

Kapitola 1

Úvod

Trojsemestrálny kurz matematického programovania.

- 1.semester - Základy funkcionálneho programovania (deklaratívneho) programovania v jazyku CL.
- 2.semester - Výroková, predikátová logika, dokazovanie v proof systéme CL.
- 3.semester - Dokazovanie vlastností programov (hlavne pomocou indukcie) s využitím znalostí z 1. a 2. semestra.

1.1 Imperatívne programovanie verzus deklaratívne programovanie

1.1.1 Imperatívne programovanie.

- Programy sú postupnosti príkazov, recepty ako niečo spraviť.
- Výpočet pozostáva z vykonávania príkazov, ktoré modifikujú pamäť.
- Ťažká otázka: čo vlastne program počíta, aké sú jeho vlastnosti, či spĺňa nami požadované vlastnosti (špecifikáciu).
- V teoretickej informatike existuje komplikovaná, dosť neprehľadná teória - sémantika procedurálnych programov, ktorá sa snaží riešiť tieto spomenuté ťažké otázky.
- Jazyky: PASCAL, C, C++, FORTRAN, COBOL, ASSEMBLER.

1.1.2 Deklaratívne programovanie.

- Má matematické základy, pracujeme s matematickými objektami.
- Programy sú vlastne definície funkcií a predikátov.
- Zhruba podľa toho rozdelenie na funkcionálne a logické programovanie.
- Na zápis samotných definícií funkcií a predikátov (ako program počíta) a vlastností, špecifikácií (čo program počíta) používame jednotne **ten istý jazyk logiky**.
- *Jazyk logiky* pozostáva z nasledujúcich množín symbolov:
 - množina premenných: x, y, z, \dots
 - množina funkčných symbolov: f, g, h, \dots
 - množina predikátových symbolov: p, q, r, \dots
 - množina logických spojok: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
 - množina kvantifikátorov: \forall, \exists
 - množina pomocných symbolov: $'(')''', ''$
- Z matematickej algebry a analýzy už máte bohatú prax v zapisovaní vlastností a definícií matematických objektov pomocou logických formúl v hore uvedenom jazyku. Teraz ju len rozšírime na programovanie.
- Výpočet hodnoty pre nejakú definovanú funkciu je vlastne vyhodnocovanie, zjednodušovanie, prepisovanie výrazov do základného ďalej už neredukovateľného tvaru. Pre ilustráciu uvažujme nasledovnú jednoduchú definíciu funkcie $F(x)$:

$$F(x) = 2 \cdot (x + 4) .$$

Potom výpočet hodnoty $F(3)$ bude takéto vyhodnocovanie, prepisovanie výrazov: $F(3) = 2 \cdot (3 + 4) = 2 \cdot 7 = 14$. Intuitívne vidíme, že decimálna konštanta 14 je dostatočne jednoduchý výraz (pre užívateľa), ktorý už nemá zmysel ďalej redukovať. V ďalšom texte sa k pojmu výpočtu ešte podrobnejšie vrátíme a ukážeme si mnohé už menej triviálne príklady.

- Jazyky: LISP, SCHEME, HASKELL, MIRANDA, ML, CL, TRILOGY (funkcionálne), PROLOG (logický).

1.2 Úvod do CL

V tomto semestri sa budeme učiť programovať a osvojovať si základné (programovacie) techniky funkcionálneho programovania v jazyku CL, ktorého autormi sú doc. Voda a Ing. Komara. Jazyk CL je jednoduchý ľahko naučiteľný elegantný pedagogicky vhodný predstaviteľ funkcionálneho programovania. Jeho syntax priamo vychádza z matematických definícií funkcií. V jazyku CL budeme programovať iba funkcie nad \mathbb{N} (čo je pre informatiku úplne postačujúce).

