains no free variables.

Combinators

A *combinator* is any expression M which contains no free variables.

Example:

- $\lambda xy.xyz$ is not a combinator (z is free)

Combinators

A *combinator* is any expression M which contains no free variables.

Example:

- $\lambda xy.xyz$ is not a combinator (z is free)
- $\lambda xy.xyy$ is a combinator (all variables bound)

Combinators

A *combinator* is any expression M which contains no free variables.

Example:

- $\lambda xy.xyz$ is not a combinator (z is free)
- $\lambda xy.xyy$ is a combinator (all variables bound)

Combinators combine values into expressions without relying on quantifiers or explicitly defining variables.

Combinator examples

Standard combinators:

- $I = \lambda x.x$ (identity)
- $K = \lambda xy.x$ (true)
- $K_* = \lambda xy.y$ (false)
- $S = \lambda xyz.xz(yz)$ (generalisation of application)

Combinator examples

Standard combinators:

- $I = \lambda x.x$ (identity)
- $K = \lambda xy.x$ (true)
- $K_* = \lambda xy.y$ (false)
- $S = \lambda xyz.xz(yz)$ (generalisation of application)

We can easily deduce (by using β reduction):

- $IM = M$
- $KMN = M$
- $K_*MN = N$
- $SMNL = ML(NL)$

Combinator examples

Standard combinators:

- $I = \lambda x.x$ (identity)
- $K = \lambda xy.x$ (true)
- $K_* = \lambda xy.y$ (false)
- $S = \lambda xyz.xz(yz)$ (generalisation of application)

We can easily deduce (by using β reduction):

- $IM = M$
- $KMN = M$
- $K_*MN = N$
- $SMNL = ML(NL)$

Interestingly, these λ-free combinators are sufficient to make expressions equal to any λ term. We will not talk about that further today though.

Solving simple equations

$$\exists G\ \forall X\ \ GX = XXX$$

(Where X, F are expressions in the lambda calculus)

Solving simple equations

$$\exists G\ \forall X\ GX = XXX$$

(Where X, F are expressions in the lambda calculus)

"There exists some G such that for all X it's true that $GC = XXX$"

Proof:

Solving simple equations

$$\exists G.\forall X.\ GX = XXX$$

(Where X, F are expressions in the lambda calculus)

"There exists some G such that for all X it's true that $GC = XXX$"

Proof:

- Let $G = \lambda x.xxx$

Solving simple equations

$$\exists G.\forall X.\ GX = XXX$$

(Where X, F are expressions in the lambda calculus)

"There exists some G such that for all X it's true that $GC = XXX$"

Proof:

- Let $G = \lambda x.xxx$
- Then $GX = (\lambda x.xxx)X = XXX$

Solving simple equations

$$\exists G. \forall X. GX = XXX$$

(Where X, F are expressions in the lambda calculus)

"There exists some G such that for all X it's true that $GC = XXX$"

Proof:

- Let $G = \lambda x.xxx$
- Then $GX = (\lambda x.xxx)X = XXX$

That was easy... But what if we need to reason about a recursive function?

Fixed Point Combinators

A *fixed point combinator* is a combinator which has a fixed point.

We say F has a fixed point if $\exists X\ (FX = X)$

i.e. some input X exists which, when applied to F, outputs X again.

Fixed point theorem (i)

Theorem:

$$\forall F\ \exists X\ FX = X$$

Fixed point theorem (i)

Theorem:

$$\forall F\ \exists X\ FX = X$$

"For all F, there exists some X such that $FX = X$"

- i.e. all functions have a fixed point

Fixed point theorem (i)

Theorem:

$$\forall F \; \exists X \; FX = X$$

"For all F, there exists some X such that $FX = X$"

- i.e. all functions have a fixed point

Proof:

Let $W = \lambda x.F(xx)$ and $X = WW$. Then:

$$X =$$

Fixed point theorem (i)

Theorem:

$$\forall F\ \exists X\ FX = X$$

"For all F, there exists some X such that $FX = X$"

- i.e. all functions have a fixed point

Proof:
Let $W = \lambda x.F(xx)$ and $X = WW$. Then:

$$X = WW \quad \text{(def. of X)}$$
$$=$$

Fixed point theorem (i)

Theorem:

$$\forall F \; \exists X \; FX = X$$

"For all F, there exists some X such that $FX = X$"

- i.e. all functions have a fixed point

Proof:
Let $W = \lambda x.F(xx)$ and $X = WW$. Then:

$$\begin{aligned} X &= WW && \text{(def. of X)} \\ &= (\lambda x.F(xx))\,W && \text{(def. of W)} \\ &= \end{aligned}$$

Fixed point theorem (i)

Theorem:

$$\forall F\ \exists X\ FX = X$$

"For all F, there exists some X such that $FX = X$"

- i.e. all functions have a fixed point

Proof:

Let $W = \lambda x.F(xx)$ and $X = WW$. Then:

$$\begin{aligned} X &= WW && \text{(def. of X)} \\ &= (\lambda x.F(xx))W && \text{(def. of W)} \\ &= F(WW) && (\beta\text{-reduction}) \\ &= \end{aligned}$$

Fixed point theorem (i)

Theorem:

$$\forall F \; \exists X \; FX = X$$

"For all F, there exists some X such that $FX = X$"

- i.e. all functions have a fixed point

Proof:

Let $W = \lambda x.F(xx)$ and $X = WW$. Then:

$$\begin{aligned} X &= WW && \text{(def. of X)} \\ &= (\lambda x.F(xx))W && \text{(def. of W)} \\ &= F(WW) && (\beta\text{-reduction}) \\ &= FX && \text{(def. of X)} \end{aligned}$$

Fixed point theorem (II)

There is a fixed point combinator (the "Y Combinator")

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

such that

$$\forall F\ F(YF) = YF$$

Proof.

$$YF =$$

Fixed point theorem (ii)

There is a fixed point combinator (the "Y Combinator")

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

such that

$$\forall F \; F(YF) = YF$$

Proof.

$$YF = \Big(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))\Big)F \qquad \text{(defn. of } Y\text{)}$$

$$=$$

Fixed point theorem (II)

There is a fixed point combinator (the "Y Combinator")

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

such that

$$\forall F \; F(YF) = YF$$

Proof.

$$\begin{aligned} YF &= \Big(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))\Big)F && \text{(defn. of } Y\text{)} \\ &= (\lambda x.F(xx))(\lambda x.F(xx)) && (\beta-reduction) \\ &= \end{aligned}$$

Fixed point theorem (ii)

There is a fixed point combinator (the "Y Combinator")

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

such that

$$\forall F \; F(YF) = YF$$

Proof:

$$\begin{aligned}
YF &= \Big(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))\Big)F && \text{(defn. of } Y\text{)} \\
&= (\lambda x.F(xx))(\lambda x.F(xx)) && (\beta - reduction) \\
&= F\Big((\lambda x.F(xx))(\lambda x.F(xx))\Big) && \text{(by pf. of theorem (i))} \\
&=
\end{aligned}$$

Fixed point theorem (ii)

There is a fixed point combinator (the "Y Combinator")

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

such that

$$\forall F \; F(YF) = YF$$

Proof:

$$\begin{aligned}
YF &= \Big(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))\Big)F && \text{(defn. of } Y\text{)} \\
&= (\lambda x.F(xx))(\lambda x.F(xx)) && (\beta - reduction) \\
&= F\Big((\lambda x.F(xx))(\lambda x.F(xx))\Big) && \text{(by pf. of theorem (i))} \\
&= F(YF) && \text{(by equality above)}
\end{aligned}$$

Y Combinator

So, uh... Why is this interesting?

Y Combinator

So, uh... Why is this interesting?

1. Good news: it allows us to write self recursive functions
   - it effectively lets us define variables

Y Combinator

So, uh... Why is this interesting?

1. Good news: it allows us to write self recursive functions
   - it effectively lets us define variables

2. Bad news: it leads to Curry's Paradox, and the incompleteness of lambda calculus
   - not all valid expressions can be proved / computed

Recursion example

Suppose we want to compute factorials:

$$f(n) = \mathit{if}\ (n == 0)\ \mathit{then}\ 1\ \mathit{else}\ n * f(n-1)$$

Recursion example

Suppose we want to compute factorials:

$$f(n) = if\ (n == 0)\ then\ 1\ else\ n * f(n-1)$$

We'll need some helper functions / encodings:

- Church numerals: $c_n = \lambda f x . f^n(x)$

Recursion example

Suppose we want to compute factorials:

$$f(n) = if\ (n == 0)\ then\ 1\ else\ n * f(n-1)$$

We'll need some helper functions / encodings:

- Church numerals: $c_n = \lambda fx.f^n(x)$
    - This notation, indicating n repetitions of $f(...)$ is a little dangerous (but convenient)
    - Be aware that Church numerals have the form:

$$\lambda fz.f(f(f(f(fz)))) \neq \lambda fz.fffffz$$

Recursion example

Suppose we want to compute factorials:

$$f(n) = if\ \ (n == 0)\ then\ 1\ else\ n * f(n-1)$$

We'll need some helper functions / encodings:

- Church numerals: $c_n = \lambda fx.f^n(x)$
- ISZERO := $\lambda n.n\ (\lambda x.FALSE)\ TRUE$

Recursion example

Suppose we want to compute factorials:

$$f(n) = if\ (n == 0)\ then\ 1\ else\ n * f(n-1)$$

We'll need some helper functions / encodings:

