Lambda calculus & Functional programming

Matthew Might
University of Utah
matt.might.net

Why learn FP?

Advantages

- Less code
- Reasonable code
- Correct code
- Great for compilers!

Functional?

- Functions as first-class values
- Lexically scoped closures
- Immutable data structures
- Pure functions (no side effects)
- Equational reasoning

Functional++

- Strong type systems
- Type-inference (Hindley-Milner)
- Algebraic data types
- Pattern-matching
- Catamorphic programming
- Monadic programming
- Continuations

Origins

Two guys named Al

Al

Al

Two guys named Al

Alonzo Church

Alan Turing

History

- 1928: Alonzo Church publishes λ-calculus.
- 1935: Alan Turing publishes Turing Machine.
- 1936: λ-calculus equals Turing Machine.
- 1958: John McCarthy creates first Lisp.

λ-calculus

Math

- Numbers
- Sets
- Functions
- Variables
- Operators
- Relations
- Sequences
- Tuples

Math

- ~~Numbers~~
- ~~Sets~~
- Functions
- Variables
- ~~Operators~~
- ~~Relations~~
- ~~Sequences~~
- ~~Tuples~~

Math

- ~~Numbers~~
- ~~Sets~~
- Functions
- Variables

- ~~Operators~~
- ~~Relations~~
- ~~Sequences~~
- ~~Tuples~~

+ Anonymous functions

Three simple forms

- v
- $e_1(e_2)$
- $\lambda v.e$

Notation

$$f(x)$$
$$= f\ x$$
$$= (f\ x)$$
$$= (f)(x)$$
$$= f.x$$
$$= fx$$

By example

- $f(x) = x^2$
- $f = \lambda x.x^2$
- $f(3) = 9$
- $(\lambda x.x^2)(3) = 9$

Evaluating expressions

$(\lambda v.body)\ arg = body$, where $v = arg$

More examples

- $(\lambda x.x + 10)(3) = 13$
- $(\lambda f.f(x))(g) = g(x)$
- $(\lambda f.\lambda x.f(x))(g)(3) = g(3)$

More examples

- $(\lambda f.\lambda x.f(x))(\lambda x.x + 10)(3) = 13$

Regular calculus

$$(\lambda x.e)' = \lambda x.\frac{d}{dx}(e)$$

Lisp, Scheme, Racket

- v ≡ `v`
- λv.e ≡ `(lambda (v) e)`
- f(e) ≡ `(f e)`

Turing-complete!

But, how!?

Sugar λ into language.

Menu

- Multiple arguments
- Void value
- Lists
- Conditionals
- Numbers
- Recursion

Multiple arguments

$$f : X \times Y \to Z$$

$$f^C : X \to Y \to Z$$

$$f^C = \lambda x.\lambda y.f(x, y)$$

Multiple arguments

```
f(x,y) => ((f x) y)
```

Void

$$\mathbf{void} = \lambda\_.\_$$

Church's trick

Encode data according to how it's used.

Conditionals

$$\mathbf{true} \equiv \lambda c.\lambda a.c(\mathbf{void})$$

$$\mathbf{false} \equiv \lambda c.\lambda a.a(\mathbf{void})$$

$$\mathbf{if}\ e_b\ \mathbf{then}\ e_t\ \mathbf{else}\ e_f \equiv e_b\ (\lambda().e_t)\ (\lambda().e_f)$$

Numerals

$$n^C = \lambda f.\lambda z.f^n(z).$$

$$\mathbf{zero} \equiv \lambda f.\lambda z.z.$$

$$e_n + 1 \equiv \lambda f.\lambda z.f(e_n\ f\ z).$$

$$e_n + e_m \equiv \lambda f.\lambda z.(e_m\ f\ (e_n\ f\ z)).$$

$$e_m \times e_n \equiv \lambda f.\lambda z.(e_m\ (e_n\ f)\ z).$$

Lists

$$\mathbf{nil} \equiv \lambda e.\lambda l.e(\mathbf{void}).$$

$$\mathbf{cons} \equiv \lambda a.\lambda b.\lambda e.\lambda l.(l\ a\ b).$$

$$\mathbf{match}\ (e) \begin{cases} \mathbf{nil} & \mapsto e_e \\ \mathbf{cons}\ a\ b & \mapsto e_l \end{cases} \quad \equiv \quad e\ (\lambda().e_e)\ (\lambda a.\lambda b.e_l).$$

$$\langle e_1, e_2, \ldots, e_n \rangle \equiv \mathbf{cons}\ e_1\ (\mathbf{cons}\ e_2\ (\ldots(\mathbf{cons}\ e_n\ \mathbf{nil})\ldots)).$$

Recursion

Non-termination

What happens when we evaluate?

$$\Omega = (\ \lambda h.(h\ h))(\lambda h.(h\ h))$$

Recursion

Self-reference is the essence of recursion.

