Hraniční stavy

- **[]** – prázdný seznam, **[_]** – právě 1 prvek, **[A,B|T]** – alespoň 2 prvky
- reprezentace bin. stromu: **nil** – prázdný strom, **t(nil,X,nil)** – 1 prvek**, t(L,X,R)** – strom

Operátory/predikáty

- X+2 **=** 3+Y – žádost o unifikaci (obdoba zapsání výrazu rovnou do hlavy / těla)
- X1 **is** X + 1 – predikát číselného vyhodnocení výrazu (musí být napravo = v těle)
- X **@>=** Y, X **@<** Z – porovnání 2 libovolných termů, X **==** Y – termy jsou identické, **\==** ident. nejsou
- X **<** Y, X **>=** Y – aritmetické porovnání, X **=:=** Y – aritmetická rovnost, X **=\=** Y – aritm. nerovnost
- **/** – dělení, **//** - celočíselné dělení, **abs** – absolutní hodnota, ****** - mocnění?
- **var(X)** – ve chvíli vyhodnocování je X volná proměnná, **nonvar(X)** – proměnná už má hodnotu
- **fail.** – predikát, co okamžitě selže, **true.** – vždy pravda, (**repeat.** – cyklus)
- **atom(X)** – X je atom, **atomic(X)** – X je atom nebo číslo, **number(X)** – X je číslo

Zápis programu

- deklarace procedury: %nazev(**+**VstupniParametr,**-**VystupniParametr)
- každá klauzule ukončena tečkou, predikáty (v těle) odděleny čárkou / středníkem (= nebo)
- způsob vyhodnocování: backtracking a unifikace

Ostatní

- **S-K** – rozdílový seznam, spojení v konstantním čase – conc(A-B,B-C,A-C).
- **!** – operátor řezu – zmrazí dané substituce, zastaví backtrackování, neskáče na další řádku
- **not** P – negace, P nesmí být volná proměnná!
    - implementováno not(P) :- P, !, fail. not(P) :- true.

Použití akumulátoru – sémantika

- přidáváme další parametr – Akum
- **reverse seznamu**: odtrhnutou hlavu vkládáme do Akum – rev([X|T],A,V) :- rev(T,[X|A],V).
    - v rámci hraniční podmínky: kopírujeme na výstup – rev([],A,A).
    - tip: palindrom(S) = rev(S,S).
- [H|A] napravo = rovnou přidává při cestě tam (prvky budou odzadu, připojuje vždy hlavu)
- [H|A] nalevo – směrem tam = odebírání ze seznamu
    - směrem ven z rekurze = přidávání zpátky, v původním pořadí – insert(P,[H|T],[H|V]) :-
- o výstup V se nestaráme, hodnoty se do něj dostanou přes hraniční podmínku z Akum

Zpracování bin. stromové struktury s Akum *(memberTree, quicksort, fillTree, linearizeTree...)*

- sémantika: zpracuje daný seznam, připojí za něj druhý už zpracovaný, předá na výstup
- implementace:
    - odtrhává hlavu, podle té si určí – levá část / pravá část
    - zpracuje pravou část, připojuje za ni prázdný Akum, posílá do S1
    - zpracuje levou část, připojuje za ni hlavu H + seznam S1, posílá na výstup
    - v hraniční podmínce zkopíruje Akum na výstup

Zápis deklarace (hlavičky)

- max :: [Int] -> Int nebo max :: (Num a, Ord a) => [a] -> a
- klíčová slova: **Int, Bool, Ordering**, nebo **Ord** a, **Num** a, **RealFloat** a, **Eq** a
- **a** – hodnota daného typu, **[a]** – seznam hodnot, **(a,a)** – dvojice, **[(a,a)]** – seznam dvojic
  - **(a -> Bool)** -> ... – parametrem je funkce přijímající 1 parametr, vracející Bool
  - **... -> a** – poslední parametr v hlavičce = návratová hodnota funkce

Hlavní myšlenky

- způsob vyhodnocování: pattern matching (ne unifikace! není to obousměrné)
- hlavní specifika: vše je funkce, možnost curryfikace (částečné aplikace funkcí)
- parametry funkcí se oddělují mezerami, priorita vyhodnocování lze ovlivnit použitím závorek
- výstupní hodnota funkce se v těle funkce nastavuje zapsáním „funkce ..... **= hodnota**“
- není zde přetěžování funkcí (nemůže být stejná funkce s různým počtem parametrů)
- syntaxe je 2D – záleží na odsazování kódu