- Church numerals: $c_n = \lambda fx.f^n(x)$
- ISZERO := $\lambda n.n\ (\lambda x.FALSE)\ TRUE$
  - Returns TRUE if the argument is a Church zero, FALSE if it's any other Church numeral

ISZERO ZERO

$$
\begin{aligned}
& ISZERO\ ZERO \\
= & \\
= & \\
= & \\
= & \\
= &
\end{aligned}
$$

ISZERO ZERO

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ZERO} \\
= & (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ZERO} && \text{(def. ISZERO)} \\
= & \\
= & \\
= & \\
= &
\end{aligned}
$$

ISZERO ZERO

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ZERO} \\
= {} & (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ZERO} && \text{(def. ISZERO)} \\
= {} & \mathit{ZERO}\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\beta) \\
= {} & \\
= {} & \\
= {} &
\end{aligned}
$$

ISZERO ZERO

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ZERO} \\
&= (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ZERO} && \text{(def. ISZERO)} \\
&= \mathit{ZERO}\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\beta) \\
&= (\lambda fz.z)\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\mathit{def}.\mathit{ZERO}) \\
&= \\
&=
\end{aligned}
$$

ISZERO ZERO

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ZERO} \\
&= (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ZERO} && \text{(def. ISZERO)} \\
&= \mathit{ZERO}\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\beta) \\
&= (\lambda f z.z)\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\mathit{def}.\mathit{ZERO}) \\
&= (\lambda z.z)\ \mathit{TRUE} && (\beta) \\
&=
\end{aligned}
$$

ISZERO ZERO

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ZERO} \\
&= (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ZERO} && \text{(def. ISZERO)} \\
&= \mathit{ZERO}\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\beta) \\
&= (\lambda fz.z)\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\mathit{def}.\mathit{ZERO}) \\
&= (\lambda z.z)\ \mathit{TRUE} && (\beta) \\
&= \mathit{TRUE} && (\beta)
\end{aligned}
$$

ISZERO ONE

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ONE} \\
= & \\
= & \\
= & \\
= & \\
= & \\
= &
\end{aligned}
$$

ISZERO ONE

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ONE} \\
= {} & (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ONE} && \text{(def. ISZERO)} \\
= {} & \\
= {} & \\
= {} & \\
= {} & \\
= {} &
\end{aligned}
$$

ISZERO ONE

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ONE} \\
= {} & (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ONE} && \text{(def. ISZERO)} \\
= {} & \mathit{ONE}\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\beta) \\
= {} & \\
= {} & \\
= {} & \\
= {} &
\end{aligned}
$$

ISZERO ONE

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ONE} \\
&= (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ONE} && \text{(def. ISZERO)} \\
&= \mathit{ONE}\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\beta) \\
&= (\lambda fz.fz)\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (def.ONE) \\
&= \\
&= \\
&=
\end{aligned}
$$

ISZERO ONE

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ONE} \\
&= (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ONE} && \text{(def. ISZERO)} \\
&= \mathit{ONE}\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\beta) \\
&= (\lambda fz.fz)\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (def.ONE) \\
&= (\lambda z.(\lambda x.\mathit{FALSE})z)\ \mathit{TRUE} && (\beta) \\
&= \\
&=
\end{aligned}
$$

ISZERO ONE

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ONE} \\
&= (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ONE} && \text{(def. ISZERO)} \\
&= \mathit{ONE}\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\beta) \\
&= (\lambda fz.fz)\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (def.\mathit{ONE}) \\
&= (\lambda z.(\lambda x.\mathit{FALSE})z)\ \mathit{TRUE} && (\beta) \\
&= (\lambda x.\mathit{FALSE})\ \mathit{TRUE} && (\beta) \\
&=
\end{aligned}
$$

ISZERO ONE

$$
\begin{aligned}
& \mathit{ISZERO}\ \mathit{ONE} & \\
&= (\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE})\ \mathit{ONE} & \text{(def. ISZERO)} \\
&= \mathit{ONE}\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} & (\beta) \\
&= (\lambda fz.fz)\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE} & (def.ONE) \\
&= (\lambda z.(\lambda x.\mathit{FALSE})z)\ \mathit{TRUE} & (\beta) \\
&= (\lambda x.\mathit{FALSE})\ \mathit{TRUE} & (\beta) \\
&= \mathit{FALSE} & (\beta)
\end{aligned}
$$

Recursion example

Suppose we want to compute factorials:

$$f(n) = if\ (n == 0)\ then\ 1\ else\ n * f(n-1)$$

We'll need some helper functions / encodings:

- Church numerals: $c_n = \lambda fx.f^n(x)$
- ISZERO $:= \lambda n.n\ (\lambda x.FALSE)\ TRUE$
- MULT $:= \lambda xyz.x(yz)$ (seen previously)

Recursion example

Suppose we want to compute factorials:

$$f(n) = \mathit{if}\ (n == 0)\ \mathit{then}\ 1\ \mathit{else}\ n * f(n-1)$$

We'll need some helper functions / encodings:

- Church numerals: $c_n = \lambda fx.f^n(x)$
- ISZERO := $\lambda n.n\ (\lambda x.\mathit{FALSE})\ \mathit{TRUE}$
- MULT := $\lambda xyz.x(yz)$ (seen previously)
- PRED := $\lambda nfx.n(\lambda gh.h(gf))(\lambda y.x)(\lambda u.u)$
  - This gives the predecessor of a number
  - PRED 1 = 0, PRED 2 = 1, ..., PRED n = (n-1)
  - The derivation of this is *much* longer than for the operations which increase numbers

Recursion example

Suppose we want to compute factorials:

$$f(n) = \mathit{if}\ (n == 0)\ \mathit{then}\ 1\ \mathit{else}\ n * f(n-1)$$

We'll need some helper functions / encodings:

- Church numerals: $c_n = \lambda fx.f^n(x)$
- ISZERO $:= \lambda n.n\ (\lambda x.FALSE)\ TRUE$
- MULT $:= \lambda xyz.x(yz)$ (seen previously)
- PRED $:= \lambda nfx.n(\lambda gh.h(gf))(\lambda y.x)(\lambda u.u)$
  - This gives the predecessor of a number
  - PRED 1 = 0, PRED 2 = 1, ..., PRED n = (n-1)
  - The derivation of this is *much* longer than for the operations which increase numbers
    - Subtraction and division are also difficult!

PRED TWO... is a monster

$$
\begin{aligned}
PRED\ TWO &= \lambda nfx.n(\lambda gh.h(gf))(\lambda y.x)(\lambda u.u)\ TWO \\
&= \lambda fx.TWO\ (\lambda gh.h(gf))(\lambda y.x)(\lambda u.u) \\
&= \lambda fx.(\lambda ab.a(ab))(\lambda gh.h(gf))(\lambda y.x)(\lambda u.u) \\
&= \lambda fx.\Big(\lambda b.(\lambda gh.h(gf))((\lambda gh.h(gf))b)\Big)(\lambda y.x)(\lambda u.u) \\
&= \lambda fx.\big(\lambda gh.h(gf)\big)\Big((\lambda gh.h(gf))(\lambda y.x)\Big)(\lambda u.u) \\
&= \lambda fx.\big(\lambda gh.h(gf)\big)\Big((\lambda h.h((\lambda y.x)f))\Big)(\lambda u.u) \\
&= \lambda fx.\big(\lambda gh.h(gf)\big)(\lambda h.hx)(\lambda u.u) \\
&= \lambda fx.\big(\lambda gh.h(gf)\big)(\lambda i.ix)(\lambda u.u) \\
&= \lambda fx.\big(\lambda h.h((\lambda i.ix)f)\big)(\lambda u.u) \\
&= \lambda fx.(\lambda h.h(fx))(\lambda u.u) \\
&= \lambda fx.(\lambda u.u)(fx)) = \lambda fx.fx = ONE
\end{aligned}
$$

Recursion example

Suppose we want to compute factorials:

$$f(n) = if\ (n == 0)\ then\ 1\ else\ n * f(n-1)$$

We'll need some helper functions / encodings:

- Church numerals: $c_n = \lambda fx.f^n(x)$
- ISZERO := $\lambda n.n\ (\lambda x.FALSE)\ TRUE$
- MULT := $\lambda xyz.x(yz)$
- PRED := $\lambda nfx.n(\lambda gh.h(gf))(\lambda y.x)(\lambda u.u)$

Recursion example

We want to write something like:
"FACT n = (ISZERO n) 1 (MULT n (FACT (PRED n)))"

We can't directly define functions self referentially, so we use the Y Combinator:

Recursion example

We want to write something like:
"FACT n = (ISZERO n) 1 (MULT n (FACT (PRED n)))"

We can't directly define functions self referentially, so we use the Y Combinator:

- $H = \lambda fn.(ISZERO\ n)\ 1\ \big(MULT\ n\ (f\ (PRED\ n))\big)$
  - H takes a function and a number. If the number is zero, it returns 1, otherwise it returns the product of n and f(n-1).

