Self Interpreting the Lambda Calculus

Daniel Chapin

The Untyped Lambda Calculus

Variables: x, y, z...

The Untyped Lambda Calculus

Variables: x, y, z...

Abstractions: \x. x

The Untyped Lambda Calculus

Variables: x, y, z...

Abstractions: \x. x

Application: f x

The Untyped Lambda Calculus

(\x. x) \y. y

\y. y

The Untyped Lambda Calculus

```
(\x. x x) \y. y
(\y. y) \y. y
\y. y
```

The Untyped Lambda Calculus

(\x. \y. x) (\u. u) (\v. v)
(\y. \u. u) (\v. v)
\u. u

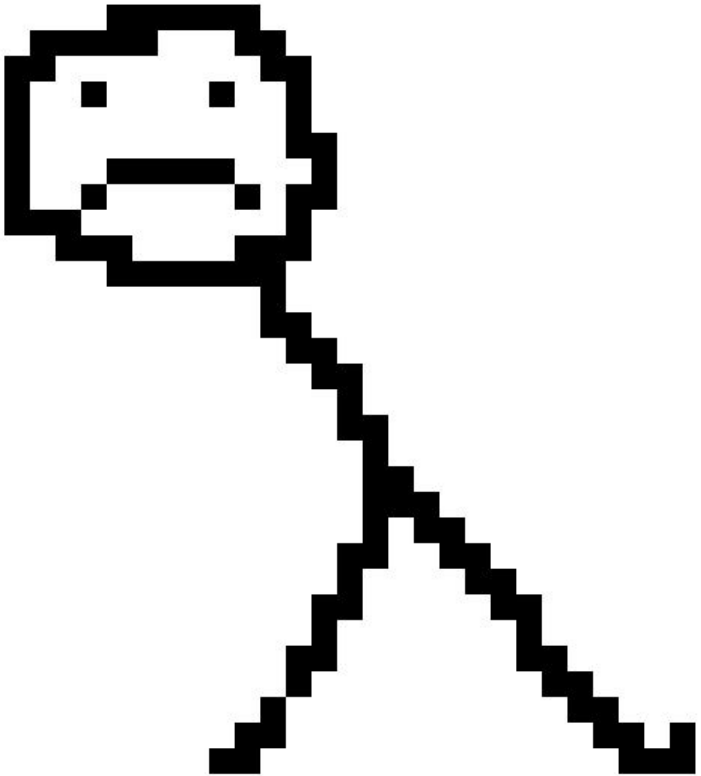
The Untyped Lambda Calculus

```
(\x. x x) (\x. x x)
(\x. x x) (\x. x x)
(\x. x x) (\x. x x)
...
```

