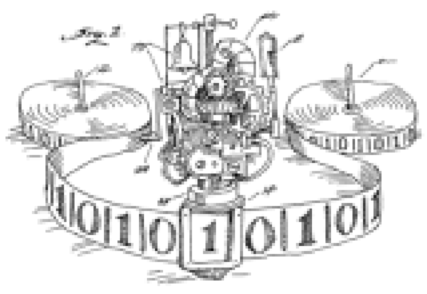
WSZYSTKO Z FUNKCJI

Marcin Benke

Dni Otwarte Kampusu Ochota, 18.04.2015

10101010

- “wszyscy wiedzą”, że komputery to zera i jedynki
- znamy maszynę Turinga



… ale skąd się wzięła?

W XVII wieku konstruowano maszyny liczące (np. Leibniz).

Ale czy da się skonstruować “maszynę myślącą”?

Hilbert, 1928

> *Czy istnieje algorytm, który potrafi rozstrzygać czy dana formuła logiczna jest prawdziwa?*

Żeby wykazać, że nie, trzeba skonstruować model obliczeń, który obejmie wszelkie możliwe algorytmy.

Turing: maszyna z nieskończoną taśmą i regułami działania

$\delta(q_0, 0) = (q_0, 1, R)$

$\delta(q_0, 1) = (q_1, 0, R)$

Church: funkcje

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

FUNKCJE

- $f(x) = x^2$
- $g(x) = x^2$
- funkcja f.. funkcja g.. funkcja x^2
- funkcja ax^2 ?
- jako funkcja x: $\lambda x.ax^2$
- jako funkcja a: $\lambda a.ax^2$

Tylko trzy konstrukcje:

$$
\begin{array}{lcll}
M & \rightarrow & x & \text{zmienna} \\
 & | & \lambda x.M & \text{definicja funkcji} \\
 & | & M_1(M_2) & \text{użycie funkcji}
\end{array}
$$

... i jedna reguła obliczenia: $(\lambda x.M)N \rightsquigarrow M[N/x]$

Pozwala obliczyć dokładnie to samo co maszyna Turinga.

Funkcję $f(x, y)$ reprezentujemy jako funkcję g argumentu x, która daje w wyniku funkcję argumentu y tak aby

$g(x)(y) = f(x, y)$

na przykład

$\lambda f \lambda x.fx$ — zastosowanie funkcji do argumentu

$\lambda f \lambda g \lambda x.f(gx)$ — złożenie funkcji f i g

Tekstowo λ zapisujemy jako `\`, np. `\x.x`

Chcemy mieć true, false, if tak, żeby

- *if true tak nie* $\rightsquigarrow$ *tak*
- *if false tak nie* $\rightsquigarrow$ *nie*

Chcemy mieć true, false, if tak, żeby

- *if true tak nie* $\rightsquigarrow$ *tak*
- *if false tak nie* $\rightsquigarrow$ *nie*

```
true = \x y. x
false = \x y. y

true x y = x
false x y = y
```

Chcemy mieć true, false, if tak, żeby

- *if true tak nie* $\rightsquigarrow$ *tak*
- *if false tak nie* $\rightsquigarrow$ *nie*

```
true = \x y. x
false = \x y. y

true x y = x
false x y = y

if b t e = b t e
```

KALKULATOR

http://benke.org/doko

Zdefiniuj not tak, żeby

```
not true = false
not false = true
```

Zdefiniuj not tak, żeby

```
not true = false
not false = true
```

```
not = \b. b false true
```

```
fst (pair x y) = x
snd (pair x y) = y
```

```
fst (pair x y) = x
snd (pair x y) = y

pair = \x\y\z.z x y
fst = \p.p true
snd = \p.p false
```

Pomysł:

$n\ f\ x = f^n(x)$

Pomysł:

$n\ f\ x = f^n(x)$

```
zero = \f x.x
```

```
one = \f x.f x
```

```
two = \f x.f(f x)
```

Jak zdefiniować funkcję następnika: `succ x = x+1` ?