Recursion example

We want to write something like:
"FACT n = (ISZERO n) 1 (MULT n (FACT (PRED n)))"

We can't directly define functions self referentially, so we use the Y Combinator:

- $H = \lambda fn.(ISZERO\ n)\ 1\ \big(MULT\ n\ (f\ (PRED\ n))\big)$
  - H takes a function and a number. If the number is zero, it returns 1, otherwise it returns the product of n and f(n-1)

- $FACTORIAL = Y\ \ H$
  - Because $YH = H(YH)$, the Y Combinator helps us to apply the H function to itself

Factorial 5 (overview)

- $H = \lambda fn.(ISZERO\ n)\ 1\ \big(MULT\ n\ (f\ (PRED\ n))\big)$
- $FACTORIAL = Y\ H$

H takes a function f and a number n. It returns 1 if the number is 0, otherwise the product of n and $f(n-1)$

$$FACTORIAL\ 5$$

Factorial 5 (overview)

- $H = \lambda fn.(ISZERO\ n)\ 1\ \big(MULT\ n\ (f\ (PRED\ n))\big)$
- $FACTORIAL = Y\ H$

H takes a function f and a number n. It returns 1 if the number is 0, otherwise the product of n and $f(n-1)$

$$FACTORIAL\ 5 = (Y\ H)\ 5$$

Factorial 5(overview)

- $H = \lambda fn.(ISZERO\ n)\ 1\ \big(MULT\ n\ (f\ (PRED\ n))\big)$
- $FACTORIAL = Y\ H$

H takes a function f and a number n. It returns 1 if the number is 0, otherwise the product of n and $f(n-1)$

$$\begin{aligned} FACTORIAL\ 5 &= (Y\ H)\ 5 \\ &= H\ (Y\ H)\ 5 \qquad \text{(Y Combinator!)} \end{aligned}$$

Factorial 5(overview)

- $H = \lambda fn.(ISZERO\ n)\ 1\ \big(MULT\ n\ (f\ (PRED\ n))\big)$
- $FACTORIAL = Y\ H$

H takes a function f and a number n. It returns 1 if the number is 0, otherwise the product of n and $f(n-1)$

$$\begin{aligned} FACTORIAL\ 5 &= (Y\ \ H)\ 5 \\ &= H\ (Y\ \ H)\ 5 && \text{(Y Combinator!)} \\ &= 5 \times ((Y\ H)\ 4) && (\text{because } 5 \neq 0) \end{aligned}$$

Factorial 5(overview)

- $H = \lambda fn.(ISZERO\ n)\ 1\ \big(MULT\ n\ (f\ (PRED\ n))\big)$
- $FACTORIAL = Y\ H$

H takes a function f and a number n. It returns 1 if the number is 0, otherwise the product of n and $f(n-1)$

$$
\begin{aligned}
FACTORIAL\ 5 &= (Y\ \ H)\ 5 \\
&= H\ (Y\ \ H)\ 5 && \text{(Y Combinator!)} \\
&= 5 * ((Y\ \ H)\ 4) && \text{(because } 5 \neq 0) \\
&= \ldots \\
&= 120 * ((Y\ \ H)\ 0)
\end{aligned}
$$

Factorial 5(overview)

- $H = \lambda fn.(ISZERO\ n)\ 1\ \big(MULT\ n\ (f\ (PRED\ n))\big)$
- $FACTORIAL = Y\ H$

H takes a function f and a number n. It returns 1 if the number is 0, otherwise the product of n and $f(n-1)$

$$
\begin{aligned}
FACTORIAL\ 5 &= (Y\ \ H)\ 5 \\
&= H\ (Y\ \ H)\ 5 && \text{(Y Combinator!)} \\
&= 5 * ((Y\ \ H)\ 4) && (\text{because } 5 \neq 0) \\
&= \ldots \\
&= 120 * ((Y\ \ H)\ 0) \\
&= 120 * 1 = 120
\end{aligned}
$$

Factorial 3 (detailed 1)

$$\begin{aligned} & \textit{FACTORIAL}\ 3 \\ = & \\ = & \\ = & \\ = & \\ = & \\ = & \\ = & \\ = & \end{aligned}$$

Factorial 3 (detailed 1)

$$
\begin{aligned}
& \mathit{FACTORIAL}\ 3 \\
= \ & Y\ H\ 3 \\
= \ & \\
= \ & \\
= \ & \\
= \ & \\
= \ & \\
= \ & \\
= \ &
\end{aligned}
$$

Factorial 3 (detailed 1)

$$
\begin{aligned}
& FACTORIAL\ 3 \\
= {} & Y\ H\ 3 \\
= {} & H\ (Y\ \ H)\ 3 \qquad \text{(Y Combinator)}
\end{aligned}
$$

Factorial 3 (detailed 1)

$$
\begin{aligned}
& FACTORIAL\ 3 \\
=\ & Y\ H\ 3 \\
=\ & H\ (Y\ H)\ 3 && \text{(Y Combinator)} \\
=\ & \Big(\lambda fn.(ISZERO\ n) 1 \Big(MULT\ n (f(PRED\ n))\Big)\Big)(YH)3 && \text{(H)} \\
=\ & \\
=\ & \\
=\ & \\
=\ & \\
=\ &
\end{aligned}
$$

Factorial 3 (detailed 1)

$$
\begin{aligned}
& \mathit{FACTORIAL}\ 3 \\
=\ & Y\ H\ 3 \\
=\ & H\ (Y\ H)\ 3 && \text{(Y Combinator)} \\
=\ & \Big(\lambda f n.(\mathit{ISZERO}\ n)\, 1 \Big(\mathit{MULT}\ n\big(f(\mathit{PRED}\ n)\big)\Big)\Big)(YH)3 && \text{(H)} \\
=\ & \Big(\lambda n.(\mathit{ISZERO}\ n)\ 1\ \Big(\mathit{MULT}\ n\big(YH(\mathit{PRED}\ n)\big)\Big)\Big)\ 3 && (\beta) \\
=\ & \\
=\ & \\
=\ & \\
=\ &
\end{aligned}
$$

Factorial 3 (detailed 1)

$$
\begin{aligned}
& \mathit{FACTORIAL}\ 3 \\
&= Y\ H\ 3 \\
&= H\ (Y\ H)\ 3 && \text{(Y Combinator)} \\
&= \Big(\lambda f n.(\mathit{ISZERO}\ n) 1 \Big(\mathit{MULT}\ n (f (\mathit{PRED}\ n))\Big)\Big)(YH)3 && \text{(H)} \\
&= \Big(\lambda n.(\mathit{ISZERO}\ n)\ 1\ \Big(\mathit{MULT}\ n\big(YH(\mathit{PRED}\ n)\big)\Big)\Big)\ 3 && (\beta) \\
&= (\mathit{ISZERO}\ 3)\ 1\ \Big(\mathit{MULT}\ 3\ (Y\ H\ (\mathit{PRED}\ 3))\Big) && (\beta) \\
&= \\
&= \\
&=
\end{aligned}
$$

Factorial 3 (detailed 1)

$$
\begin{aligned}
& \mathit{FACTORIAL}\ 3 \\
&= Y\ H\ 3 \\
&= H\ (Y\ H)\ 3 && \text{(Y Combinator)} \\
&= \Big(\lambda f n.(\mathit{ISZERO}\ n) 1 \Big(\mathit{MULT}\ n (f (\mathit{PRED}\ n))\Big)\Big)(YH)3 && \text{(H)} \\
&= \Big(\lambda n.(\mathit{ISZERO}\ n)\ 1\ \Big(\mathit{MULT}\ n\big(YH(\mathit{PRED}\ n)\big)\Big)\Big)\ 3 && (\beta) \\
&= (\mathit{ISZERO}\ 3)\ 1\ \Big(\mathit{MULT}\ 3\ \big(Y\ H\ (\mathit{PRED}\ 3)\big)\Big) && (\beta) \\
&= \ldots = \mathit{FALSE}\ 1\ \Big(\mathit{MULT}\ 3\ \big(Y\ H\ (\mathit{PRED}\ 3)\big)\Big) && (3 \neq 0) \\
&= \\
&=
\end{aligned}
$$

FACTORIAL 3 (DETAILED 1)

$$
\begin{aligned}
& \mathit{FACTORIAL}\ 3 \\
&= Y\ H\ 3 \\
&= H\ (Y\ H)\ 3 && \text{(Y Combinator)} \\
&= \Big(\lambda f n.(\mathit{ISZERO}\ n) 1 \Big(\mathit{MULT}\ n (f(\mathit{PRED}\ n))\Big)\Big)(YH)3 && \text{(H)} \\
&= \Big(\lambda n.(\mathit{ISZERO}\ n)\ 1\ \Big(\mathit{MULT}\ n \big(YH(\mathit{PRED}\ n)\big)\Big)\Big)\ 3 && (\beta) \\
&= (\mathit{ISZERO}\ 3)\ 1\ \Big(\mathit{MULT}\ 3\ (Y\ H\ (\mathit{PRED}\ 3))\Big) && (\beta) \\
&= \ldots = \mathit{FALSE}\ 1\ \Big(\mathit{MULT}\ 3\ (Y\ H\ (\mathit{PRED}\ 3))\Big) && (3 \neq 0) \\
&= \ldots = \mathit{MULT}\ 3\ \big(Y\ H\ (\mathit{PRED}\ 3)\big) && (\text{def. } \mathit{FALSE}) \\
&=
\end{aligned}
$$

