Mogensen–Scott encoding

Alexander Gerasimov
for Berlin HUG, 2017

Church Encoding

Church Numerals

```
zero f x = x
one  f x = f x
two  f x = f (f x)
```

Church Numerals

```
zero f x = x
one  f x = f x
two  f x = f (f x)
```

```
succ n  = \f x -> f $ n f x
add m n = \f x -> n f $ m f x
```

Church Booleans

```
true  t f = t
false t f = f
```

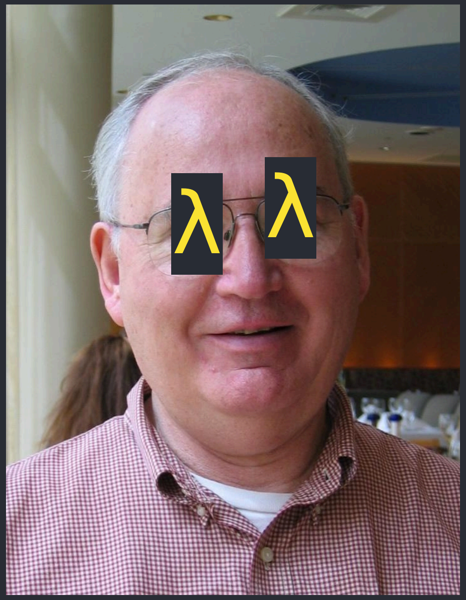
```
cond c t f = c t f
```

Closure = Memory

```
storeTriple a b c f = f a b c

let triple = storeTriple 5 23 37

f t = t \x y z -> …
f triple
```



Scott Encoding

Types with m constructors of varying arity, possibly recursive

```
data T
  = C1
  | C2 a
  | C3 b c
  | C4 d (T e)
  | C5 f g h
  ...
  | Cm z
```

```
data List a
  = Nil
  | Cons a (List a)
```

Scott Encoding

$$\lambda a_1\ a_2\ \ldots\ a_{\mathcal{A}_i} .\ \lambda f_i\ f_2\ \ldots\ f_m .\ f_i\ a_1\ a_2\ \ldots\ a_{\mathcal{A}_i}$$

$$C_i\ \underline{a_1\ a_2\ \ldots\ a_{\mathcal{A}_i}}\ \underline{f_1\ f_2\ \ldots\ f_m} = f_i\ \underline{a_1\ a_2\ \ldots\ a_{\mathcal{A}_i}}$$

m constructors with A_i arguments each

partially applied to a's but not f's

Scott Encoding

$$C_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}} \; \underline{f_1 \; f_2 \ldots f_m} = f_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}}$$

Boolean

$$m = 2$$

True $\mathcal{A}_1 = 0$

False $\mathcal{A}_2 = 0$

Scott Encoding

$$C_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}} \; \underline{f_1 \, f_2 \ldots f_m} = f_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}}$$

Boolean $C_i \, \underline{f_1 \, f_2} = f_i$

$$m = 2$$

True $\mathcal{A}_1 = 0$

False $\mathcal{A}_2 = 0$

```
true  f t = f
false f t = t
```

Scott Encoding

$$C_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underline{f_1\ f_2 \ldots f_m} = f_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

Enum: Suit

$m = 4$

Hearts	$\mathcal{A}_1 = 0$
Diamonds	$\mathcal{A}_2 = 0$
Clubs	$\mathcal{A}_3 = 0$
Spades	$\mathcal{A}_4 = 0$

```
data Suit
      = Hearts
      | Diamonds
      | Clubs
      | Spades
```

Scott Encoding

$$C_i\ \underline{a_1\ a_2\ \ldots\ a_{\mathcal{A}_i}}\ \underline{f_1\ f_2\ \ldots\ f_m} = f_i\ \underline{a_1\ a_2\ \ldots\ a_{\mathcal{A}_i}}$$

Enum: Suit

$$C_i\ \underline{f_1\ f_2\ f_3\ f_4} = f_i$$

$m = 4$

Hearts	$\mathcal{A}_1 = 0$	`hearts`	♡	♢	♧	♤	= ♡
Diamonds	$\mathcal{A}_2 = 0$	`diamonds`	♡	♢	♧	♤	= ♢
Clubs	$\mathcal{A}_3 = 0$	`clubs`	♡	♢	♧	♤	= ♧
Spades	$\mathcal{A}_4 = 0$	`spades`	♡	♢	♧	♤	= ♤

