Vědy 4. Funkcionální jazyky a jejich vlastnosti. Lambda kalkulus, iterativní konstrukty a rekurze.

---

materialy vseobecne a z predmetu FUP:
https://drive.google.com/drive/u/0/folders/1fBBM7qEnWok71J58bDF0Ta4xK7oRelLx

---

Funkcionální jazyky a jejich vlastnosti

Funkcionální jazyky

- Haskell
- Lisp
- Closure
- Scala
- Erlang

Vlastnosti

- Deklarativní - využíváme funkce, k popisu závislostí mezi daty, např.: y = f(x).
- Výrazy se se vyhodnotí - výraz se nahradí hodnotou výrazu - často eager
- Program se spouští vyhodnocením počátečního výrazu

Pure functional programming

No side effects, no mutable data

- output of a function depends only on its inputs
- function does not change anything in evaluation
- can be evaluated in any order (many times, never)

More complex function based on recursion

- no for/while cycles
- natural problem decomposition
- mathematical induction

Lexikální vs dynamický scope

- Lexical scope – functions use bindings available where defined
- Dynamic scope – functions use bindings available where executed

Eval

Eval is a function defined in the top level context

```
(define x 5)
(define (f y)
  (eval '(+ x y)))
```

Fails because of lexical scoping

Otherwise the following would be problematic

```
(define (eval x)
      (eval-expanded (macro-expand x)))
```

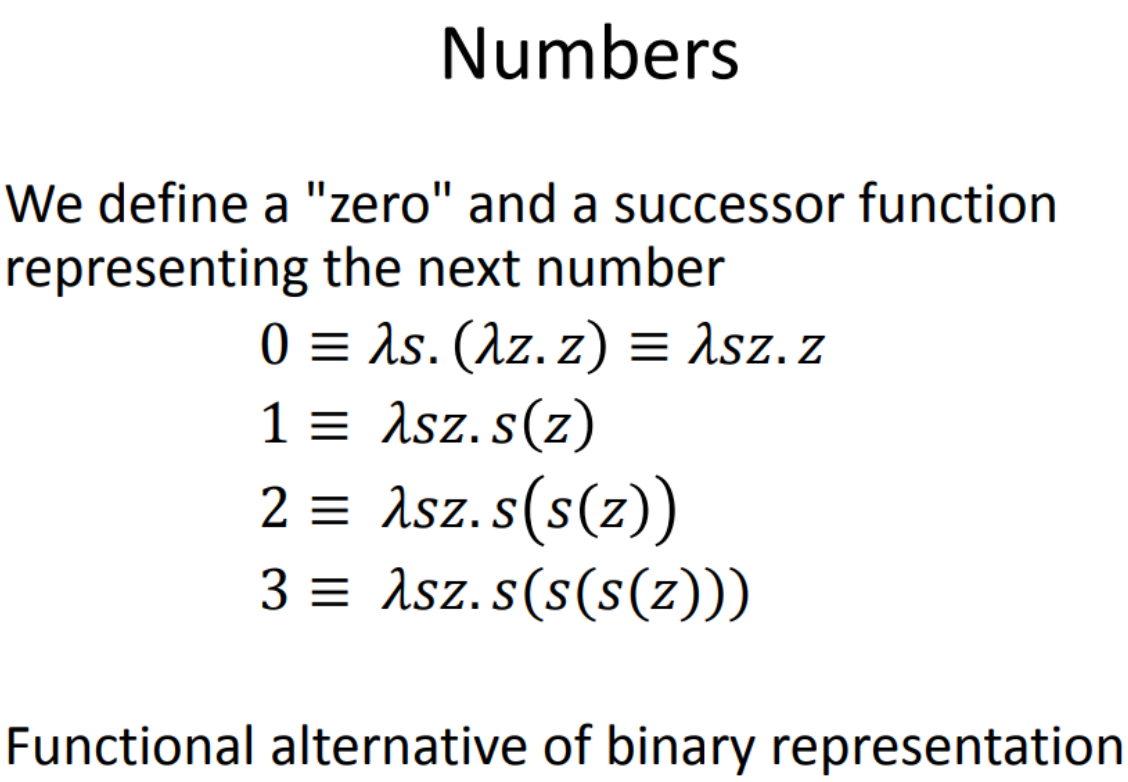
Lambda kalkulus

Theory developed for studying properties of effectively computable functions
Formal basis for functional programming

- as Turing machines for imperative programming

Why do I care?