Factorial 3 (detailed 1)

$$
\begin{aligned}
& \mathit{FACTORIAL}\ 3 \\
&= Y\ H\ 3 \\
&= H\ (Y\ \ H)\ 3 && \text{(Y Combinator)} \\
&= \Big(\lambda f n.(\mathit{ISZERO}\ n)\,1\Big(\mathit{MULT}\ n\big(f\big(\mathit{PRED}\ n\big)\big)\Big)\Big)(YH)3 && \text{(H)} \\
&= \Big(\lambda n.(\mathit{ISZERO}\ \ n)\ 1\ \Big(\mathit{MULT}\ \ n\big(YH\big(\mathit{PRED}\ \ n\big)\big)\Big)\Big)\ 3 && (\beta) \\
&= (\mathit{ISZERO}\ 3)\ 1\ \Big(\mathit{MULT}\ 3\ \big(Y\ H\ (\mathit{PRED}\ 3)\big)\Big) && (\beta) \\
&= \ldots = \mathit{FALSE}\ 1\ \Big(\mathit{MULT}\ 3\ \big(Y\ \ H\ \ (\mathit{PRED}\ 3)\big)\Big) && (3 \neq 0) \\
&= \ldots = \mathit{MULT}\ 3\ \big(Y\ \ H\ \ (\mathit{PRED}\ 3)\big) && \text{(def. } \mathit{FALSE}) \\
&= \ldots = \mathit{MULT}\ 3\ (Y\ \ H\ \ 2) && (\mathit{PRED}\ 3 = 2)
\end{aligned}
$$

Factorial 3 (detailed 2)

$$
\begin{aligned}
& FACTORIAL\ 3 \\
&= \ldots = MULT\ 3\ (Y\ \ H\ \ 2) \\
&= \\
&= \\
&= \\
&= \\
&= \\
&= \\
&=
\end{aligned}
$$

Factorial 3 (detailed 2)

$$FACTORIAL\ 3$$
$$= \ldots = MULT\ 3\ (Y\ \ H\ \ 2)$$
$$= \ldots = MULT\ 3\ (MULT\ 2\ (Y\ \ H\ \ 1))$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$

Factorial 3 (detailed 2)

$$
\begin{aligned}
& FACTORIAL\ 3 \\
= & \ldots = MULT\ 3\ (Y\ \ H\ \ 2) \\
= & \ldots = MULT\ 3\ (MULT\ 2\ (Y\ \ H\ \ 1)) \\
= & \ldots = MULT\ 3\ (MULT\ 2\ (MULT\ 1\ (Y\ \ H\ \ 0))) \\
= & \\
= & \\
= & \\
= & \\
= &
\end{aligned}
$$

Factorial 3 (detailed 2)

$$
\begin{aligned}
& FACTORIAL\ 3 \\
&= \ldots = MULT\ 3\ (Y\ \ H\ \ 2) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (Y\ \ H\ \ 1)) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (MULT\ 1\ (Y\ \ H\ \ 0))) \\
&= \ldots = \ldots (ISZERO\ 0)\ 1 \ldots \\
&= \\
&= \\
&= \\
&=
\end{aligned}
$$

Factorial 3 (detailed 2)

$$
\begin{aligned}
& FACTORIAL\ 3 \\
&= \ldots = MULT\ 3\ (Y\ \ H\ \ 2) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (Y\ \ H\ \ 1)) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (MULT\ 1\ (Y\ \ H\ \ 0))) \\
&= \ldots = \ldots(ISZERO\ 0)\ 1\ldots \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (MULT\ 1\ 1)) \\
&= \\
&= \\
&=
\end{aligned}
$$

Factorial 3 (detailed 2)

$$
\begin{aligned}
& FACTORIAL\ 3 \\
&= \ldots = MULT\ 3\ (Y\ \ H\ \ 2) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (Y\ \ H\ \ 1)) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (MULT\ 1\ (Y\ \ H\ \ 0))) \\
&= \ldots = \ldots(ISZERO\ 0)\ 1\ldots \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (MULT\ 1\ 1)) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ 1) \\
&= \\
&=
\end{aligned}
$$

Factorial 3 (detailed 2)

$$
\begin{aligned}
& FACTORIAL\ 3 \\
&= \ldots = MULT\ 3\ (Y\ \ H\ \ 2) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (Y\ \ H\ \ 1)) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (MULT\ 1\ (Y\ \ H\ \ 0))) \\
&= \ldots = \ldots(ISZERO\ 0)\ 1\ldots \\
&= \ldots = MULT\ 3\ (MULT\ 2\ (MULT\ 1\ 1)) \\
&= \ldots = MULT\ 3\ (MULT\ 2\ 1) \\
&= \ldots = MULT\ 3\ 2 \\
&=
\end{aligned}
$$

Factorial 3 (detailed 2)

$$
\begin{aligned}
& \mathit{FACTORIAL}\ 3 \\
& = \ldots = \mathit{MULT}\ 3\ (Y\ \ H\ 2) \\
& = \ldots = \mathit{MULT}\ 3\ (\mathit{MULT}\ 2\ (Y\ \ H\ 1)) \\
& = \ldots = \mathit{MULT}\ 3\ (\mathit{MULT}\ 2\ (\mathit{MULT}\ 1\ (Y\ \ H\ 0))) \\
& = \ldots = \ldots(\mathit{ISZERO}\ 0)\ 1\ldots \\
& = \ldots = \mathit{MULT}\ 3\ (\mathit{MULT}\ 2\ (\mathit{MULT}\ 1\ 1)) \\
& = \ldots = \mathit{MULT}\ 3\ (\mathit{MULT}\ 2\ 1) \\
& = \ldots = \mathit{MULT}\ 3\ 2 \\
& = \ldots = 6
\end{aligned}
$$

Y Combinator reminder

This worked because the Y Combinator

$$Y = \lambda f.\big(\lambda x.f\,(xx)\big)\big(\lambda x.f\,(xx)\big)$$

Has the property that $F(YF) = YF$ for all F.

Important:

- When performing the reductions, use that property
- *Don't* β-reduce the Y Combinator directly.

Outline

- Revision – Lambda Calculus
- Y Combinator

- **Encodings**
  - numbers (a different way)
  - pairs
  - lists

- Functional Programming

Booleans

Reminder: our definition of Church Booleans lets us write:

$$if\ B\ then\ P\ else\ Q$$

simply as:

$$B\ P\ Q$$

where B is some boolean. i.e. anything which reduces to

- $TRUE = \lambda xy.x$, or
- $FALSE = \lambda xy.y$

Pairs (Barendregt style)

Let P, Q be expressions in the lambda calculus.

If we write:

$$[M, N] = \lambda z.(if\ \ z\ \ then\ \ M\ \ else\ \ N)$$
$$= \lambda z.zMN$$

Then:

- $[M, N]\ \mathit{TRUE} = M$
- $[M, N]\ \mathit{FALSE} = N$

We can use $[M, N]$ to denote an ordered pair.

Pairs (Barendregt)

$$
\begin{aligned}
[M, N]\ \mathit{TRUE} &= (\lambda z.z\ M\ N)\ \mathit{TRUE} \\
&= \mathit{TRUE}\ M\ N \\
&= (\lambda xy.x)\ M\ N \\
&= (\lambda y.M)\ N \\
&= M
\end{aligned}
$$

Pairs (Barendregt)

$$\begin{aligned}[M, N]\ \mathit{TRUE} &= (\lambda z.z\ M\ N)\ \mathit{TRUE} \\ &= \mathit{TRUE}\ M\ N \\ &= (\lambda xy.x)\ M\ N \\ &= (\lambda y.M)\ N \\ &= M\end{aligned}$$

$$\begin{aligned}[M, N]\ \mathit{FALSE} &= (\lambda z.z\ M\ N)\ \mathit{FALSE} \\ &= \mathit{FALSE}\ M\ N \\ &= (\lambda xy.y)\ M\ N \\ &= (\lambda y.y)\ N \\ &= N\end{aligned}$$

Numbers (Barendregt style)

Recall that predecessor, subtraction, division were difficult in the Church encoding.

Numbers (Barendregt style)

Recall that predecessor, subtraction, division were difficult in the Church encoding.

We can use pairs to make another encoding for numbers:

- $0 = I = \lambda x.x$
- $n + 1 = [FALSE, n]$

Numbers (Barendregt style)

Recall that predecessor, subtraction, division were difficult in the Church encoding.

We can use pairs to make another encoding for numbers:

- $0 = I = \lambda x.x$
- $n + 1 = [FALSE, n]$

For example:

- $1 = [FALSE, 0] = [FALSE, I] = \lambda z.z(\lambda xy.y)(\lambda x.x)$

Numbers (Barendregt style)

Recall that predecessor, subtraction, division were difficult in the Church encoding.

We can use pairs to make another encoding for numbers:

- $0 = I = \lambda x.x$
- $n + 1 = [FALSE, n]$

For example:

- $1 = [FALSE, 0] = [FALSE, I] = \lambda z.z(\lambda xy.y)(\lambda x.x)$
- $2 = [FALSE, 1] = [FALSE, [FALSE, I]]$

Numbers (Barendregt style)

Recall that predecessor, subtraction, division were difficult in the Church encoding.