Scott Encoding

$$C_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underline{f_1\ f_2 \ldots f_m} = f_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

Read/Switch/Match

$$\textit{let}\ v = C_i\ a_1\ a_2 \ldots a_{\mathcal{A}_i}$$

$$\textit{then}\ \mathit{match}\ v\ \underline{f_1\ f_2 \ldots f_m} = v\ \underline{f_1\ f_2 \ldots f_m}$$

= application ($) of an m-argument function

Scott Encoding

$$C_i\ \underline{a_1\ a_2\ \ldots\ a_{\mathcal{A}_i}}\ \underline{f_1\ f_2\ \ldots\ f_m} = f_i\ \underline{a_1\ a_2\ \ldots\ a_{\mathcal{A}_i}}$$

Read/Switch/Match

$$match\ v\ \underline{f_1\ f_2\ \ldots\ f_m} = v\ \underline{f_1\ f_2\ \ldots\ f_m}$$

Boolean: cond/if

```
cond v t f = v t f
```

Suit: match

```
match v h d c s =
      v h d c s
```

Scott Encoding

$$C_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underline{f_1\ f_2 \ldots f_m} = f_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

Read/Switch/Match

$$\mathit{match}\ v\ \underline{f_1\ f_2 \ldots f_m} = v\ \underline{f_1\ f_2 \ldots f_m}$$

```
match v h d c s =
      v h d c s
```

```
case v of
  Hearts   -> h
  Diamonds -> d
  Clubs    -> c
  Spades   -> s
```

Scott Encoding

$$C_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}} \; \underline{f_1 \, f_2 \ldots f_m} = f_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}}$$

Read/Switch/Match

$$match \; v \; \underline{f_1 \, f_2 \ldots f_m} = v \; \underline{f_1 \, f_2 \ldots f_m}$$

nothing is stored

value-based flow control

Scott Encoding

$$\underline{C_i\ a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underline{f_1\ f_2 \ldots f_m} = f_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

Either

$m = 2$

Left $\mathcal{A}_1 = 1$

Right $\mathcal{A}_2 = 1$

Scott Encoding

$$C_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underline{f_1\ f_2 \ldots f_m} = f_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

Either

$m = 2$

Left $\mathcal{A}_1 = 1$

Right $\mathcal{A}_2 = 1$

$$C_i\ \underline{a_1}\ \underline{f_1\ f_2} = f_i\ \underline{a_1}$$

Scott Encoding

$$C_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underline{f_1\ f_2 \ldots f_m} = f_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

Either

$$C_i\ \underline{a_1}\ \underline{f_1\ f_2} = f_i\ \underline{a_1}$$

```
left  a l r = l a        either v l r = v l r
right a l r = r a
```

Scott Encoding

$$C_i\ \underbrace{a_1\ a_2\ \ldots\ a_{\mathcal{A}_i}}\ \underbrace{f_1\ f_2\ \ldots\ f_m} = f_i\ \underbrace{a_1\ a_2\ \ldots\ a_{\mathcal{A}_i}}$$

$\mathcal{A}_i$ => constructors can be polymorphic in arity

Scott Encoding

$$C_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underline{f_1\ f_2 \ldots f_m} = f_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

Maybe

$$m = 2$$

$$C_i\ \underline{a_{\mathcal{A}_i}}\ \underline{f_1\ f_2} = f_i\ \underline{a_{\mathcal{A}_i}}$$

Nothing	$\mathcal{A}_1 = 0$
Just	$\mathcal{A}_2 = 1$

Scott Encoding

$$C_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underline{f_1\ f_2 \ldots f_m} = f_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

Maybe

$$C_i\ \underline{a_{\mathcal{A}_i}}\ \underline{f_1\ f_2} = f_i\ \underline{a_{\mathcal{A}_i}}$$

```
nothing   n j = n
just    a n j = j a
```

```
maybe v n j = v n j
n :: b
j :: a -> b
```

