The λ-calculus

$$
\begin{aligned}
e \; ::= \;& var \\
\mid \;& e \; e \\
\mid \;& \lambda \; var \to e
\end{aligned}
$$

x p q
add
doStuff

(sub one) two

if b q (r,s)

(λ x → mul x x)

(λ f → λ x →
f (x x))

```
fibonacci :: Int -> Int
fibonacci n =
    if (n < 2)
    then 1
    else fibonacci (n-1)
       + fibonacci (n-2)
```

(Nat → Nat) → Nat → Nat

Nat → Nat

::

:: (Nat→Nat) → Nat → Nat

```
fibonacci = fix λ fib ->
  λ n ->
    if (lt n 2) (
      one
    )(
      add (fib (sub n one))
          (fib (sub n two))
    )
```

*

Booleans

$$\text{false} :: \text{Bool}$$
$$\text{true} :: \text{Bool}$$
$$\text{if} :: \text{Bool} \to a \to a \to a$$

$$\text{if true } e_1 \; e_2 \rightsquigarrow e_1$$

$$\text{if false } e_1 \; e_2 \rightsquigarrow e_2$$

$$(\lambda v \to e) \; t$$

$$(\lambda v \to \ldots v \ldots) \; t \rightsquigarrow \ldots t \ldots$$

$true = \lambda \bar{t}\ \bar{f} \to \bar{t}$

$false = \lambda \bar{t}\ \bar{f} \to \bar{f}$

$if = true$

$if\ \overset{false}{\cancel{true}}\ e_1\ e_2 \quad \rightsquigarrow \quad \cancel{e_1}\ e_2$

$true\ \overset{false}{\cancel{true}}\ e_1\ e_2$

$\cancel{true}\ false\ e_1\ e_2$

$\rightsquigarrow$

e_1

$if = \lambda b \to b$

$if\ b\ e_1\ e_2$

$\Updownarrow$

$b\ e_1\ e_2$

$\rightsquigarrow e_1 \qquad \rightsquigarrow e_2$

Naturals

```
data Nat
  = Zero
  | Succ Nat
```

zero :: Nat

succ :: Nat → Nat

lt :: Nat → Nat → Bool

add :: Nat → Nat → Nat

---

zero = $\lambda\ \bar{z}\ \bar{s} \to \bar{z}$

succ = $\lambda\ \cancel{z}\, n \to$
$\quad \lambda\ \bar{z}\ \bar{s} \to \bar{s}\ n$

lt n_1 n_2

$\begin{cases} \text{false} & \text{if } n_1 \not< n_2 \\ \text{true} & \text{if } n_1 < n_2 \end{cases}$

```
lt :: Nat -> Nat -> Bool
lt n1 n2 =
  case n1 of
    Zero ->
      case n2 of
        zero -> false
        succ m -> true

    succ n1 ->
      case n2 of
        zero -> false
        succ m ->
          lt n1 m
```

data Maybe a

= Just a

| Nothing

~~| Two a a~~

case

case m of

Just a $\rightarrow$ e_1

Nothing $\rightarrow$ e_2

case m

$\rightarrow$ (λ a $\rightarrow$ e_1) $\rightsquigarrow$ e_1

$\rightarrow$ (e_2) $\rightsquigarrow$ e_2

$$\mathrm{case} = \lambda\, m \to \lambda\, \bar{j}\, \bar{n} \to m\, \bar{j}\, \bar{n}$$

$$\mathrm{just} = \lambda x \to \lambda\, \bar{j}\, \bar{n} \to \bar{j}\, \cancel{a}\, x$$

$$\mathrm{nothing} = \lambda\, \bar{j}\, \bar{n} \to \bar{n}$$

$$\begin{aligned}
&\mathrm{just}\ a \\
&\quad (\lambda a \to e_1) \quad \rightsquigarrow \quad e_1 \\
&\quad (\quad e_2 \quad)
\end{aligned}$$

$$\begin{aligned}
&\mathrm{nathing} \\
&\quad (\lambda a \to e_1) \quad \rightsquigarrow \quad e_2 \\
&\quad (\quad e_2)
\end{aligned}$$

Arbitrary Data Types

$$\text{data } D$$
$$\mid C_1 \; a_{11} \cdots a_{1m_1}$$
$$\vdots$$
$$\mid C_n \; a_{n1} \cdots a_{1m_n}$$

① $\text{case} = \lambda m \to \cancel{\lambda \bar{c}_1 \cdots \bar{c}_n \to} m \cancel{\bar{c}_1 \cdots \bar{c}_n}$

② $c_i = \lambda a_{i1} \cdots a_{im_i} \to \lambda \bar{c}_1 \cdots \bar{c}_n \to \bar{c}_i \; a_{i1} \cdots a_{im_i}$

Scott Encoding

Recursion

$$\text{fibonacci} = (\text{fix } f) :: (Nat \to Nat) \to Nat \to Nat$$

$$\downarrow$$

$$f\ (\text{fix } f) \qquad\qquad Y$$

$$H = (\lambda x y.\ y(x\,x\,y))\ (\lambda x y.\ y(x\,x\,y))$$

$$H\ f = ((\lambda x y.\ y(x\,x\,y))\ (\lambda x y.\ y(x\,x\,y)))\ f$$

$$\downarrow$$

$$(\lambda y.\ y((\lambda x y.\ y(x\,x\,y))(\lambda x y.\ y(x\,x\,y))\,y)\ f$$

$$\downarrow$$

$$f((\lambda x y.\ y(x\,x\,y))(\lambda x y.\ y(x\,x\,y))\,f)$$

$$= f(H\ f)$$

So far

- Bool
  - true
  - false
  - if
- Nat
  - zero
  - succ
  - case
- Any data type (Scott encoding)
- Recursion (Turing's Θ combinator)