We can use pairs to make another encoding for numbers:

- $0 = I = \lambda x.x$
- $n + 1 = [FALSE, n]$

For example:

- $1 = [FALSE, 0] = [FALSE, I] = \lambda z.z(\lambda xy.y)(\lambda x.x)$
- $2 = [FALSE, 1] = [FALSE, [FALSE, I]]$
- $3 = [FALSE, 2] = [FALSE, [FALSE, [FALSE, I]]]$

Numbers (Barendregt style)

Some of the operators are a *lot* simpler:

- $SUCC = \lambda x.[FALSE, x]$ (the next number)
  - This simply puts another FALSE in front.
  - $SUCC\ ONE = (\lambda x.[FALSE, x])ONE = [FALSE, ONE] = [FALSE, [FALSE, I]]$
- $PRED = \lambda x.x\ FALSE$ (the previous number)
  - $PRED\ ONE = (\lambda x.x\ FALSE)ONE = ONEFALSE = [FALSE, I]FALSE = I$
- $ISZERO = \lambda x.x\ TRUE$

Numbers (Barendregt style)

Some of the operators are a *lot* simpler:

- $SUCC = \lambda x.[FALSE, x]$ (the next number)
  - This simply puts another FALSE in front.
  - $SUCC\ ONE = (\lambda x.[FALSE, x])ONE = [FALSE, ONE] = [FALSE, [FALSE, I]]$
- $PRED = \lambda x.x\ FALSE$ (the previous number)
  - $PRED\ ONE = (\lambda x.x\ FALSE)ONE = ONEFALSE = [FALSE, I]FALSE = I$
- $ISZERO = \lambda x.x\ TRUE$

... recall that PRED for the Church numerals was

$$\lambda nfx.n(\lambda gh.h(gf))(\lambda y.x)(\lambda u.u)$$

Numbers (Barendregt style)

Addition is more complex, but quite intuitive

- base case: $ADD(0, y) = y$
- recursive case: $ADD(x, y) = 1 + ADD(x - 1, y)$

Numbers (Barendregt style)

Addition is more complex, but quite intuitive:

- base case: $ADD(0, y) = y$
- recursive case: $ADD(x, y) = 1 + ADD(x - 1, y)$

To implement the recursion we can use the Y Combinator again:

$$ADD = Y\Big(\lambda fxy.(ISZERO\ x)\ y\ (SUCC(f\,(PRED\ x)\ y))\Big)$$

- i.e. Y "if x is 0 then y else $(1 + f(x - 1,\ y))$"

Generalised recursion

We can generalise this idea of recursion to support an arbitrary number of variables, base and recursive cases.

See section 3.11 in the reference text (Barendregt) if you're interested in the fine details of this.

Pairs (Church)

Similar idea, but not identical to the encoding Barendregt uses.

- $PAIR = \lambda xyz.zxy$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$

e.g.

$$FIRST\ (PAIR\ a\ b) =$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$

Pairs (Church)

Similar idea, but not identical to the encoding Barendregt uses.

- $PAIR = \lambda xyz.zxy$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$

e.g.

$$
\begin{aligned}
FIRST\ (PAIR\ a\ b) &= (\lambda p.p\ TRUE)\ (PAIR\ a\ b) \\
&= \\
&= \\
&= \\
&= \\
&= \\
&= \\
&= \\
&=
\end{aligned}
$$

Pairs (Church)

Similar idea, but not identical to the encoding Barendregt uses.

- $PAIR = \lambda xyz.zxy$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$

e.g.

$$\begin{aligned} FIRST\ (PAIR\ a\ b) &= (\lambda p.p\ TRUE)\ (PAIR\ a\ b) \\ &= PAIR\ a\ b\ TRUE \\ &= \\ &= \\ &= \\ &= \\ &= \\ &= \\ &= \end{aligned}$$

Pairs (Church)

Similar idea, but not identical to the encoding Barendregt uses.

- $PAIR = \lambda xyz.zxy$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$

e.g.

$$
\begin{aligned}
FIRST\ (PAIR\ a\ b) &= (\lambda p.p\ TRUE)\ (PAIR\ a\ b)\\
&= PAIR\ a\ b\ TRUE\\
&= (\lambda xyz.zxy)\ a\ b\ TRUE\\
&=\\
&=\\
&=\\
&=\\
&=\\
&=
\end{aligned}
$$

Pairs (Church)

Similar idea, but not identical to the encoding Barendregt uses.

- $PAIR = \lambda xyz.zxy$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$

e.g.

$$
\begin{aligned}
FIRST\ (PAIR\ a\ b) &= (\lambda p.p\ TRUE)\ (PAIR\ a\ b) \\
&= PAIR\ a\ b\ TRUE \\
&= (\lambda xyz.zxy)\ a\ b\ TRUE \\
&= (\lambda yz.zay)\ b\ TRUE \\
&= \\
&= \\
&= \\
&= \\
&=
\end{aligned}
$$

Pairs (Church)

Similar idea, but not identical to the encoding Barendregt uses.

- $PAIR = \lambda xyz.zxy$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$

e.g.

$$
\begin{aligned}
FIRST\ (PAIR\ a\ b) &= (\lambda p.p\ TRUE)\ (PAIR\ a\ b)\\
&= PAIR\ a\ b\ TRUE\\
&= (\lambda xyz.zxy)\ a\ b\ TRUE\\
&= (\lambda yz.zay)\ b\ TRUE\\
&= (\lambda z.zab)\ TRUE\\
&=\\
&=\\
&=\\
&=
\end{aligned}
$$

Pairs (Church)

Similar idea, but not identical to the encoding Barendregt uses.

- $PAIR = \lambda xyz.zxy$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$

e.g.

$$
\begin{aligned}
FIRST\ (PAIR\ a\ b) &= (\lambda p.p\ TRUE)\ (PAIR\ a\ b)\\
&= PAIR\ a\ b\ TRUE\\
&= (\lambda xyz.zxy)\ a\ b\ TRUE\\
&= (\lambda yz.zay)\ b\ TRUE\\
&= (\lambda z.zab)\ TRUE\\
&= TRUE\ a\ b\\
&=\\
&=\\
&=
\end{aligned}
$$

Pairs (Church)

Similar idea, but not identical to the encoding Barendregt uses.

- $PAIR = \lambda xyz.zxy$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$

e.g.

$$
\begin{aligned}
FIRST\ (PAIR\ a\ b) &= (\lambda p.p\ TRUE)\ (PAIR\ a\ b) \\
&= PAIR\ a\ b\ TRUE \\
&= (\lambda xyz.zxy)\ a\ b\ TRUE \\
&= (\lambda yz.zay)\ b\ TRUE \\
&= (\lambda z.zab)\ TRUE \\
&= TRUE\ a\ b \\
&= (\lambda xy.x)ab \\
&= \\
&=
\end{aligned}
$$

Pairs (Church)

Similar idea, but not identical to the encoding Barendregt uses.

- $PAIR = \lambda xyz.zxy$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$

e.g.

$$
\begin{aligned}
FIRST\ (PAIR\ a\ b) &= (\lambda p.p\ TRUE)\ (PAIR\ a\ b)\\
&= PAIR\ a\ b\ TRUE\\
&= (\lambda xyz.zxy)\ a\ b\ TRUE\\
&= (\lambda yz.zay)\ b\ TRUE\\
&= (\lambda z.zab)\ TRUE\\
&= TRUE\ a\ b\\
&= (\lambda xy.x)ab\\
&= (\lambda y.a)b\\
&= a
\end{aligned}
$$

List (Church)

Idea: lists are pairs of (head, tail)

- head is the *first* list entry
- tail is *everything else* in the list.

I will denote lists as $\{a, b, c, d, ...\}$

List (Church)

We need a way to signal if the list is empty.

List (Church)

We need a way to signal if the list is empty.

Idea: each list entry is a nested pair (isempty, (head, tail))

List (Church)

We need a way to signal if the list is empty.

Idea: each list entry is a nested pair (isempty, (head, tail))

- Empty list = $NIL = PAIR\ TRUE\ TRUE$
- Non-empty list = $PAIR\ FALSE\ (PAIR\ head\ tail)$
- i.e. each list entry has a boolean acting as a sentinel, signaling if this sublist is empty

List (Church)

We need a way to signal if the list is empty.

Idea: each list entry is a nested pair (isempty, (head, tail))

- Empty list = $NIL = PAIR\ TRUE\ TRUE$
- Non-empty list = $PAIR\ FALSE\ (PAIR\ head\ tail)$
- i.e. each list entry has a boolean acting as a sentinel, signaling if this sublist is empty

A list containing $\{a, b, c, d\}$ would look like:

$$
\begin{aligned}
&(PAIR\ FALSE\ (PAIR\ a \\
&\quad (PAIR\ FALSE\ (PAIR\ b \\
&\quad\quad (PAIR\ FALSE\ (PAIR\ c \\
&\quad\quad\quad (PAIR\ FALSE\ (PAIR\ d\ NIL))))))))
\end{aligned}
$$

List (Church)

To make our lists useful, we want the following functions:

- NIL is an empty list
- $ISNIL$ checks if the list is empty
- $HEAD$ gets the first element
- $TAIL$ gets the rest
- $CONS$ prepends a given value to the head of a given list

List (Church)

Encoding:

- $NIL =$
- $ISNIL =$
- $HEAD =$
- $TAIL =$
- $CONS =$

List (Church)

Encoding:

- $NIL = PAIR\ TRUE\ TRUE$ (an empty list)
- $ISNIL =$
- $HEAD =$
- $TAIL =$
- $CONS =$

List (Church)

Encoding:

- $NIL = PAIR\ TRUE\ TRUE$ (an empty list)
- $ISNIL = FIRST$ (is the list empty)
- $HEAD =$
- $TAIL =$
- $CONS =$

List (Church)

Encoding:

- $NIL = PAIR\ TRUE\ TRUE$ (an empty list)
- $ISNIL = FIRST$ (is the list empty)
- $HEAD = \lambda z.FIRST\ (SECONDz)$
- $TAIL =$
- $CONS =$

List (Church)

Encoding:

- $NIL = PAIR\ TRUE\ TRUE$ (an empty list)
- $ISNIL = FIRST$ (is the list empty)
- $HEAD = \lambda z.FIRST\ (SECONDz)$
- $TAIL = \lambda z.SECOND\ (SECONDz)$
- $CONS =$

List (Church)

Encoding:

- $NIL = PAIR\ TRUE\ TRUE$ (an empty list)
- $ISNIL = FIRST$ (is the list empty)
- $HEAD = \lambda z.FIRST\ (SECOND z)$
- $TAIL = \lambda z.SECOND\ (SECOND z)$
- $CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$

List (Church)

Example:

$$ISNIL\ NIL$$

$$=$$

$$=$$

$$=$$

$$=$$

$$=$$

$$=$$

$$=$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \mathit{NIL} \\
= \ & \mathit{FIRST}\ \mathit{NIL} \\
= \ & \\
= \ & \\
= \ & \\
= \ & \\
= \ & \\
= \ &
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \mathit{NIL} \\
=\ & \mathit{FIRST}\ \ \mathit{NIL} \\
=\ & (\lambda p.p\ \ \mathit{TRUE})\ \mathit{NIL} \\
=\ & \\
=\ & \\
=\ & \\
=\ & \\
=\ &
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& ISNIL\ NIL \\
= {} & FIRST\ \ NIL \\
= {} & (\lambda p.p\ \ TRUE)\ NIL \\
= {} & NIL\ TRUE \\
= {} & \\
= {} & \\
= {} & \\
= {} &
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \mathit{NIL} \\
&= \mathit{FIRST}\ \mathit{NIL} \\
&= (\lambda p.p\ \mathit{TRUE})\ \mathit{NIL} \\
&= \mathit{NIL}\ \mathit{TRUE} \\
&= \mathit{PAIR}\ \mathit{TRUE}\ \mathit{TRUE}\ \mathit{TRUE} \\
&= \\
&= \\
&=
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \mathit{NIL} \\
=\ & \mathit{FIRST}\ \mathit{NIL} \\
=\ & (\lambda p.p\ \mathit{TRUE})\ \mathit{NIL} \\
=\ & \mathit{NIL}\ \mathit{TRUE} \\
=\ & \mathit{PAIR}\ \mathit{TRUE}\ \mathit{TRUE}\ \mathit{TRUE} \\
=\ & (\lambda xyz.\ zxy)\ \mathit{TRUE}\ \mathit{TRUE}\ \mathit{TRUE} \\
=\ & \\
=\ &
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \mathit{NIL} \\
= & \ \mathit{FIRST}\ \mathit{NIL} \\
= & \ (\lambda p.p\ \mathit{TRUE})\ \mathit{NIL} \\
= & \ \mathit{NIL}\ \mathit{TRUE} \\
= & \ \mathit{PAIR}\ \mathit{TRUE}\ \mathit{TRUE}\ \mathit{TRUE} \\
= & \ (\lambda xyz.\, zxy)\ \mathit{TRUE}\ \mathit{TRUE}\ \mathit{TRUE} \\
= & \ \ldots = \mathit{TRUE}\ \mathit{TRUE}\ \mathit{TRUE} \\
=
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \mathit{NIL} \\
&= \mathit{FIRST}\ \mathit{NIL} \\
&= (\lambda p.p\ \mathit{TRUE})\ \mathit{NIL} \\
&= \mathit{NIL}\ \mathit{TRUE} \\
&= \mathit{PAIR}\ \mathit{TRUE}\ \mathit{TRUE}\ \mathit{TRUE} \\
&= (\lambda xyz.\ zxy)\ \mathit{TRUE}\ \mathit{TRUE}\ \mathit{TRUE} \\
&= \ldots = \mathit{TRUE}\ \mathit{TRUE}\ \mathit{TRUE} \\
&= \ldots = \mathit{TRUE}
\end{aligned}
$$

List (Church)

Example:

$$\mathit{ISNIL}\ \{a, b, c, d\}$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \{a, b, c, d\} \\
= {} & \mathit{FIRST}\ \{a, b, c, d\} \\
= {} & \\
= {} & \\
= {} & \\
= {} & \\
= {} & \\
= {} &
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \{a, b, c, d\} \\
=\ & \mathit{FIRST}\ \{a, b, c, d\} \\
=\ & (\lambda p.p\ \mathit{TRUE})\ \{a, b, c, d\} \\
=\ & \\
=\ & \\
=\ & \\
=\ & \\
=\ &
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& ISNIL\ \{a, b, c, d\} \\
= & \ FIRST\ \{a, b, c, d\} \\
= & \ (\lambda p.\, p\ TRUE)\ \{a, b, c, d\} \\
= & \ \{a, b, c, d\}\ TRUE \\
= & \\
= & \\
= & \\
= &
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \{a, b, c, d\} \\
=\ & \mathit{FIRST}\ \{a, b, c, d\} \\
=\ & (\lambda p.\, p\ \mathit{TRUE})\ \{a, b, c, d\} \\
=\ & \{a, b, c, d\}\ \mathit{TRUE} \\
=\ & \mathit{PAIR}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
=\ & \\
=\ & \\
=\ &
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \{a, b, c, d\} \\
&= \mathit{FIRST}\ \{a, b, c, d\} \\
&= (\lambda p.\, p\ \mathit{TRUE})\ \{a, b, c, d\} \\
&= \{a, b, c, d\}\ \mathit{TRUE} \\
&= \mathit{PAIR}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
&= (\lambda xyz.\, zxy)\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
&= \\
&=
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \{a, b, c, d\} \\
= {} & \mathit{FIRST}\ \{a, b, c, d\} \\
= {} & (\lambda p.p\ \mathit{TRUE})\ \{a, b, c, d\} \\
= {} & \{a, b, c, d\}\ \mathit{TRUE} \\
= {} & \mathit{PAIR}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
= {} & (\lambda xyz.zxy)\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
= {} & \ldots = \mathit{TRUE}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\}) \\
= {} &
\end{aligned}
$$

List (Church)

Example:

$$
\begin{aligned}
& \mathit{ISNIL}\ \{a, b, c, d\} \\
&= \mathit{FIRST}\ \{a, b, c, d\} \\
&= (\lambda p.p\ \mathit{TRUE})\ \{a, b, c, d\} \\
&= \{a, b, c, d\}\ \mathit{TRUE} \\
&= \mathit{PAIR}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
&= (\lambda xyz.zxy)\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
&= \ldots = \mathit{TRUE}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\}) \\
&= \ldots = \mathit{FALSE}
\end{aligned}
$$

List (Church) example

$$HEAD\ \{a, b, c, d\}$$

$$=$$

$$=$$

$$=$$

$$=$$

$$=$$

$$=$$

$$=$$

List (Church) example

$$
\begin{aligned}
& HEAD\ \{a, b, c, d\} \\
=\ & (\lambda z.FIRST\ (SECOND\ z))\ \{a, b, c, d\} \\
=\ & \\
=\ & \\
=\ & \\
=\ & \\
=\ & \\
=\ &
\end{aligned}
$$

List (Church) example

$$
\begin{aligned}
& HEAD \; \{a, b, c, d\} \\
&= (\lambda z.FIRST \; (SECOND \; z)) \; \{a, b, c, d\} \\
&= FIRST \; (SECOND \; \{a, b, c, d\}) \\
&= \\
&= \\
&= \\
&= \\
&=
\end{aligned}
$$

List (Church) example

$$
\begin{aligned}
& \mathit{HEAD}\ \{a, b, c, d\} \\
= & (\lambda z.\mathit{FIRST}\ (\mathit{SECOND}\ z))\ \{a, b, c, d\} \\
= & \mathit{FIRST}\ (\mathit{SECOND}\ \{a, b, c, d\}) \\
= & (\lambda p.p\ \mathit{TRUE})\ (\mathit{SECOND}\ \{a, b, c, d\}) \\
= & \\
= & \\
= & \\
= &
\end{aligned}
$$

List (Church) example

$$
\begin{aligned}
& \mathit{HEAD}\ \{a, b, c, d\} \\
&= (\lambda z.\mathit{FIRST}\ (\mathit{SECOND}\ z))\ \{a, b, c, d\} \\
&= \mathit{FIRST}\ (\mathit{SECOND}\ \{a, b, c, d\}) \\
&= (\lambda p.p\ \mathit{TRUE})\ (\mathit{SECOND}\ \{a, b, c, d\}) \\
&= \mathit{SECOND}\ \{a, b, c, d\}\ \mathit{TRUE} \\
&= \\
&= \\
&=
\end{aligned}
$$

List (Church) example

$$
\begin{aligned}
& \mathit{HEAD}\ \{a, b, c, d\} \\
&= (\lambda z.\mathit{FIRST}\ (\mathit{SECOND}\ z))\ \{a, b, c, d\} \\
&= \mathit{FIRST}\ (\mathit{SECOND}\ \{a, b, c, d\}) \\
&= (\lambda p.p\ \mathit{TRUE})\ (\mathit{SECOND}\ \{a, b, c, d\}) \\
&= \mathit{SECOND}\ \{a, b, c, d\}\ \mathit{TRUE} \\
&= (\lambda p.p\ \mathit{FALSE})\ \{a, b, c, d\}\ \mathit{TRUE} \\
&= \\
&=
\end{aligned}
$$

List (Church) example

$$
\begin{aligned}
& HEAD\ \{a, b, c, d\} \\
&= (\lambda z.FIRST\ (SECOND\ z))\ \{a, b, c, d\} \\
&= FIRST\ (SECOND\ \{a, b, c, d\}) \\
&= (\lambda p.p\ TRUE)\ (SECOND\ \{a, b, c, d\}) \\
&= SECOND\ \{a, b, c, d\}\ TRUE \\
&= (\lambda p.p\ FALSE)\ \{a, b, c, d\}\ TRUE \\
&= \{a, b, c, d\}\ FALSE\ TRUE \\
&=
\end{aligned}
$$