Pomysł:

$n\ f\ x = f^n(x)$

`zero = \f x.x`

`one = \f x.f x`

`two = \f x.f(f x)`

Jak zdefiniować funkcję następnika: `succ x = x+1` ?

$succ : f^n(x) \mapsto f(f^n(x))$

Pomysł:

$n\,f\,x = f^n(x)$

```
zero = \f x.x
```

```
one = \f x.f x
```

```
two = \f x.f(f x)
```

Jak zdefiniować funkcję następnika: `succ x = x+1` ?

$succ : f^n(x) \mapsto f(f^n(x))$

```
succ = \n f x. f(n f x)
```

Dodawanie

```
succ two l o = l(l(l o))
add three two l o = l(l(l(l(l o))))
```

ĆWICZENIE 3 - ARYTMETYKA

Dodawanie

```
succ two l o = l(l(l o))
add three two l o = l(l(l(l(l o))))
```

```
add m n = m succ n
add = \m\n\f\x.m f (m f (n f x))
```

Mnożenie

Idea:

$(f^n)^m(x) = f^{m*n}(x)$


```
mul three two = l(l(l(l(l(l o))))))
```

ĆWICZENIE 3 - ARYTMETYKA

Dodawanie

```
succ two l o = l(l(l o))
add three two l o = l(l(l(l(l o))))
```

```
add m n = m succ n
add = \m\n\f\x.m f (m f (n f x))
```

Mnożenie

Idea:

$(f^n)^m(x) = f^{m*n}(x)$

```
mul three two = l(l(l(l(l(l o))))))
```

$f(0) = c$

$f(n+1) = h(n, f(n))$

Stworzymy ciąg par $(0, a_0), (1, a_1), \ldots, (n, a_n)$ taki, że

$a_0 = c;\ a_{i+1} = h(a_i);\ f(n) = snd(a_n)$

```
init = pair zero c
step = \p. pair (succ(fst p)) (h p)
f = \n. snd(n step init)
```

$$pred(0) = 0$$

$$pred(n + 1) = h(n, f(n))$$

$$h\,(x, y) = x$$

$pred(0) = 0$

$pred(n+1) = h(n, f(n))$

$h\,(x, y) = x$

```
init = pair zero zero
step = \x. pair (succ (fst x)) (fst x)
pred = \n. snd (n step init)
```

Odejmowanie

sub n m = m pred n

```
nil = pair true true
isnil = fst
cons h t = pair false (pair h t)
```

Ćwiczenie: napisz funkcje dające głowę i ogon listy

```
nil = pair true true
isnil = fst
cons h t = pair false (pair h t)
```

Ćwiczenie: napisz funkcje dające głowę i ogon listy

```
head = \z.fst(snd z)
tail = \z.snd(snd z)
```

Potęgowanie

```
exp m n = \f\x.(n m) f x
```

albo krócej

```
exp = \m n. n m
```

Listy inaczej

```
nil = false
cons = pair
head = fst
```

$(\lambda x.M)N \rightsquigarrow M[N/x]$

$M[N/x]$ oznacza “M z N wstawionym zamiast x”

$x[N/x] = N$

$y[N/x] = y$ (gdy $x \neq y$)

$(M_1(M_2))[N/x] = M_1[N/x]$

$(\lambda y.M)[N/x] = \lambda y.(M[N/x])$ (gdy y nie występuje w N).

Zakładamy, że wszystkie zmienne mają różne nazwy

Możemy to zawsze zapewnic odpowiednio zmnieniając nazwy:

$\lambda y.M$ jest równowazne $\lambda z.M[z/y]$

Tekstowo piszemy `\x.M` zamiast $\lambda x.M$

opuszczamy nawiasy tam gdzie niepotrzebne, MNP oznacza (MN)P