The Untyped Lambda Calculus

\oper.(oper[0] ((\num.((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) ((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.f[0]) \x.x[0])) (((\left.\right.\oper.oper[0] left[2] right[1]) (((((\arg*.(\shift*.\d.\c.((((\abstraction.\application.\variable.\exp.(\pair.\func.pair[1] func[0]) exp[0] \flag.\val.(((\cond.\then.\else.cond[2] then[1] else[0]) ((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0]) \t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) flag[1])) (abstraction[5] val[0])) ((((\cond.\then.\else.cond[2] then[1] else[0]) ((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0]) \t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) (\num.(\pair.pair[0] \left.\right.right[0]) num[0]) flag[1])) (application[4] val[0])) (((\cond.\then.\else.cond[2] then[1] else[0]) ((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0]) \t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) (\num.(\pair.pair[0] \left.\right.right[0]) num[0]) ((\num.(\pair.pair[0] \left.\right.right[0]) num[0]) flag[1]))) (variable[3] val[0])) \ERROR.ERROR[0])) \body.(\body.((\left.\right.\oper.oper[0] left[2] right[1]) (((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.f[0]) \x.x[0])) body[0]) ((shift*[3] d[2] ((\num.((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) c[1])) body[0])) \exps.(\pair.\func.pair[1] func[0]) exps[0] \l.\r.((\l.\r.((\left.\right.\oper.oper[0] left[2] right[1]) ((\num.((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) ((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.f[0]) \x.x[0])) ((\left.\right.\oper.oper[0] left[2] right[1]) l[1]) r[0])) (shift*[5] d[4] c[3] l[1])) (shift*[5] d[4] c[3] r[0])) \x.(((\cond.\then.\else.cond[2] then[1] else[0]) (((\fun*.(\arg*.arg*[0] arg*[0]) \arg*.fun*[1] (arg*[0] arg*[0])) \lt*.\a.\b.(((\cond.\then.\else.cond[2] then[1] else[0]) ((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0]) \t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) b[0])) \t.\f.f[0]) ((((\cond.\then.\else.cond[2] then[1] else[0]) ((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0]) \t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) a[1])) \t.\f.t[1]) (lt*[2] ((\num.(\pair.pair[0] \left.\right.right[0]) num[0]) a[1])) ((\num.(\pair.pair[0] \left.\right.right[0]) num[0]) b[0]))) x[0] c[1])) ((\num.(((\left.\right.\oper.oper[0] left[2] right[1]) ((\num.(((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) (\num.(((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) (((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.f[0]) \x.x[0]))) num[0]) x[0])) ((\num.(((\left.\right.\oper.oper[0] left[2] right[1]) ((\num.(((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) (\num.(((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) (((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.f[0]) \x.x[0]))) num[0]) ((\fun*.(\arg*.arg*[0] arg*[0]) \arg*.fun*[1] (arg*[0] arg*[0])) \plus*.\a.\b.((\cond.\then.\else.cond[2] then[1] else[0]) ((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0]) \t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) a[1])) b[0] ((plus*[2] ((\num.(\pair.pair[0] \left.\right.right[0]) num[0]) a[1])) (\num.(((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) b[0])) x[0] d[2])) (arg*[0] arg*[0])) \arg*.(\shift*.\d.\c.((((\abstraction.\application.\variable.\exp.(\pair.\func.pair[1] func[0]) exp[0] \flag.\val.(((\cond.\then.\else.cond[2] then[1] else[0]) ((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0]) \t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) flag[1])) (abstraction[5] val[0])) ((((\cond.\then.\else.cond[2] then[1] else[0]) ((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0]) \t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) (\num.(\pair.pair[0] \left.\right.right[0]) num[0]) flag[1])) (application[4] val[0])) (((\cond.\then.\else.cond[2] then[1] else[0]) ((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0]) \t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) (\num.(\pair.pair[0] \left.\right.right[0]) num[0]) ((\num.(\pair.pair[0] \left.\right.right[0]) num[0]) flag[1]))) (variable[3] val[0])) \ERROR.ERROR[0])) \body.(\body.((\left.\right.\oper.oper[0] left[2] right[1]) (((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.f[0]) \x.x[0])) body[0]) ((shift*[3] d[2] ((\num.((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) c[1])) body[0])) \exps.(\pair.\func.pair[1] func[0]) exps[0] \l.\r.((\l.\r.((\left.\right.\oper.oper[0]

Human



Human

- Human Readable (ish)
- Function definition syntax
- Output declarations
- Import statements
- Macros

Human

(See human/class/ex1.human)

```
import util as {id};

def apply_self = \x.x x;

out ex1 = apply_self id;
```

Variables...

(\x. \y. x) \y. y

\y. \y. y

De Bruijn Indices

(\x. \y. x) \y. y
\y. \y. y

(\x. \y. 1) \y. 0
\y. \y. 0

De Bruijn Indices

\y. (\x. 0) 0


De Bruijn Indices

\y. (\x. 0) 0
\y. (\x. x) y

De Bruijn Indices in Human

\y. (\x. 0) 0
\y. (\x. x) y
\y. (\x. x[0]) y[0]

Lambda Calculus Encodings

true = \t. \f. t

false = \t. \f. f

Lambda Calculus Encodings

true = \t. \f. t

false = \t. \f. f

What about numbers?

Tuples? (see human/stl/)

The Paper

THEORETICAL PEARLS

Self-interpretation in lambda calculus

HENK BARENDREGT

Faculty of Mathematics and Computer Science, Catholic University of Nijmegen, The Netherlands

The Challenge... (encodings)

```
\oper.(oper[0] ((\num.((\left.\right.\oper.oper[0] left[2] right[1])
\t.\f.t[1]) num[0]) (\num.((\left.\right.\oper.oper[0] left[2] right[1])
\t.\f.t[1]) num[0]) (((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.f[0])
\x.x[0]))) ((((\fun*.(\arg*.arg*[0] arg*[0]) \arg*.fun*[1] (arg*[0] arg*[0]))
\plus*.\a.\b.((\cond.\then.\else.cond[2] then[1] else[0])
((\num.(\val.((\cond.\then.\else.cond[2] then[1] else[0]) val[0] \t.\f.f[0])
\t.\f.t[1]) ((\pair.pair[0] \left.\right.left[1]) num[0])) a[1])) b[0] ((plus*[2]
((\num.(\pair.pair[0] \left.\right.right[0]) num[0]) a[1]))
(\num.((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.t[1]) num[0]) b[0]))
(((\left.\right.\oper.oper[0] left[2] right[1]) \t.\f.f[0]) \x.x[0]))
\oper.(oper[0] \x.\y.x[1]) \oper.(oper[0] \x.\y.y[0]) \x.x[0])
```

The Solution

(See human/interpreter/interpreter.human)

Insights

- Raw LC is nightmare fuel

Insights

- Raw LC is nightmare fuel
- Debugging LC with lazy eval is harder


Insights

- Raw LC is nightmare fuel
- Debugging LC with lazy eval is harder
- Modules make insights