- Understand that lambda and application is enough to build any program
  - without mutable state, assignment, define, etc.
- Understand how numbers, conditions, recursion can be created in a purely functional way
- Think about programming yet a little differently
- Have a clue when someone mentions λ-calculus
- Understand that Scheme syntax is not the worst



Vzhledem k tomu že na ČVUTU jsme si prošili peklem tak předpokládm že javascript zlávdáme nebo to nebude problém naučit se. Toto video vysvětluje lambda calculus na javscriptu.

https://youtu.be/3VQ382QG-y4

https://youtu.be/3VQ382QG-y4

Iterativní konstruktory

- Higher order functions take functions as arguments or return functions
- Used to capture/reuse common patterns
- Create fundamentally new concepts
- Filter, apply, map, fold, swap

List

The key data structure of Lisp
S-expressions are just lists
'(+ 1 2 3 4 5)
Lists can be created by a function
(list item1 item2 ... itemN)
List are linked lists of pairs with '() at the end
sometimes abbreviated by .
(cons 1 2), (cons 1 (cons 2 '())

Acumulator

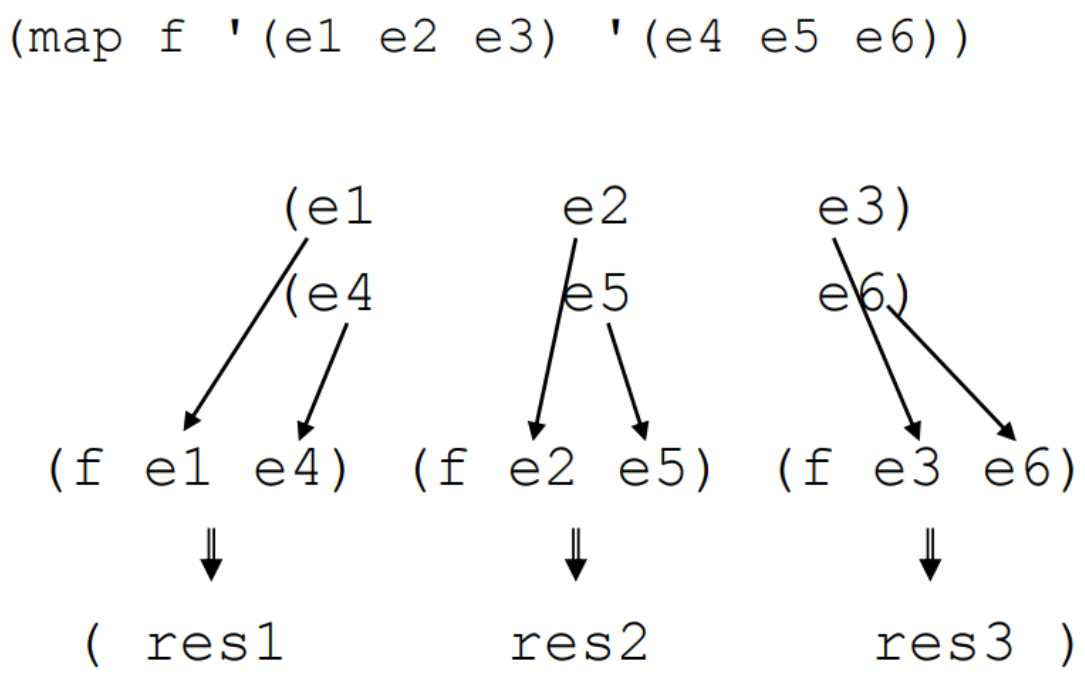
Slouží k realizaci dynamického programování ve funkcionálním světě. Často máme dvě funkce. Rekurzivní funkci, která řeší problém a pomocnou rekurzivní funkci, která řeší problém pomocí akumulátoru. Obvykle hlavní funkce volá pomocnou s prázdným akumulátorem. Fibonacciho posloupnost se dvěma akumulátory:

```
fib_h:: Integer->Integer->Integer->Integer
fib_h 1 acc1 acc2 = (acc1+acc2)
fib_h n acc1 acc2 = fib_h (n-1) acc2 (acc1+acc2)
fib:: Integer->Integer
fib 1 = 1
fib 2 = 1
fib n = fib_h (n-2) 1 1
```