U Combinator

$$\mathbf{U} = \lambda h.(h\ h)$$

$$\Omega = \mathbf{U}(\mathbf{U})$$

Factorial

$$fact_{\mathbf{U}} = \mathbf{U}(\lambda h.\lambda n.\mathbf{if}\ (n \leq 0)\ \mathbf{then}\ 1\ \mathbf{else}\ n \times (h\ h)(n-1))$$

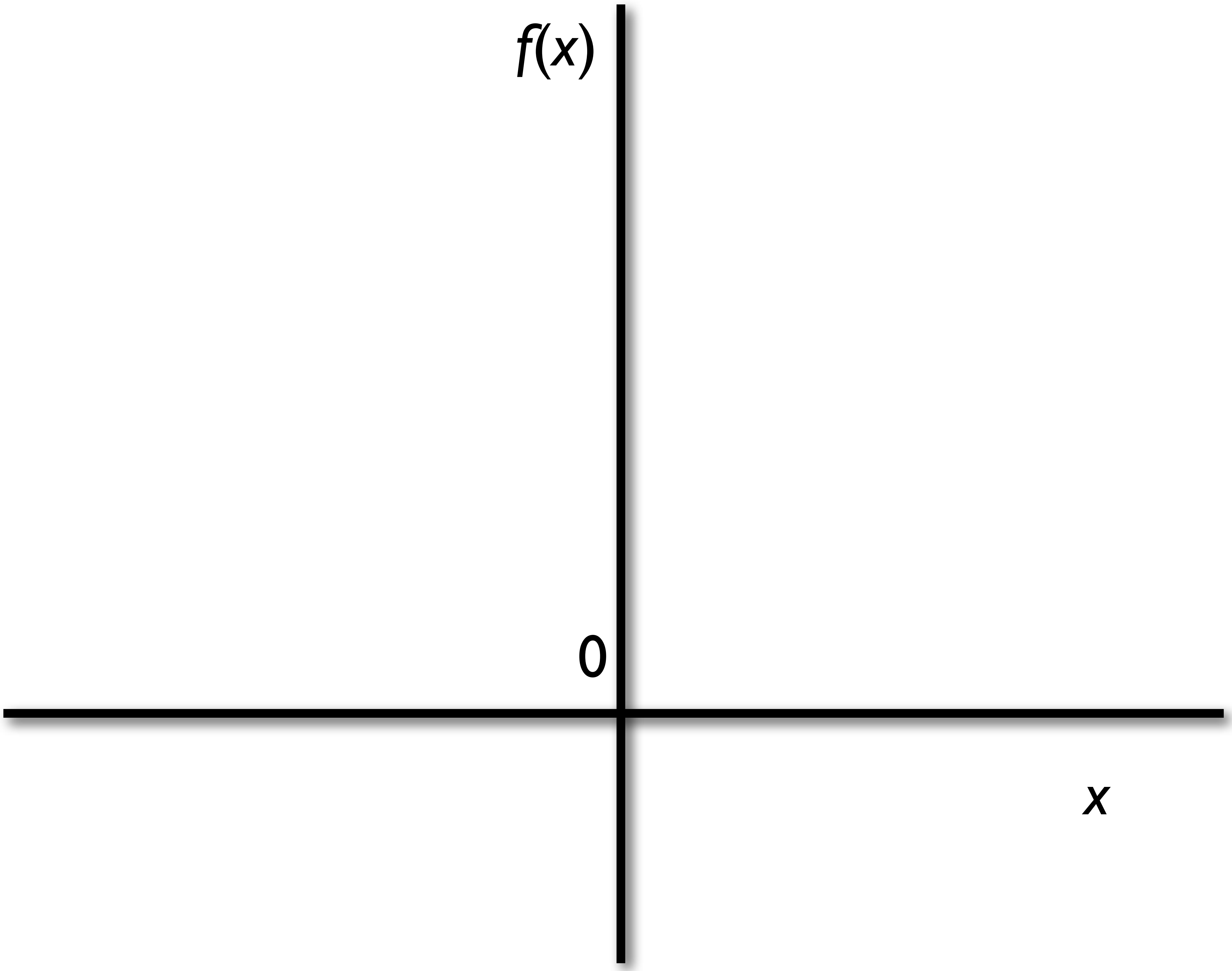
A little more elegance

Fixed points

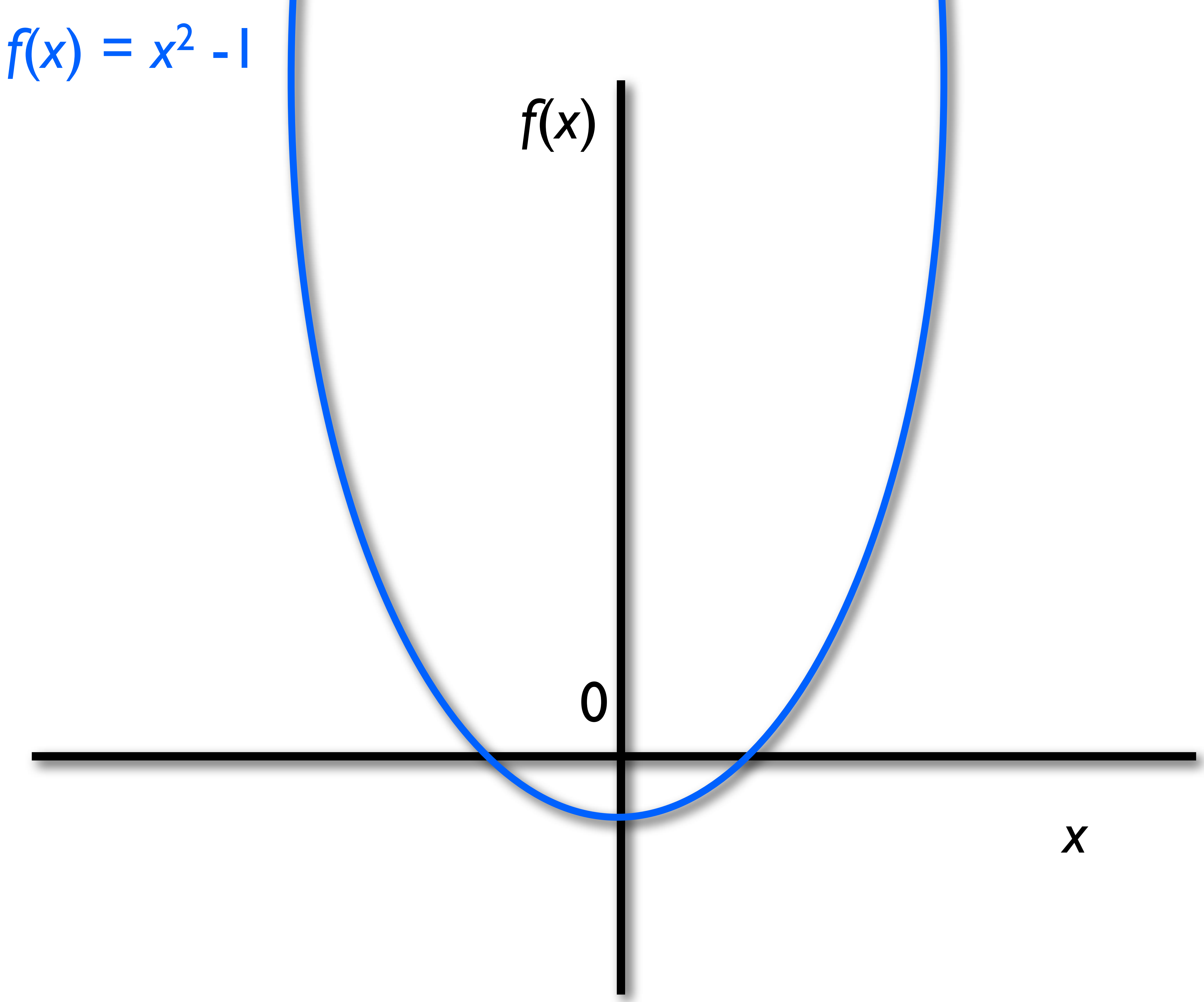
If $x = f(x)$, then the point x is
a **fixed point** of the function f.