Programovací konstrukce

- matchování vůči patternám:
  - funkce *pattern* = příkaz **[]** – prázdný seznam, **[x]** – 1 prvek, **(x:xs)** – seznam, **xs** – cokoliv
  - ... = **error** “xxx“ – vyhození chybové hlášky
- varianty dané patterny
  - true / false upřesňující podmínky dané patterny
  - tvar: **| podmínka = příkaz** (uvozené odsazeným svislítkem, před ním se nepíše =)
  - př. **filter p (x:xs)** | p x == True = x : filter p xs | **otherwise** = filter p xs
- možnost pojmenování částí výpočtu
  - funkce ... = **let** a = ... b = ... **in** a + 2 * b
- vnořená funkce (př. zavolání varianty funkce s akumulátorem)
  - funkce x = funkce2 x y **where** funkce2 pattern1 = ... pattern2 = ...
- možnost dalšího řízení výpočtu (podmínkové příkazy, lze vložit napravo místo příkazu)
  - **if** podmínka **then** příkaz **else** příkaz (else větev musí být)
  - **case** objekt **of** pattern1 **->** příkaz pattern2 **->** příkaz
    - obdoba variant dané patterny, objekt = např. seznam
- pojmenování částí patterny (nebo výrazu) = as pattern
  - merge s**@**(x:xs) [] = s
- přímé zapsání funkce jako parametr = lambda výraz
  - \x = 2 * x nebo \x y = x + y
- naplnění seznamu konkrétními hodnotami = list comprehensions
  - fib = 1 : 1 : **[**a + b **|** (a,b) **<-** zip fib (tail fib)**]** let first = sort **[**a **|** a **<-** xs, a <= x**]** in ...

Operátory

- **x : acc** – dvojtečka = připojení hlavy k seznamu
- **first ++ second** – spojení dvou seznamů (pozor, v lineárním čase!)
- **f $ g x** – zřetězují se výsledky funkcí (abychom nemuseli psát závorky), **(f . g) x** – zřetězují se funkce

Užitečné funkce

- **length** – délka seznamu, **max** – největší prvek seznamu, **compare** – vrátí větší ze 2 hodnot
- **replicate** – zopakuje n-krát daný prvek, **reverse** – obrátí seznam
- **take** – vezme n prvků ze seznamu, **takeWhile** – vezme prvky ze seznamu, dokud splňují Bool funkci
- **zip** – ze 2 seznamů vytvoří seznam dvojic
- **zipWith** – provede funkci mezi prvním a druhým seznamem, hodnotu vrátí do třetího seznamu
- **elem** – prvek je v seznamu, **flip** – obrátí 2 parametry funkce
- **filter** – do nového seznamu umístí jen ty prvky ze seznamu, které vyhovují funkci
- **merge** – spojí 2 setříděné? seznamy, **flatten** – zlinearizuje pole polí hodnot do jednoho pole hodnot
- **groupBy** – ze vstupního seznamu podle (a -> a -> Bool) funkce udělá seznam seznamů (= skupin)
- **fst, snd** – vrátí první / druhý prvek dvojice (a,b)

Zjednodušení zápisu operací se seznamy

- předáváme binární funkci, startovní hodnotu (= akumulátor) a seznam
- **foldl** f z [] = z foldl f z (x:xs) = foldl f (f z x) xs ... **foldr** f z (x:xs) = f x (foldr f z xs)
    - př. sum = foldl (+) 0 reverse = foldl (\acc x -> x : acc) [] flatten = foldr (++) []
    - 3. parametr (seznam) se vezme pomocí curryfikace

Konstrukce nových typů (vlastní struktury, stromy...)

- **data** – konstrukce nového typu, nalevo = typový konstruktor, napravo = datové konstruktory
    - definice: **data** Point **=** Pt Float Float **data** Shape **=** Circle Point Float **|** Rectagle ... ...
        - volání: *spocitejObsah* (Circle (Pt 0 0)) – dosazujeme konkrétní hodnoty
    - možnost pojmenování parametrů: **data** Person = Person **{** firstName **::** String, ... **}**
        - volání: *vytiskni* (Person { firstname="jmeno", ... })
    - využití jako výčtové hodnoty: **data** Day = Monday **|** Tuesday **|** ...
    - definice stromu:
        - **data** Tree a **=** Empty **|** Node a (Tree a) (Tree a)
        - př. následná deklarace funkce: treeInsert :: (Ord a) => a -> Tree a -> Tree a
        - a matchování vůči patternám: treeInsert x Empty = ...
        - rychlé vložení hodnot do stromu: foldr treeInsert Empty [1,2,3,4]
- **type** – typová synonyma (možnost pojmenování už vytvořeného objektu jiným jménem)
    - **type** Name **=** String **type** PhoneBook **=** [(Name, PhoneNumber)]

Další konstrukty

- „nullable typ" – **Maybe a**
    - př. **Maybe Int** – může obsahovat buď hodnotu **Just číslo**, nebo **Nothing**
- **String** je typové synonymum pro **[Char]**
    - připojení 1 znaku na začátek = dvoutečkou (jako seznam)
    - vypsání hodnoty: **show x**
    - spojování stringů: **shows** hodnota string
        - připojí danou hodnotu ke stringu, funguje správně (v lineárním čase)
        - lze řetězit tečkou: **(shows** 123 **. shows** 456**)** "abcde"