List (Church) example

$$
\begin{aligned}
& \mathit{HEAD}\ \{a, b, c, d\} \\
&= (\lambda z.\mathit{FIRST}\ (\mathit{SECOND}\ z))\ \{a, b, c, d\} \\
&= \mathit{FIRST}\ (\mathit{SECOND}\ \{a, b, c, d\}) \\
&= (\lambda p.p\ \mathit{TRUE})\ (\mathit{SECOND}\ \{a, b, c, d\}) \\
&= \mathit{SECOND}\ \{a, b, c, d\}\ \mathit{TRUE} \\
&= (\lambda p.p\ \mathit{FALSE})\ \{a, b, c, d\}\ \mathit{TRUE} \\
&= \{a, b, c, d\}\ \mathit{FALSE}\ \mathit{TRUE} \\
&= \mathit{PAIR}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{FALSE}\ \mathit{TRUE} \\
&= \ldots
\end{aligned}
$$

List (Church) example (continued)

$$HEAD\,\{a,b,c,d\}$$
$$= \ldots$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$
$$=$$

List (Church) example (continued)

$$HEAD\ \{a, b, c, d\}$$
$$= \ldots$$
$$= (\lambda xyz.zxy)\ FALSE\ (PAIR\ a\ \{b, c, d\})\ FALSE\ TRUE$$

$$=$$
$$=$$
$$=$$

$$=$$
$$=$$
$$=$$
$$=$$

List (Church) example (continued)

$$HEAD \; \{a, b, c, d\}$$

$$= \ldots$$

$$= (\lambda xyz.zxy) \; FALSE \; (PAIR \; a \; \{b, c, d\}) \; FALSE \; TRUE$$

$$= (\lambda yz.z \; FALSE \; y) \; (PAIR \; a \; \{b, c, d\}) \; FALSE \; TRUE$$

$$=$$

$$=$$

$$=$$

$$=$$

$$=$$

$$=$$

List (Church) example (continued)

$$
\begin{aligned}
& HEAD\,\{a, b, c, d\} \\
&= \ldots \\
&= (\lambda xyz.zxy)\ FALSE\ (PAIR\ a\ \{b, c, d\})\ FALSE\ TRUE \\
&= (\lambda yz.z\ FALSE\ y)\ (PAIR\ a\ \{b, c, d\})\ FALSE\ TRUE \\
&= (\lambda z.z\ FALSE\ (PAIR\ a\ \{b, c, d\}))\ FALSE\ TRUE \\
&= \\
&= \\
&= \\
&= \\
&=
\end{aligned}
$$

List (Church) example (continued)

$$
\begin{aligned}
& \mathit{HEAD}\ \{a, b, c, d\} \\
&= \ldots \\
&= (\lambda xyz.zxy)\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{FALSE}\ \mathit{TRUE} \\
&= (\lambda yz.z\ \mathit{FALSE}\ y)\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{FALSE}\ \mathit{TRUE} \\
&= (\lambda z.z\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\}))\ \mathit{FALSE}\ \mathit{TRUE} \\
&= \mathit{FALSE}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
&= \\
&= \\
&= \\
&=
\end{aligned}
$$

List (Church) example (continued)

$$\begin{aligned}
& HEAD \{a, b, c, d\} \\
&= \ldots \\
&= (\lambda xyz.zxy)\ FALSE\ (PAIR\ a\ \{b, c, d\})\ FALSE\ TRUE \\
&= (\lambda yz.z\ FALSE\ y)\ (PAIR\ a\ \{b, c, d\})\ FALSE\ TRUE \\
&= (\lambda z.z\ FALSE\ (PAIR\ a\ \{b, c, d\}))\ FALSE\ TRUE \\
&= FALSE\ FALSE\ (PAIR\ a\ \{b, c, d\})\ TRUE \\
&= \ldots = PAIR\ a\ \{b, c, d\}\ TRUE \qquad \text{(FALSE a b = b)} \\
&= \\
&= \\
&=
\end{aligned}$$

List (Church) example (continued)

$$
\begin{aligned}
& \mathit{HEAD}\ \{a, b, c, d\} \\
&= \ldots \\
&= (\lambda xyz.zxy)\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{FALSE}\ \mathit{TRUE} \\
&= (\lambda yz.z\ \mathit{FALSE}\ y)\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{FALSE}\ \mathit{TRUE} \\
&= (\lambda z.z\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\}))\ \mathit{FALSE}\ \mathit{TRUE} \\
&= \mathit{FALSE}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
&= \ldots = \mathit{PAIR}\ a\ \{b, c, d\}\ \mathit{TRUE} && \text{(FALSE a b = b)} \\
&= (\lambda xyz.zxy)\ a\ \{b, c, d\}\ \mathit{TRUE} \\
&= \\
&=
\end{aligned}
$$

List (Church) example (continued)

$$
\begin{aligned}
& \mathit{HEAD}\ \{a, b, c, d\} \\
&= \ldots \\
&= (\lambda xyz.zxy)\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{FALSE}\ \mathit{TRUE} \\
&= (\lambda yz.z\ \mathit{FALSE}\ y)\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{FALSE}\ \mathit{TRUE} \\
&= (\lambda z.z\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\}))\ \mathit{FALSE}\ \mathit{TRUE} \\
&= \mathit{FALSE}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
&= \ldots = \mathit{PAIR}\ a\ \{b, c, d\}\ \mathit{TRUE} && \text{(FALSE a b = b)} \\
&= (\lambda xyz.zxy)\ a\ \{b, c, d\}\ \mathit{TRUE} \\
&= \ldots = \mathit{TRUE}\ a\ \{b, c, d\}) && \text{(3 } \beta\text{-reductions)} \\
&=
\end{aligned}
$$

List (Church) example (continued)

$$
\begin{aligned}
& \mathit{HEAD}\ \{a, b, c, d\} \\
&= \ldots \\
&= (\lambda xyz.zxy)\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{FALSE}\ \mathit{TRUE} \\
&= (\lambda yz.z\ \mathit{FALSE}\ y)\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{FALSE}\ \mathit{TRUE} \\
&= (\lambda z.z\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\}))\ \mathit{FALSE}\ \mathit{TRUE} \\
&= \mathit{FALSE}\ \mathit{FALSE}\ (\mathit{PAIR}\ a\ \{b, c, d\})\ \mathit{TRUE} \\
&= \ldots = \mathit{PAIR}\ a\ \{b, c, d\}\ \mathit{TRUE} && \text{(FALSE a b = b)} \\
&= (\lambda xyz.zxy)\ a\ \{b, c, d\}\ \mathit{TRUE} \\
&= \ldots = \mathit{TRUE}\ a\ \{b, c, d\}) && \text{(3 } \beta\text{-reductions)} \\
&= \ldots = a && \text{(TRUE a b = a)}
\end{aligned}
$$

List (Church) cons

$$CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned} & CONS\ a\ NIL \\ = & \\ = & \\ = & \\ = & \end{aligned}$$

List (Church) cons

$$CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned} &CONS\ a\ NIL \\ &= (\lambda ht.PAIR\ FALSE\ (PAIR\ h\ t))\ a\ NIL \\ &= \\ &= \\ &= \end{aligned}$$

List (Church) cons

$$CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned}
& CONS\ a\ NIL \\
&= (\lambda ht.PAIR\ FALSE\ (PAIR\ h\ t))\ a\ NIL \\
&= (\lambda t.PAIR\ FALSE\ (PAIR\ a\ t))\ NIL \\
&= \\
&=
\end{aligned}$$

List (Church) cons

$$CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned}
&CONS\ a\ NIL \\
&= (\lambda ht.PAIR\ FALSE\ (PAIR\ h\ t))\ a\ NIL \\
&= (\lambda t.PAIR\ FALSE\ (PAIR\ a\ t))\ NIL \\
&= PAIR\ FALSE\ (PAIR\ a\ NIL) \\
&=
\end{aligned}$$

List (Church) cons

$$CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned}
&CONS\ a\ NIL \\
&= (\lambda ht.PAIR\ FALSE\ (PAIR\ h\ t))\ a\ NIL \\
&= (\lambda t.PAIR\ FALSE\ (PAIR\ a\ t))\ NIL \\
&= PAIR\ FALSE\ (PAIR\ a\ NIL) \\
&= \{a\}
\end{aligned}$$

List (Church) cons

$$CONS = \lambda h t.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$
\begin{aligned}
& CONS\ b\ (CONS\ a\ NIL) \\
= & \\
= & \\
= & \\
= & \\
= & \\
= &
\end{aligned}
$$

List (Church) cons

$$CONS = \lambda h t.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned}
& CONS\ b\ (CONS\ a\ NIL) \\
= & (\lambda h t.PAIR\ FALSE\ (PAIR\ h\ t))\ b\ (CONS\ a\ NIL) \\
= & \\
= & \\
= & \\
= & \\
= &
\end{aligned}$$

List (Church) cons

$$CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned}
& CONS\ b\ (CONS\ a\ NIL) \\
= & (\lambda ht.PAIR\ FALSE\ (PAIR\ h\ t))\ b\ (CONS\ a\ NIL) \\
= & (\lambda t.PAIR\ FALSE\ (PAIR\ b\ t))\ (CONS\ a\ NIL) \\
= & \\
= & \\
= & \\
= &
\end{aligned}$$