Algebra

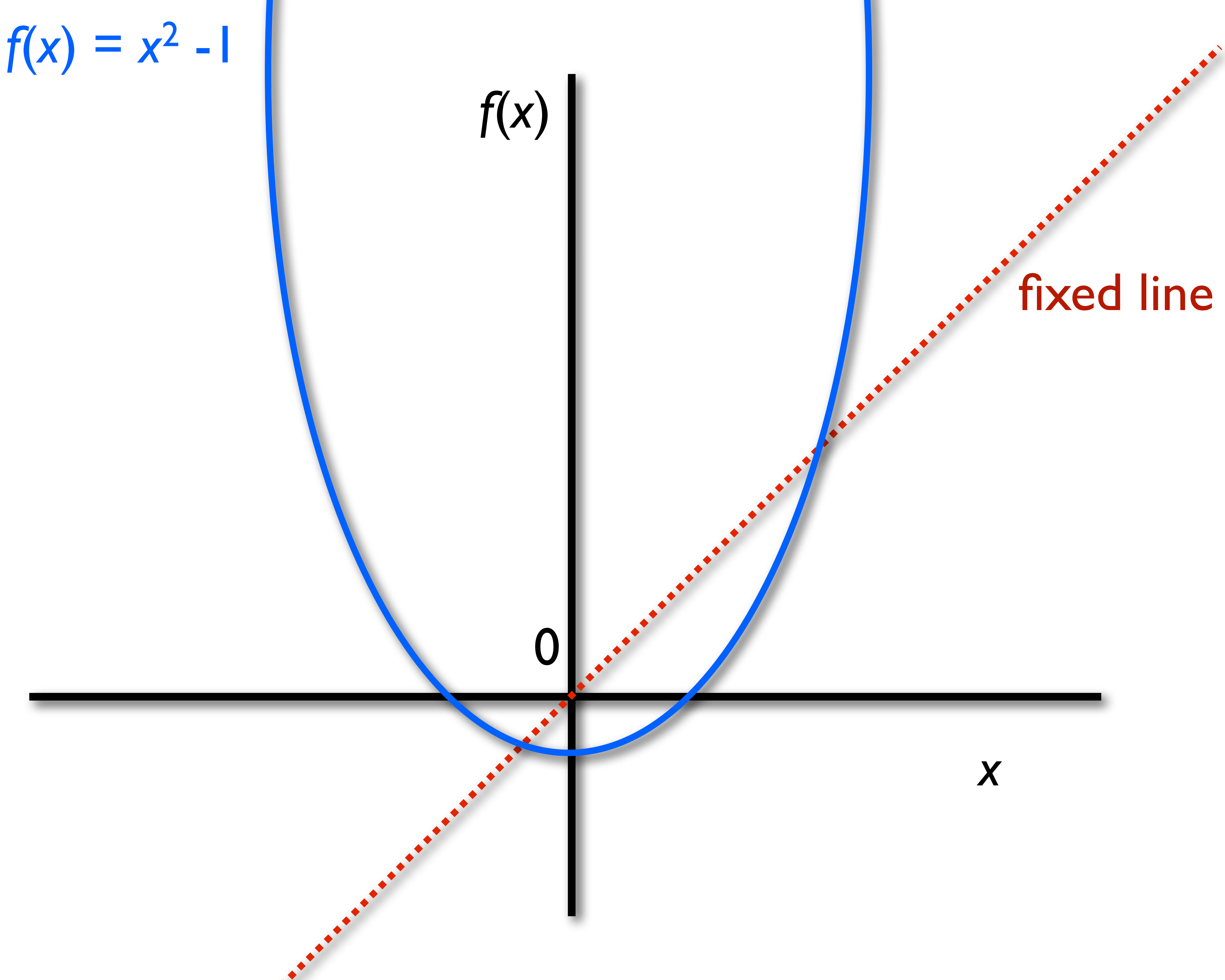
- $x = x^2 - 1$ is a recursive definition of x
- If $f(v) = v^2 - 1$, then $x = f(x)$.
- Solutions are the fixed points of f.