Scott Encoding

$$C_i\ \underbrace{a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underbrace{f_1\ f_2 \ldots f_m} = f_i\ \underbrace{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

List

$m = 2$

Nil $\mathcal{A}_1 = 0$

Cons $\mathcal{A}_2 = 2$

```
data List a
  = Nil
  | Cons a (List a)
```

Scott Encoding

$$C_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}\ \underline{f_1\ f_2 \ldots f_m} = f_i\ \underline{a_1\ a_2 \ldots a_{\mathcal{A}_i}}$$

List

$$C_i\ a_{\underline{\mathcal{A}_i}}\ \underline{f_1\ f_2} = f_i\ \underline{a_{\mathcal{A}_i}}$$

```
nil          n c = n
cons v  l  n c = c v l
```

```
uncons l n c = l n c
n :: b
c :: h -> t -> b
```

Scott Encoding

List

```
nil        n c = n
cons v l n c = c v l
```

```
uncons l n c = l n c
```

```
n :: b
c :: a -> [a] -> b
```

```
l2_37 = cons 2 $ cons 37 nil
```

```
length l = uncons l 0 \h t -> 1 + length t
```

Scott Encoding

$$\underbrace{C_i\, a_1\, a_2 \ldots a_{\mathcal{A}_i}}\; \underbrace{f_1\, f_2 \ldots f_m} = f_i\, \underbrace{a_1\, a_2 \ldots a_{\mathcal{A}_i}}$$

Binary Tree

$m = 2$

Leaf $\mathcal{A}_1 = 0$

Node $\mathcal{A}_2 = 3$

```
data Tree a
  = Leaf
  | Node (Tree a) a (Tree a)
```

Scott Encoding

$$C_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}} \; \underline{f_1 \; f_2 \ldots f_m} = f_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}}$$

Binary Tree

```
leaf           l n  = l
node lt v rt l n  = n lt v rt
```

```
untree t l n = t l n
```

```
l :: b
n :: Tree a -> a -> Tree a
```

Scott Encoding

Binary Tree

```
leaf            l n  = l
node lt v rt l n  = n lt v rt

untree t l n = t l n


sum t = untree t 0 \l v r -> sum l + v + sum r
```

Scott Encoding

$$C_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}} \; \underline{f_1 \, f_2 \ldots f_m} = f_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}}$$

$\underline{f_1 \, f_2 \ldots f_m}$ can be encoded and passed as an **m**-tuple

$$\mathcal{T}_f$$

$$C_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}} \; \underline{\mathcal{T}_f} = f_i \; \underline{a_1 \; a_2 \ldots a_{\mathcal{A}_i}}$$

Mogensen Extension

$m = 3$

Term t $\mathcal{A}_1 = 1$

Application $f\,g$ $\mathcal{A}_2 = 2$

Abstraction $\lambda x.f\,x$ $\mathcal{A}_3 = 1$

Mogensen Extension

$m = 3$

```
fs = 
```
$\mathcal{T}_f$
```
f1 = fst fs, …
```

Term t

$\mathcal{A}_1 = 1$

```
term t fs = f1 t
```

Application $f\,g$

$\mathcal{A}_2 = 2$

```
ap f g fs = f2 $ f g
```

Abstraction $\lambda x.f\,x$

$\mathcal{A}_3 = 1$

```
abs f fs = f3 $ const f
```

Mogensen Extension

$\mathcal{M}\,[\lambda x.f\,(inc\ x)]$

$Abs\ (x \text{ -> } \mathcal{M}[f\,(inc\ x)])$

$Abs\ (x \text{ -> } App\ (\mathcal{M}[f]\ \mathcal{M}[(inc\ x)]))$

$Abs\ (x \text{ -> } App\ (\mathcal{T}(f)\ App(\mathcal{M}[inc]\ \mathcal{M}[x])))$

$Abs\ (x \text{ -> } App\ (\mathcal{T}(f)\ App(\mathcal{T}(f)\ \mathcal{T}(x))))$

Mogensen Extension

Evaluation

```
unmse λ ft fa fabs = λ ft fa fabs

ft t = t

fa g h = unmse g $ unmse h

fabs f = \x -> unmse $ f x
```

Thanks!