

github.com/sjsyrek/presentations/lambda-calculus

1. Visit the code repo at the above link.
2. Download or clone the repo.
3. Open `lambda.html` and a browser console.
4. Alternatively, you can copy and paste the code from `javascript/lambda.js` directly into the console.
5. Confirm that all tests have passed.

λ calculus*

***for those who can't be bothered
to learn it**

Steven Syrek

@sjsyrek

definition

$\langle \text{expression} \rangle \quad := \langle \text{name} \rangle \mid \langle \text{function} \rangle \mid \langle \text{application} \rangle$
 $\langle \text{function} \rangle \quad := \lambda \langle \text{name} \rangle. \langle \text{expression} \rangle$
 $\langle \text{application} \rangle \quad := \langle \text{expression} \rangle \langle \text{expression} \rangle$

- x
- $\lambda x. x$
- $\lambda x. \lambda y. x y$
- $(\lambda x. x) y$
- $(\lambda x. x)(\lambda y. y)$

operations

α -substitution

β -reduction

η -conversion

- $(\lambda a. a) \equiv (\lambda z. z) \equiv (\lambda \text{Poop}. \text{Poop}) \equiv (\lambda \text{💩}. \text{💩})$
- $(\lambda x. \lambda y. x y) p q \rightarrow (\lambda y. p y) q \rightarrow p q$
- $(\lambda x. x x)(\lambda x. x x) \rightarrow (\lambda x. x x)(\lambda x. x x) \rightarrow \dots$
- $(\lambda x. f x) \leftrightarrow f$

identity combinator

- $\text{id} \equiv \lambda x. x$
- $\backslash x \rightarrow x$
- $x => x$
- `lambda x: x`
- $\rightarrow x \{ x \}$
- `|x| { x };`

boolean combinators

- $\text{true} \equiv (\lambda x. \lambda y. x)$
- $\text{false} \equiv (\lambda x. \lambda y. y)$
- $\text{and} \equiv (\lambda a. \lambda b. a b \text{ false})$
- $\text{or} \equiv (\lambda a. \lambda b. a \text{ true } b)$
- $\text{not} \equiv (\lambda a. a \text{ false true})$

reducing expressions

- $\text{and true false} \equiv$
- $(\lambda a. \lambda b. a b (\lambda x. \lambda y. y))(\lambda x. \lambda y. x)(\lambda x. \lambda y. y) \equiv$
- $(\lambda b. (\lambda x. \lambda y. x) b (\lambda x. \lambda y. y))(\lambda x. \lambda y. y) \equiv$
- $(\lambda x. \lambda y. x)(\lambda x. \lambda y. y)(\lambda x. \lambda y. y) \equiv$
- $(\lambda y. (\lambda x. \lambda y. y))(\lambda x. \lambda y. y) \equiv$
- $(\lambda x. \lambda y. y) \equiv$
- false

predicates

- $\text{isZero} \equiv (\lambda x. x \text{ false not false})$
- $\text{isZero}(n)(\text{true})(\text{false})$
- $\text{if} \equiv \lambda p. \lambda x. \lambda y. p \ x \ y$
- $\text{if}(\text{isZero}(n))(\langle \text{then} \rangle)(\langle \text{else} \rangle)$
- $\text{if} \equiv \text{id}$
- $P \ E_1 \ E_2 = \text{if } P \text{ is true then } E_1 \text{ else } E_2$

numbers

- $0 \equiv (\lambda f. \lambda x. x)$
- $1 \equiv (\lambda f. \lambda x. f(x))$
- $2 \equiv (\lambda f. \lambda x. f(f(x)))$
- $3 \equiv (\lambda f. \lambda x. f(f(f(x))))$
- $4 \equiv (\lambda f. \lambda x. f(f(f(f(x)))))$
- $5 \equiv (\lambda f. \lambda x. f(f(f(f(f(x))))))$
- ...