f(x)
0
x



$f(x) = x^2 - 1$
$f(x)$
0
x



$f(x) = x^2 - 1$
$f(x)$
fixed line
0
x

Factorial again

Factorial again

$$fact(n) = \textbf{if } (n \leq 0) \textbf{ then } 1 \textbf{ else } n \times fact(n-1)$$

Factorial again

$$fact(n) = \textbf{if } (n \leq 0) \textbf{ then } 1 \textbf{ else } n \times fact(n-1)$$

$$fact = \lambda n.\textbf{if } (n \leq 0) \textbf{ then } 1 \textbf{ else } n \times fact(n-1)$$

Factorial again

$$fact(n) = \textbf{if } (n \leq 0) \textbf{ then } 1 \textbf{ else } n \times fact(n-1)$$

$$fact = \lambda n.\textbf{if } (n \leq 0) \textbf{ then } 1 \textbf{ else } n \times fact(n-1)$$

$$fact = F(fact)$$

Factorial again

$$fact(n) = \textbf{if } (n \leq 0) \textbf{ then } 1 \textbf{ else } n \times fact(n-1)$$

$$fact = \lambda n.\textbf{if } (n \leq 0) \textbf{ then } 1 \textbf{ else } n \times fact(n-1)$$

$$fact = F(fact)$$

$$F(f) = \lambda n.\textbf{if } (n \leq 0) \textbf{ then } 1 \textbf{ else } n \times f(n-1)$$

Fixed-point finder

- We want function Y that finds fixed points
- Technically, $Y(F) = x$, such that $F(x) = x$.
- Start off derivation with $Y(F) = F(Y(F))$.

Solving for **Y**

$$Y(F) = F(Y(F))$$

Solving for **Y**

$$Y(F) = F(Y(F))$$

$$Y = \lambda F.F(Y(F))$$

Solving for **Y**

$$Y(F) = F(Y(F))$$

$$Y = \lambda F.F(Y(F))$$

$$Y = \mathbf{U}(\lambda h.\lambda F.F((h\ h)(F)))$$

Solving for **Y**

$$Y(F) = F(Y(F))$$

$$Y = \lambda F.F(Y(F))$$

$$Y = \mathbf{U}(\lambda h.\lambda F.F((h\ h)(F)))$$

Does this work?

Solving for **Y**

$$Y(F) = F(Y(F))$$

Solving for **Y**

$$Y(F) = F(Y(F))$$

$$Y = \lambda F.F(Y(F))$$

Solving for **Y**

$$Y(F) = F(Y(F))$$

$$Y = \lambda F.F(Y(F))$$

$$Y = \lambda F.F(\lambda x.(Y(F)(x)))$$

Solving for **Y**

$$Y(F) = F(Y(F))$$

$$Y = \lambda F.F(Y(F))$$

$$Y = \lambda F.F(\lambda x.(Y(F)(x)))$$

$$Y = \mathbf{U}(\lambda h.\lambda F.F(\lambda x.((h\ h)(F)(x))))$$

Y

$$\mathbf{Y} = (\lambda h.\lambda F.F(\lambda x.((h\ h)(F)(x))))(\lambda h.\lambda F.F(\lambda x.((h\ h)(F)(x))))$$

Factorial again

Factorial again

$$fact = F(fact)$$

$$F(f) = \lambda n.\mathbf{if}\ (n \leq 0)\ \mathbf{then}\ 1\ \mathbf{else}\ n \times f(n-1)$$

Factorial again

$$fact = F(fact)$$

$$F(f) = \lambda n.\mathbf{if}\ (n \leq 0)\ \mathbf{then}\ 1\ \mathbf{else}\ n \times f(n-1)$$

$$fact = \mathbf{Y}(F)$$

Factorial again

$$fact = F(fact)$$

$$F(f) = \lambda n.\mathbf{if}\ (n \leq 0)\ \mathbf{then}\ 1\ \mathbf{else}\ n \times f(n-1)$$

$$fact = \mathbf{Y}(F)$$

$$fact = \mathbf{Y}(\lambda f.\lambda n.\mathbf{if}\ (n \leq 0)\ \mathbf{then}\ 1\ \mathbf{else}\ n \times f(n-1))$$