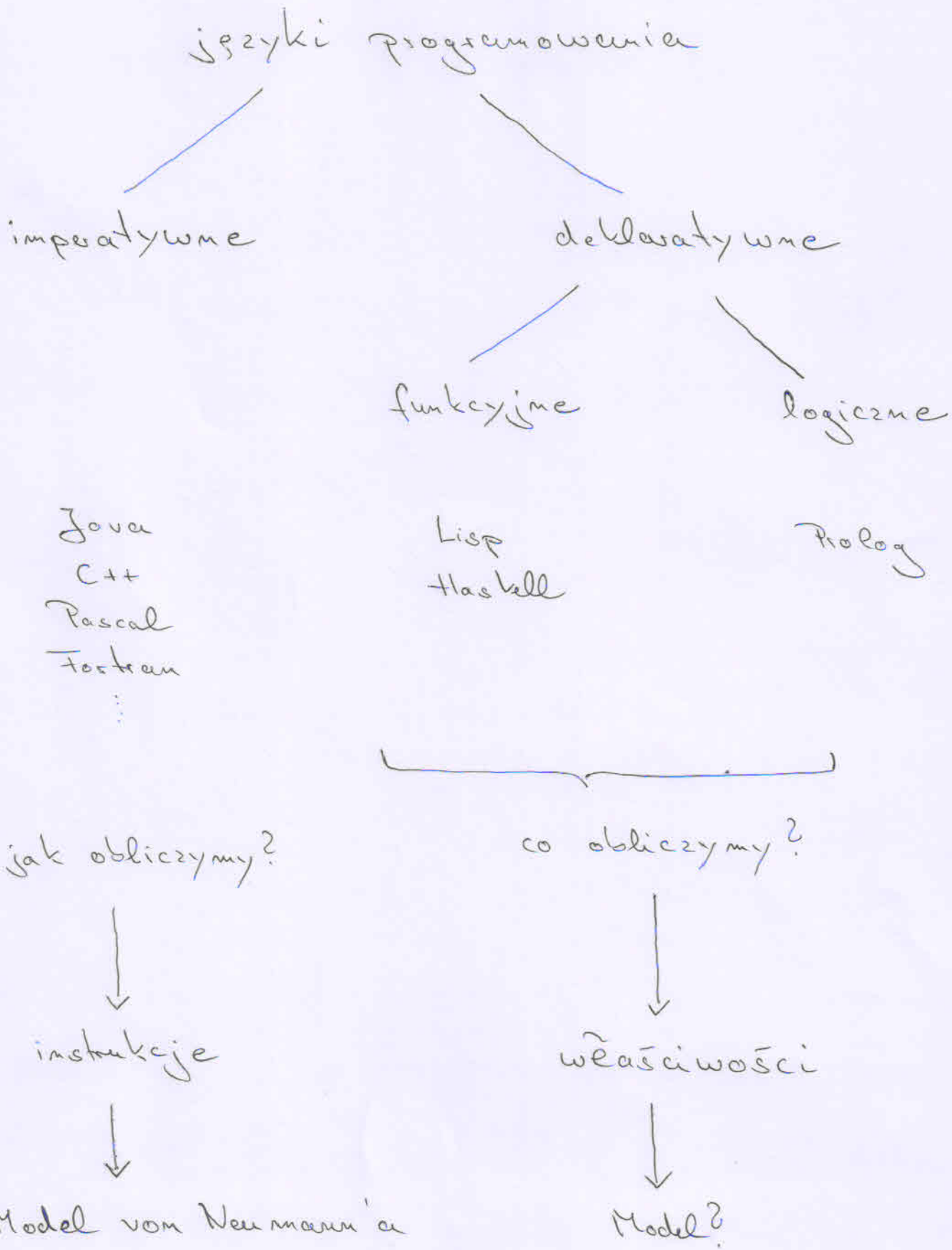
Programowanie funkcyjne

§1 Wprowadzenie

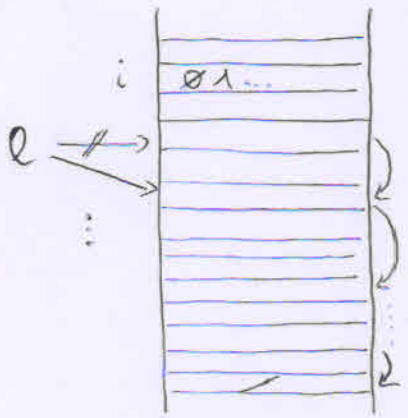


Przykład: Długość listy

(i) wersja imperatywna

"manipuluje wartościami zmiennych w pamięci"

```
int length ( lista l ) {
    int i := 0;
    while l.next ≠ NULL {
        i := i+1;
        l := l.next; }
    return i; }
```



(ii) wersja funkcyjna

"używa właściwości (funkcji) length"

- długość pustej listy, to 0.
- długość niepustej listy, to jeden plus długość ogona listy

Uwaga: Właściwości te (też) w wersji imperatywnej pokazują poprawność programu

```
length []   = 0
length (x:xs) = 1 + length xs
```

obliczenie, to "zastosowanie równań z lewej do prawej":

length [1,2]

$\Rightarrow$ 1 + length [2] — bo x = 1 i xs = [2] implikuje x:xs = [1,2]

$\Rightarrow$ 1 + (1 + length []) — bo x = 2 i xs = [] implikuje x:xs = [2]

$\Rightarrow$ 1 + (1 + 0)

$\overset{*}{\Rightarrow}$ 2

↳ obliczenie, to ewaluacja wyrażeń

Przykład: Argumenty funkcyjne

```
plus_5 x = x + 5
apply f = f 3

apply plus_5
  => plus_5 3      - bo f "=" plus_5
  => 3 + 5         - bo x "=" 3
  => 8
```

Przykład: quicksort

```
qs [] = []
qs (a:as) = qs [b | b <- as, b < a]
            ++ [a]
            ++ qs [b <- as, b > a]
```

Trochę historii

- Rachunek lambda (A. Church, 1930)

$(\lambda x.\lambda y.\ x+5)\ 3\ 5 \xrightarrow{*} 8$

$(\lambda f.\ f\ 3)(\lambda x.\ x+5)$

$\rightarrow (\lambda x.\ x+5)\ 3 \xrightarrow{*} 8$

- Lisp (McCarthy, 1960)

Listy . (1 2 3)

. (lambda (x) (+ x y))

((lambda (x) (+ x y)) 3 5) $\xrightarrow{*}$ 8

bez typowania,

ale z przepisaniem

- ML (Gordon, Milner, 1975)
  dla dowodów maszynowych (LCF)
  z typowaniem (bez deklaracji),
  ale z przepisaniem
- Hope (Burstall, McQueen, 1980)
  pattern matching
- Miranda (Turner, 1980)
  lazy evaluation
- Haskell (Hudak, Wadler, 1990)
  połączenie technik,
  klasy typów
- Ocaml (Leroy, 1988)
  Funnel (Odersky, 1999)

Haskell

- czyste funkcyjne
- typy polymorficzne
- mocno typowany
- pattern matching
- ewaluacja leniwa

Literatur:

- Hutton; Programming in Haskell
- Bird; Introduction to Functional Programming
- Abelson, Sussman; Structure and Interpretation of Computer Programs → Scheme