enumeration

- $\text{succ} \equiv (\lambda n. \lambda f. \lambda x. f(n f x))$
- $\text{pred} \equiv (\lambda n. n (\lambda p. \lambda z. z(\text{succ } (p \text{ true}))(p \text{ true}))$
 $(\lambda z. z \ 0 \ 0) \text{ false})$
- $\text{succ } 1 \equiv (\lambda n. \lambda f. \lambda x. f(n f x))(\lambda f. \lambda x. f(x)) \equiv$
 $(\lambda f. \lambda x. f(f(x))) \equiv 2$
- $\text{pred } 1 \equiv (\lambda n. n (\lambda p. \lambda z. z(\text{succ } (p \text{ true}))(p \text{ true}))$
 $(\lambda z. z \ 0 \ 0) \text{ false})(\lambda f. \lambda x. f(x)) \equiv (\lambda f. \lambda x. x) \equiv 0$

arithmetic

- $\text{add} \equiv (\lambda n. \lambda m. \lambda f. \lambda x. n \ f(m \ f \ x))$
- $\text{add} \equiv (\lambda n. \lambda m. m \ \text{succ} \ n)$
- $\text{sub} \equiv (\lambda m. \lambda n. n \ \text{pred} \ m)$
- $\text{mult} \equiv (\lambda n. \lambda m. \lambda f. n(m \ f))$
- $\text{mult} \equiv (\lambda n. \lambda m. m \ (\text{add} \ n) \ 0)$
- $\text{exp} \equiv (\lambda x. \lambda y. y \ x)$

arithmetic

- $\text{add } 1 \ 2 \equiv (\lambda n. \lambda m. \lambda f. \lambda x. n \ f(m \ f \ x))(\lambda f. \lambda x. f(x))(\lambda f. \lambda x. f(f(x))) = (\lambda f. \lambda x. f(f(f(x)))) \equiv 3$
- $\text{sub } 2 \ 1 \equiv (\lambda m. \lambda n. n \ (\lambda n. n \ (\lambda p. \lambda z. z((\lambda n. \lambda f. \lambda x. f(n \ f \ x))(p \ (\lambda x. \lambda y. x))))(p \ (\lambda x. \lambda y. x))))(\lambda z. z \ (\lambda f. \lambda x. x)(\lambda f. \lambda x. x))(\lambda x. \lambda y. y)) \ m)(\lambda f. \lambda x. f(f(x)))(\lambda f. \lambda x. f(x)) = (\lambda f. \lambda x. f(x)) \equiv 1$
- $\text{mult } 3 \ 0 \equiv (\lambda n. \lambda m. \lambda f. n(m \ f))(\lambda f. \lambda x. f(f(f(x))))(\lambda f. \lambda x. x) = (\lambda f. \lambda x. x) \equiv 0$
- $\text{exp } 2 \ 3 \equiv (\lambda x. \lambda y. y \ x)(\lambda f. \lambda x. f(f(x)))(\lambda f. \lambda x. f(f(f(x)))) = (\lambda f. \lambda x. f(f(f(f(f(f(f(x)))))))) \equiv 8$

recursion

- $\text{fix} \equiv (\lambda y. (\lambda x. y(x\ x))(\lambda x. y(x\ x)))$
- $F \equiv (\lambda f. \lambda n. ((\text{isZero } n) \ 1 \ (\text{mult } n \ (f(\text{pred } n)))))$
- $\text{fact} \equiv (\text{fix } F)$
- $((\lambda y. (\lambda x. y(x\ x))(\lambda x. y(x\ x))) \ F)$
- $(\lambda x. F(x\ x))(\lambda x. F(x\ x))$
- $(F((\lambda x. F(x\ x))(\lambda x. F(x\ x))))$
- $(F(\text{fix } F))$

factorial

- `fact 2`
- `(fix F) 2`
- `((λy. (λx. y(x x))(λx. y(x x))) F) 2`
- `((λx. F(x x))(λx. F(x x))) 2`
- `(F((λx. F(x x))(λx. F(x x)))) 2`
- `(F(fix F)) 2`
- `((λf. λn. ((isZero n) 1 (mult n (f(pred n))))) (fix F)) 2`
- `(λn. ((isZero n) 1 (mult n ((fix F)(pred n))))) 2`
- `((isZero 2) 1 (mult 2 ((fix F)(pred 2))))`
- `(mult 2 ((fix F)(pred 2)))`
- `(mult 2 ((fix F) 1))`
- `(mult 2 1)`
- `2`

factorial (expanded)

github.com/sjsyrek/malc

@sjsyrek