What else?

let x = e in b $\quad$ $(\lambda x \to b)\ e$

do x ← m ; n $\quad$ ((>>=) m)(λx → n)

[e | x ← l] $\quad$ do x ← l ; e

Type ✗ $\quad$ (GHC Core)

IO ✗

Fast Arrays ✗

Fast Anything ✗

$$(\lambda x \to 4)\ (\text{fib } 11)$$

$$4$$

$$(\lambda x \to 4)\ (\text{fix } (\lambda \text{fib} \to \dots))\ 11$$

$$\vdots$$

$$(\lambda x \to 4)\ 89$$

(Confluence)

$$(\lambda v \to \underbrace{\dots v \dots})\ e$$

Smaller ?

$$
\begin{aligned}
e \; ::= \;& S \\
|\;& K \\
|\;& I \\
|\;& e \; e
\end{aligned}
$$

SKI

$$S \; e_1 \; e_2 \; e_3 \leadsto (e_1 \; e_3)(e_2 \; e_3)$$

$$K \; e_1 \; e_2 \leadsto e_1$$

$$I \; e_1 \leadsto e_1$$

Micro Haskell

---

conv :: Lam → SKI with Vars

conv $(e_1 \; e_2)$ = (conv e_1)(conv e_2)

conv $(\lambda v \to e)$ = remove v (conv e)

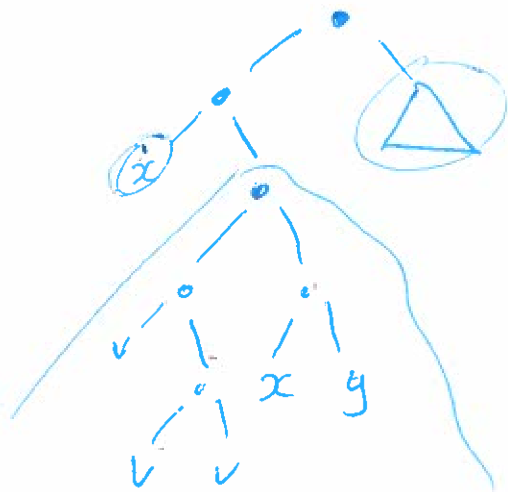
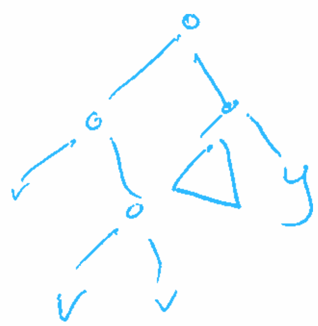
conv v = v

remove :: Var -> SKI with Vars -> SKI with Vars

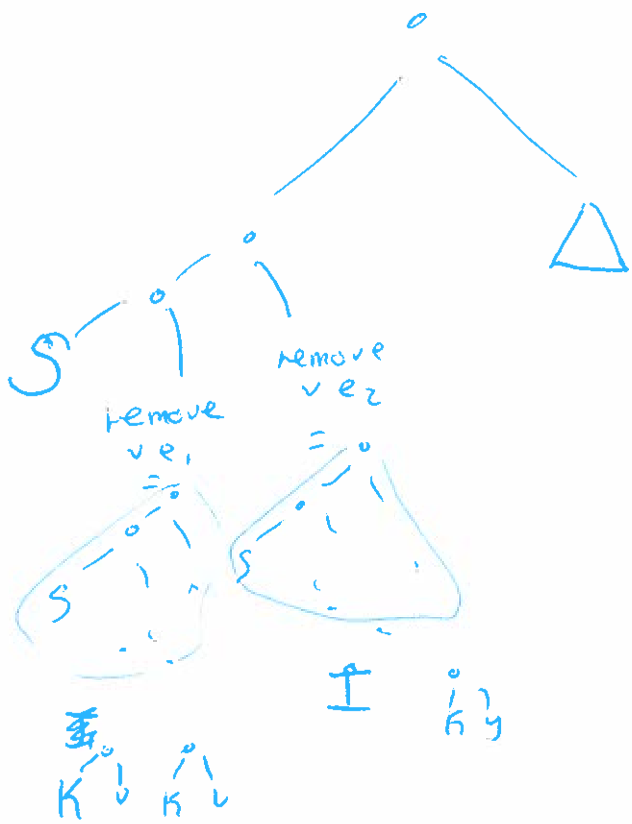
remove v (u)
  | v == u = I
  | v =/= u = K u

remove v (e₁ e₂) =
  S
    (remove v e₁)
    (remove v e₂)

(λv -> ...v...) e

~~~

...e...



Bracket Abstraction



Supercombinators
~~~

(remove v t) e



Even Smaller

Jeroen Fokker

$$X = \lambda f \to f\ S\ (\lambda p \ldots \to p)$$

$$K = X\ X$$

$$S = X\ (X\ X)$$

Applications

λ calculus

- Haskell core IR
- Research

SKI

- Haskell runtime (Augustsson's MicroHaska