Map

Zavolá proceduru s n argumenty na seznam hodnot. Výsledek vrátí jako seznam. Obvyklé použití je transformace dat z jedné množiny na druhou.

Map



Reduce

Bere seznam prvků a funkci, která ze dvou prvků udělá jeden. Typický příklad je třeba suma. Redukovací funkce by vypadala takto:

```
(x,y) => x+y
```

výstupem by byl součet prvků

Foldl

Prochází seznam zleva - a redukuje pomocí funkce, kterou předáme jako parametr.

Foldr

Prochází seznam zprava - a redukuje pomocí funkce, kterou předáme jako parametr.

Filter

Vrátí prvky z listu, které splňuji predikát.

- Prázdný list? → prázdný list

- Neprázdný list? → zavolej filter na zbytek listu → pokud první prvek splňuje predikát přidej ho k tomuto zbytku

Compose

Funguje stejně jako složená funkce v matematice. Argumentem je seznam funkcí. Výstupem je funkce, která je složeninou.

Apply

Aplikuje funkci na argument.

```
(apply + 1 2 3 '(4 5))
```

Let

Některé výpočty trvají dlouho. Je tedy vhodné si jejich výsledky uložit a použít je znovu.

Lambda funkce

- Anonymní funkce
- Definovaná přímo na místě použití
- Používáme, když potřebujeme předat funkci jako parametr, ale jinde nemá využití, je tedy zbytečné ji pojmenovávat
- Lisp: (lambda (parameters) body)
- Haskell: (\x y -> x + y)

Closure

- Speciální případ lambda funkce
- Kromě svých vlastních proměnných, může přistupovat k hodnotám proměnných svého prostředí
- Příklad z javy, ale princip je stejný

```
int z = 5;
(x,y) -> x + y + z;
```

Thunk

Lze implementovat využitím lambda funkce. Využívá se k odložení vyhodnocení výrazu. Mějme nějaký složitý výraz, jehož vyhodnocení trvá. Chceme jej předat jako parametr, ale funkce které ho předáváme si jej vyhodnotí jen když bude chtít. Toho docílíme tak, že jej zabalíme do lambdy.

```
(define-syntax w-delay (syntax-rules () ((w-delay expr) (lambda () expr))))
(define-syntax w-force (syntax-rules () ((w-force expr) (expr))))
```

Stream

Je to takový lazy list. Často se používá pro popis nekonečných množin (např.: všechna sudá čísla). Je definován jako dvojice:

- První prvek streamu
- Funkce, která vrací zbytek streamu jako stream

```
(define (ints-from n) (cons n (w-delay (ints-from (+ n 1)))))
```

Speciální případ streamu je prázdný stream. Se streamem pracujeme pomocí speciálních funkcí. Jednou z nich je například funkce **take**, která vezme prvních n prvků streamu.

Rekurze

Avoiding infinite recursion

1. First expression of the function is a cond
2. The first test is a termination condition
3. The "then" of the first test is not recursive
4. The cond pairs are in increasing order of the
5. amount of work required
6. The last cond pair is a recursive call
7. Each recursive call brings computations
8. closer to the termination condition

Tail recursion

Last thing a function does is the recursive call

Analytic / synthetic

Return value from termination condition / composed

Tree recursion

Function is called recursively multiple times (qsort)

Indirect recursion

Function A calls function B which calls A