List (Church) cons

$$CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned}
& CONS\ b\ (CONS\ a\ NIL) \\
=\ & (\lambda ht.PAIR\ FALSE\ (PAIR\ h\ t))\ b\ (CONS\ a\ NIL) \\
=\ & (\lambda t.PAIR\ FALSE\ (PAIR\ b\ t))\ (CONS\ a\ NIL) \\
=\ & PAIR\ FALSE\ (PAIR\ b\ (CONS\ a\ NIL)) \\
=\ & \\
=\ & \\
=\ &
\end{aligned}$$

List (Church) cons

$$CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned}
& CONS\ b\ (CONS\ a\ NIL) \\
&= (\lambda ht.PAIR\ FALSE\ (PAIR\ h\ t))\ b\ (CONS\ a\ NIL) \\
&= (\lambda t.PAIR\ FALSE\ (PAIR\ b\ t))\ (CONS\ a\ NIL) \\
&= PAIR\ FALSE\ (PAIR\ b\ (CONS\ a\ NIL)) \\
&= PAIR\ FALSE\ (PAIR\ b\ (CONS\ a\ NIL)) \\
&= \\
&=
\end{aligned}$$

List (Church) cons

$$CONS = \lambda h t. PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned}
& CONS\ b\ (CONS\ a\ NIL) \\
= & (\lambda h t. PAIR\ FALSE\ (PAIR\ h\ t))\ b\ (CONS\ a\ NIL) \\
= & (\lambda t. PAIR\ FALSE\ (PAIR\ b\ t))\ (CONS\ a\ NIL) \\
= & PAIR\ FALSE\ (PAIR\ b\ (CONS\ a\ NIL)) \\
= & PAIR\ FALSE\ (PAIR\ b\ (CONS\ a\ NIL)) \\
= & PAIR\ FALSE\ (PAIR\ b\ (PAIR\ FALSE\ (PAIR\ a\ NIL))) \\
= &
\end{aligned}$$

List (Church) cons

$$CONS = \lambda ht.PAIR\ FALSE\ (PAIR\ h\ t)$$

$$\begin{aligned}
& CONS\ b\ (CONS\ a\ NIL) \\
= & (\lambda ht.PAIR\ FALSE\ (PAIR\ h\ t))\ b\ (CONS\ a\ NIL) \\
= & (\lambda t.PAIR\ FALSE\ (PAIR\ b\ t))\ (CONS\ a\ NIL) \\
= & PAIR\ FALSE\ (PAIR\ b\ (CONS\ a\ NIL)) \\
= & PAIR\ FALSE\ (PAIR\ b\ (CONS\ a\ NIL)) \\
= & PAIR\ FALSE\ (PAIR\ b\ (PAIR\ FALSE\ (PAIR\ a\ NIL))) \\
= & \{b, a\}
\end{aligned}$$

List encodings

We now have structured data *and* recursion!

List encodings

We now have structured data *and* recursion!

Don't forget, just as there are many ways to represent a List ADT in imperative programming, there are many possible encodings for lists and other structures in lambda calculus.

OUTLINE

- Revision – Lambda Calculus
- Y Combinator

- Encodings
  - numbers (a different way)
  - pairs
  - lists

- **Functional Programming**

Fibonacci

In the last tutorial, you probably implemented Fibonacci like this:


```
(defun fib (x)
  (if (< x 2)
    1
    (+ (fib (- x 1)) (fib (- x 2)))
  )
)
```

Fibonacci

In the last tutorial, you probably implemented Fibonacci like this:

```
(defun fib (x)
  (if (< x 2)
    1
    (+ (fib (- x 1)) (fib (- x 2)))
  )
)
```

This works, but is not very efficient (exponential time complexity!)

Fibonacci

In the last tutorial, you probably implemented Fibonacci like this:

```
(defun fib (x)
  (if (< x 2)
    1
    (+ (fib (- x 1)) (fib (- x 2))))
  )
)
```

This works, but is not very efficient (exponential time complexity!)

In imperative programming you would use variables to store the sequence (linear time complexity).

Fibonacci

In the last tutorial, you probably implemented Fibonacci like this:

```
(defun fib (x)
  (if (< x 2)
    1
    (+ (fib (- x 1)) (fib (- x 2)))
  )
)
```

This works, but is not very efficient (exponential time complexity!)

In imperative programming you would use variables to store the sequence (linear time complexity).

A comparable approach in FP is to compute the sequence, e.g. as a list.

Lists in LISP


```
nil                  ; an empty list
(cons e l)           ; prepend e to list l
(list a b c ...)     ; new list (a b c ...)
(car l)              ; the head element of l
(cdr l)              ; the tail list of l
(last l)             ; the last element l
(append a b)         ; combine two lists
(member e l)         ; first sublist starting e in l
(reverse l)          ; a mirror of the list
```

Lists in LISP (examples)

```
? (list 1 2 3)
(1 2 3)
? (cons 1 (cons 2 (cons 3 nil)))
(1 2 3)
? (member 2 (list 1 3 5))
NIL
? (member 3 (list 1 3 5))
(3 5)
? (cdr (list 1 2 3 4 5))
(2 3 4 5)
? (cdr (cdr (list 1 2 3 4 5)))
(3 4 5)
? (car (cdr (list 1 2 3 4 5)))
2
```

Fibonacci

Idea: given part of the Fibonacci sequence and a number, add that many more elements of the sequence.


```
(defun fib (n a)
  (if (zerop n)
    a
    (fib (- n 1)
      (cons
        (+ (car a) (car (cdr a)))
        a
) ) ) )

(fib 100 (list 1 0))
```

FIBONACCI

Making it a bit nicer:

- We can make optional (default) arguments
- (car (cdr x)) = (cadr x)
  - You can repeat the a, d as many times as required
  - e.g. (cadddr x) is the 4th element

```
(defun fib (n &optional (a (list 1 0)))
  (if (zerop n)
    a
    (fib (- n 1)
      (cons
        (+ (car a) (cadr a))
        a
))))

(fib 100)
```

A note on loops in LISP

I avoided using loops to keep the first example closer to lamda calculus.

There are several ways to use loops in LISP, here's one:


```
(defun fib (n)
  (loop for f1 = 0 then f2
        and f2 = 1 then (+ f1 f2)
        repeat n finally (return f1)))
```



```
(fib 100)
```

[1]

[1] source: https://www.cliki.net/Fibonacci

Java

```java
public boolean isPrime(long number) {
  return number > 1 &&
    LongStream
      .rangeClosed(2, (long) Math.sqrt(number))
      .noneMatch(index -> number % index == 0);
}
isPrime(9220000000000000039L) // Output: true
```

[2]

- “.rangeClosed” gives a stream of values within the range
- “.noneMatch” checks the stream against a predicate
- “variable -> expression” is a lambda abstraction!
  - It takes a value (index) from the range, and tests if it divides the number we're checking.

[2] source: https://www.voxxed.com/2015/12/functional-vs-imperative-programming-fibonacci-prime-and-factorial-in-

Java

```
public boolean isPrime(long number) {
  return number > 1 &&
    LongStream
      .rangeClosed(2, (long) Math.sqrt(number))
      .parallel()
      .noneMatch(index -> number % index == 0);
}
isPrime(9220000000000000039L) // Output: true
```

[3]

Adding ".parallel()" is enough magic sauce to get an embarassingly good speedup.

[3] source: https://www.voxxed.com/2015/12/functional-vs-imperative-programming-fibonacci-prime-and-factorial-in-

Python

If you write much Python you probably write more functional programming code than you thought.


```
>>> grades = [43, 68, 35, 89, 67, 65, 70]
>>> len(list(filter(lambda x: x>=50, grades)))
5
>>> sum(map(lambda x: x>=50, grades))
5
>>> [x + 5 for x in grades]
[48, 73, 40, 94, 72, 70, 75]
>>> max([x + 5 for x in grades])
94
```

Python

```
>>> from functools import reduce
>>> prices = [43, 68, 35, 89, 67, 65, 70]
>>> sales = [3, 5, 0, 3, 2, 10, 30]
>>> reduce(lambda x,y: x+y,
           map(lambda x: x[0]*x[1],
               zip(prices, sales)))
3620
```

- zip combines elements from two iterables into pairs
  - e.g. [(43, 3), (68, 5), ...]

Python

```
>>> from functools import reduce
>>> prices = [43, 68, 35, 89, 67, 65, 70]
>>> sales = [3, 5, 0, 3, 2, 10, 30]
>>> reduce(lambda x,y: x+y,
           map(lambda x: x[0]*x[1],
               zip(prices, sales)))
3620
```


- zip combines elements from two iterables into pairs
  - e.g. [(43, 3), (68, 5), ...]
- map applies a function to every element of an iterable
  - e.g. [43*3, 68*5, ...]

Python

```
>>> from functools import reduce
>>> prices = [43, 68, 35, 89, 67, 65, 70]
>>> sales = [3, 5, 0, 3, 2, 10, 30]
>>> reduce(lambda x,y: x+y,
              map(lambda x: x[0]*x[1],
                  zip(prices, sales)))
3620
```


- zip combines elements from two iterables into pairs
  - e.g. [(43, 3), (68, 5), ...]
- map applies a function to every element of an iterable
  - e.g. [43*3, 68*5, ...]
- reduce combines the elements using a two parameter function
  - (((0+129) + 340) + 0) + ...

Review

- Revision - Lambda Calculus
  - When α-reductions are *required*
  - η-reductions
- Y Combinator
  - Combinators
  - Fixed-Point Theorem
  - Y Combinator
  - Implementing recursion
- Encodings
  - numbers (a different way)
  - pairs
  - lists
- Functional Programming
  - Using lists in LISP
  - Stream processing in Java
  - Some ubiquitous Python