1.2.1 Matematické definície funkcií. Matematické definície funkcií môžeme rozdeliť na explicitné a na induktívne (rekurzívne). Explicitnú definíciu funkcie môžeme schématicky zapísať nasledovne:

$$F(\bar{a}) = \begin{cases} V_1 & \text{if } \text{cond}_1(\bar{a}) \\ \vdots & \vdots \\ V_n & \text{if } \text{cond}_n(\bar{a}), \end{cases}$$

kde $F(\cdot)$ sa už nevyskytuje vo V_i a $\text{cond}_i(\bar{a})$. Ak chceme definovať funkciu F nad nejakou množinou, doménou D , aby definícia bola naozaj korektná, musí spĺňať dve vlastnosti.

1. **Výlučnosť:** pre ľubovoľné argumenty \bar{a} existuje najviac jeden riadok s podmienkou $\text{cond}_i(\bar{a})$ splnenou.
2. **Úplnosť:** pre ľubovoľné argumenty \bar{a} existuje aspoň jeden riadok s podmienkou $\text{cond}_i(\bar{a})$ splnenou.

Čiže dokopy: pre ľubovoľné argumenty \bar{a} existuje práve jeden riadok s podmienkou $\text{cond}_i(\bar{a})$ splnenou. Z výlučnosti a úplnosti sa dá dokázať, že existuje práve jedna funkcia F nad D , ktorá spĺňa danú definíciu. Platí, že

$$\left. \begin{array}{l} \text{z úplnosti} \Rightarrow \text{jednoznačnosť} \\ \text{z výlučnosti} \Rightarrow \text{existencia} \end{array} \right\} \text{ funkcie.}$$

1.2.2 Nesprávne príklady. Teraz si uvedieme nekorektné definície funkcií:
Neúplna definícia:

$$F(\bar{a}) = \begin{cases} 1 & \text{if } x > 3 \\ 2 & \text{if } x < 3. \end{cases}$$

Pre $x = 3$ nemá určenú hodnotu, čiže nekoňečne veľa funkcií tvaru

$$F(0) = f(1) = f(2) = 2$$

$$F(3) = n \in \mathbb{N}$$

$$F(x) = 1 \text{ pre } x > 3$$

vyhovuje definícii.

Nevýlučná definícia:

$$F(\bar{a}) = \begin{cases} 1 & \text{if } x \geq 3 \\ 2 & \text{if } x \leq 3. \end{cases}$$

Platí, že $F(3) = 1 \neq 2 = F(3)$, teda vyhovujúca funkcia neexistuje.

1.2.3 Úvod do syntaxe jazyka CL. Teraz si spravíme malý úvod do syntaxe jazyka CL, aby sme vedeli písať (programovať) jednoduché explicitné definície. Podobne ako pri explicitných matematických definíciách, aj explicitná CL-definícia sa skladá z niekoľkých riadkov (odpovedajú riadkom matematickej definície) nazývaných *klauzulami*. Klauzula má tvar

$$F(a_1, \dots, a_k) = v \leftarrow t_1 \text{Rel}_1 t_2 \ \& \ \dots \ \& \ t_{2n-1} \text{Rel}_n t_{2n}$$

(v matematickom zápise namiesto $\&$ používame \wedge), kde

1. F je identifikátor definovanej funkcie, alfa-numerický reťazec začínajúci sa s veľkým písmenom. Vo vnútri reťazca nie je už dovolené veľké písmeno, ale môžeme použiť $_$ (napríklad: Max_3);
2. a -čka, v -čko, t -čka sú termy (ako v logike) definované indukčne:
Term je
 - (a) premenná (reťazec z malých písmien, môže sa končiť jedno alebo dvojčíferným indexom $x1$, $x99$, $x0 \equiv x$, $x00 \equiv x$),
 - (b) číslo - prirodzené decimálne (konštanta),
 - (c) výraz $F(t_1, \dots, t_n)$ (kde F je identifikátor funkcie a t -čka sú termy).

