

# WSZYSTKO Z FUNKCJI

---

Marcin Benke

Dni Otwarte Kampusu Ochota, 18.04.2015

- “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

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

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

	$M \rightarrow x$	zmienna
Tylko trzy konstrukcje:	$\lambda x.M$	definicja funkcji
	$M_1(M_2)$	użycie funkcji

... 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  $\lambda$  zapisujemy jako  $\backslash$ , np.  $\backslash x. x$

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`

`true 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`

`true x y = y`

`if b t e = b t e`

<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:

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

$$\text{zero} = \lambda f\ x. x$$

$$\text{one} = \lambda f\ x. f\ x$$

$$\text{two} = \lambda f\ x. f(f\ x)$$

Jak zdefiniować funkcję następnika:  $\text{succ}\ x = x+1$  ?

Pomysł:

$$n \text{ f } x = f^n(x)$$

$$\text{zero} = \lambda f \ x. x$$

$$\text{one} = \lambda f \ x. f \ x$$

$$\text{two} = \lambda f \ x. f(f \ x)$$

Jak zdefiniować funkcję następnika:  $\text{succ } x = x+1$  ?

$$\text{succ} : f^n(x) \mapsto f(f^n(x))$$

Pomysł:

$$n \text{ f } x = f^n(x)$$

$$\text{zero} = \lambda f \ x. x$$

$$\text{one} = \lambda f \ x. f \ x$$

$$\text{two} = \lambda f \ x. f(f \ x)$$

Jak zdefiniować funkcję następnika:  $\text{succ } x = x+1$  ?

$$\text{succ} : f^n(x) \mapsto f(f^n(x))$$

$$\text{succ} = \lambda n \ f \ x. \ f(n \ f \ x)$$

## Dodawanie

$$\text{succ two } l \ o = l(l \ o))$$
$$\text{add three two } l \ o = l(l(l(l \ o))))$$

## Dodawanie

$$\text{succ two } 1 \text{ o} = 1(1(1 \text{ o}))$$

$$\text{add three two } 1 \text{ o} = 1(1(1(1(1 \text{ o}))))$$

$$\text{add } m \text{ n} = m \text{ succ } n$$

$$\text{add} = \lambda m \lambda n \lambda f \lambda x. m \text{ f } (m \text{ f } (n \text{ f } x))$$

## Mnożenie

Idea:

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

$$\text{mul three two} = 1(1(1(1(1(1 \text{ o}))))))$$

## Dodawanie

$$\text{succ two } 1 \text{ o} = 1(1(1 \text{ o}))$$

$$\text{add three two } 1 \text{ o} = 1(1(1(1(1 \text{ o}))))$$

$$\text{add } m \text{ n} = m \text{ succ } n$$

$$\text{add} = \lambda m \lambda n \lambda f \lambda x. m \text{ f } (m \text{ f } (n \text{ f } x))$$

## Mnożenie

Idea:

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

$$\text{mul three two} = 1(1(1(1(1(1 \text{ o}))))))$$

$$f(0) = c$$

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

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

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

`init = pair zero c`

`step = \p. pair (succ(fst p)) (h p)`

`f = \n. snd(n step init)`

$$\textit{pred}(0) = 0$$

$$\textit{pred}(n + 1) = h(n, f(n))$$

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



$$\text{pred}(0) = 0$$

$$\text{pred}(n + 1) = h(n, f(n))$$

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

```
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

$$\text{exp } m \ n = \backslash f \backslash x. (n \ m) \ f \ x$$

albo krócej

$$\text{exp} = \backslash m \ n. \ n \ m$$

## Listy inaczej

$$\text{nil} = \text{false}$$
$$\text{cons} = \text{pair}$$
$$\text{head} = \text{fst}$$

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

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

$$x[N/x] = N$$

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

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

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

Zakładamy, że wszystkie zmienne mają różne nazwy

Możemy to zawsze zapewnić odpowiednio zmieniając nazwy:

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

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

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