V CL-ku sú zabudované aritmetické relácie:

$$\begin{array}{ll} = & \neq < > \leq \geq & \text{matematický zápis} \\ = & != < > <= >= & \text{ASCII zápis v CL-ku.} \end{array}$$

Ďalej v CL-ku sú zabudované nasledujúce aritmetické funkcie, ktoré budeme používať (sú binárne, preto sa dajú písať infixne):

$$\begin{array}{ll} \text{násobenie } x \cdot y & \text{ASCII } * \\ \text{sčítanie } x + y & \text{ASCII } +, \end{array}$$

modifikované odčítanie

$$x \div y = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{if } x < y \end{cases} \quad \text{ASCII } -$$

(klasické odčítanie nie je uzavreté na \mathbb{N}),
celočíselné delenie a zvyšková funkcia

$$\begin{array}{ll} x \div y & \text{ASCII } x/y \\ x \bmod y & \text{ASCII } x \bmod y, \end{array}$$

ktoré spĺňajú

$$\begin{array}{l} y > 0 \rightarrow x = (x \div y) \cdot y + x \bmod y \wedge x \bmod y < y \\ x \div 0 = x \quad \bmod 0 = 0. \end{array}$$

1.2.4 Príklady explicitne definovaných funkcií nad \mathbb{N} a ich zápis v jazyku CL.

Druhá mocnina.

$$\text{Power}(x) = x \cdot x.$$

V CL to isté.

Maximum.

$$\text{max}(x, y) = \begin{cases} x & \text{if } x > y \\ y & \text{if } x \leq y \end{cases}.$$

V CL:

$$\text{max}(x, y) = x \leftarrow x > y$$

$$\text{max}(x, y) = y \leftarrow x \leq y \quad ,$$

zápis v **if_then_else** forme:

$$\text{max}(x, y) = \text{if } x > y \text{ then } x \\ \text{else } y \quad .$$

Funkcia F1.

$$F1(x, y) = \begin{cases} 1 & \text{if } x < y \\ 2 & \text{if } x = y \\ 3 & \text{if } x > y \end{cases}.$$

V jazyku CL:

$$F1(x, y) = 1 \leftarrow x < y$$

$$F1(x, y) = 2 \leftarrow x = y$$

$$F1(x, y) = 3 \leftarrow x > y \quad ,$$

zápis v **if_then_else** forme:

$$F1(x, y) = \text{if } x < y \text{ then } 1 \\ \text{else if } x = y \text{ then } 2 \\ \text{else } 3 \quad .$$

Funkcia F2.

$$F2(x, y) = \begin{cases} 1 & \text{if } x = y \\ 2 & \text{if } x \neq y \end{cases}.$$

V jazyku CL:

$$F2(x, y) = 1 \leftarrow x = y$$

$$F2(x, y) = 2 \leftarrow x \neq y \quad ,$$

zápis v **if_then_else** forme:

$$F2(x,y) = \mathbf{if} \ x = y \ \mathbf{then} \ 1 \\ \mathbf{else} \ 2 .$$

Funkcia F3.

$$F3(x, y, z) = \begin{cases} 1 & \text{if } x < y \wedge y = z \\ 2 & \text{if } x < y \wedge y \neq z \\ 3 & \text{if } x \geq y \wedge y \leq z \\ 4 & \text{if } x \geq y \wedge y > z \end{cases} .$$

V jazyku CL:

$$\begin{aligned} F3(x, y, z) = 1 &\rightarrow x < y \wedge y = z \\ F3(x, y, z) = 2 &\rightarrow x < y \wedge y \neq z \\ F3(x, y, z) = 3 &\rightarrow x \geq y \wedge y \leq z \\ F3(x, y, z) = 4 &\rightarrow x \geq y \wedge y > z \end{aligned} ,$$

zápis v **if_then_else** forme:

$$F3(x,y) = \mathbf{if} \ x < y \ \mathbf{then} \ \mathbf{if} \ y = z \ \mathbf{then} \ 1 \ \mathbf{else} \ 2 \\ \mathbf{else} \ \mathbf{if} \ y \leq z \ \mathbf{then} \ 3 \ \mathbf{else} \ 4 .$$

1.2.5 Výpočet v CL. Neformálne, výpočet v CL prebieha: zľava-doprava po stĺpcoch. Dosadíme za premenné argumenty, skočíme za \leftarrow a v stĺpcoch vyselektujeme odpovedajúci riadok, ktorého hodnotu (hodnota termu za $=$) priradíme danej aplikovanej funkcii na zadané argumenty. V jazyku CL chceme dovoliť iba výlučné a úplné explicitné definície. Dokazovať výlučnosť a úplnosť definície však môže byť netriviálny problém. Pri výlučnosti kompilátor dokáže rozpoznávať iba syntakticky zrejmé prípady. Tento proces sa volá diskriminácia. Na úvod si uvedieme diskrimináciu pomocou aritmetických relácií.

1. Definícia musí mať rovnaký začiatok $F(\bar{a})$ vo všetkých riadkoch;
2. za \leftarrow máme stĺpce tvaru:

$$\begin{array}{ccc} & \text{dichotómia} & \text{richotómia} \\ x = y & x < y & x < y \\ x \neq y & x \geq y & x = y \\ & & x > y \\ & \text{analogicky} & \\ & x > y & \\ & x \leq y & ; \end{array}$$

3. riadky v stĺpci sa môžu opakovať, permutovať (počítame zľava do prava po stĺpcoch, preto na poradí riadkov nezáleží);

4. pre vnorený podprípád začiatok z musí byť rovnaký, napríklad:

$$\begin{aligned} &\leftarrow z \wedge r_1 \\ &\leftarrow z \wedge r_2 \\ &\leftarrow z \wedge r_3. \end{aligned}$$

Úplnosť definície sa zabezpečí oveľa jednoduchšie. Ak sa pri vyhodnotení nevyskytuje žiadny riadok defaultovo vezmeme 0 ako hodnotu funkcie pre dané argumenty. Implicitne predpokladáme, že v definícii je riadok tvaru $F(\cdot) = 0 \leftarrow$ otherwise.

Príklady:

(Neúplná trichotómia)

$$F(x) = 1 \leftarrow x = 5$$

$$F(x) = 2 \leftarrow x > 5$$

pre $x < 5$ $F(x) = 0$, alebo

$$F(x) = 10 \leftarrow x = 5$$

pre $x \neq 5$ $F(x) = 0$.

Kapitola 2

Induktívne definície

Teraz si povieme niečo o induktívnych (rekurzívnych) definíciách. V matematike ste sa neraz stretli napríklad s nasledovnými induktívnymi definíciami:

- mocnina s prirodzeným exponentom:

$$\begin{aligned}x^0 &= 1 \\x^{n+1} &= x \cdot x^n\end{aligned}$$

- faktorial:

$$\begin{aligned}0! &= 1 \\(n+1)! &= (n+1) \cdot n!\end{aligned}$$

- Fibonacciho funkcia:

$$\begin{aligned}F(0) &= 0 \\F(1) &= 1 \\F(n+2) &= F(n+1) + F(n).\end{aligned}$$

Všeobecne induktívne definície funkcií môžeme schématicky zapísať nasledovne:

$$F(\bar{a}) = \begin{cases} V_1 & \text{cond}_1(\bar{a}) \\ \vdots & \vdots \\ V_n & \text{cond}_n(\bar{a}), \end{cases}$$

čo sa veľmi podobá na schému explicitných definícií. Rozdiel je v tom, že táto schéma je všeobecnejšia, totiž umožňuje, aby sa term tvaru $F(\cdot)$ vyskytoval aj vo V -čkách a $\text{cond}_i(\bar{a})$. Opäť chceme, aby naša definícia nám korektne definovala jednu funkciu F nad doménou D . K tomu budeme potrebovať, aby definíciu okrem výlučnosti a úplnosti spĺňala i podmienku *regularity*: existuje funkcia - miera z $D^{\text{ar}(F)} \rightarrow \mathbb{N}$, taká, že pre ľubovoľnú aplikáciu funkcie $F - F(\bar{b})$ z pravej strany definície (za svorkou) vyskytujúcej sa vo V -čkách či $\text{cond}_i(\bar{a})$, musí platiť, že $m(\bar{a}) > m(\bar{b})$.

2.0.6 Kontrapríklady. Uvažujme nasledovnú definíciu: $F(x) = F(x)$ funkcie nad \mathbb{N} . Tá je zrejme úplná a výlučná, ale nie regulárna, lebo nemôže existovať taká funkcia-miera m , že $m(x) > m(x)$. Definíciu vyhovuje ľubovoľná funkcia nad \mathbb{N} , čiže definícia neurčuje jednoznačne funkciu.

Podobne definícia: $F(x) = F(x) + 1$ je úplná a výlučná, ale nie je regulárna, lebo opäť nemôže existovať miera m , že $m(x) > m(x)$. Ďalej pre žiadnu funkciu nemôže platiť, že $F(x) = F(x) + 1$, z toho vyplýva, že daná definícia neurčuje žiadnu funkciu.

Summa summarum, ak porušíme podmienku regularity, môže sa stať že neexistuje žiadna funkcia vyhovujúca definícii alebo viaceru rôznych funkcií jej vyhovuje.

Na vytváranie induktívnych definícií nám budú postačovať znalosti z CL-syntaxe uvedené pri explicitných definíciách. Okrem diskriminácie pomocou aritmetických výrazov budeme používať ďalší typ klauzálnych definícií využívajúcich diskrimináciu na prirodzených číslach. V hlavách klauzúl sa bude vyskytovať stĺpec tvaru:

$$\begin{array}{c} 0 \\ 1 \\ 2 \\ \vdots \\ k \\ n + (k + 1), \end{array}$$

kde $k \in \mathbb{N}$ a argumenty nachádzajúce sa naľavo od tohoto stĺpca (v hlavě klauzuly) musia byť vo všetkých riadkoch-klauzulách identické.

V ďalšom texte už budeme uvádzať iba CL-definície nakoľko sú veľmi podobné matematickým a ich vzájomný prepis je priamočiary.

2.1 Príklady

Ako vyzerá výpočet pomocou induktívnych, rekurzívnych definícií? Minule sme si niečo povedali o výpočte pomocou explicitných definícií. Teraz si naše intuitívne znalosti rozšírime. Mohli by sme charakterizovať výpočet vo funkcionálnom programovaní ako vyhodnocovanie výrazov - technickejšie ako prepisovanie (redukcia) výrazov (termov) do nejakých základných, ireducibilných, ďalej sa už neprepisujúcich tvarov. Uvažujme nasledujúce jednoduché príklady:

2.1.1 Mocnina.

$$\begin{array}{l} x^0 = 1 \\ x^{n+1} = x \cdot x^n \quad \text{pre } (n \geq 0) \in \mathbb{N}. \end{array}$$

Ako by sme ju spravili v Pascali pomocou cyklov?

Možeme ju naprogramovať napríklad pomocou **while**-cyklu:

```
p := 1;
i := n;
{ invariant } ←
while i > 0 do
  begin
    p := x*p;
    i := i-1
  end.
```

Tento cyklus nám reprezentuje jeden typ výpočtu mocniny. Klesajúci parameter cyklu je i , testuje sa či je $i > 0$, po každom zbehnutí tela cyklu sa nám zníži o jedna; z toho vyplýva, že cyklus skončí. Invariant cyklu (podmienka nemeniaca sa počas cyklu, platí na začiatku aj na konci cyklu v bode \leftarrow) bude:

$$x^n = p \cdot x^i \wedge i \geq 0.$$

1. Na začiatku máme $i = n \geq 0$, $x^n = 1 \cdot x^n$.
2. Ak cyklus zbehol, tak muselo $i > 0$. Vieme, že pre $i > 0$ platí $x^n = p \cdot x \cdot x^{i-1} = \bar{p} \cdot x^{\bar{i}}$. Súčasne $\bar{i} \geq 0$. Čiže invariant platí aj pre nové hodnoty \bar{p} a \bar{i} po vykonaní cyklu.
3. Na konci máme $i = 0$, $x^n = p \cdot x^0 = p \cdot 1 = p$.

Z platnosti invariantu teda vidíme, že cyklus naozaj korektne vypočíta mocninu do premennej p .

CL-definícia:

$$\begin{aligned} \text{Power}(x, 0) &= 1 \\ \text{Power}(x, n + 1) &= x \cdot \text{Power}(x, n) \end{aligned}$$

Výpočet (neformálne) pre $x = 2$; $n = 5$ odsimulujeme nasledovne:

$$\begin{aligned} \text{Power}(2, 5) &= 2 \cdot \text{Power}(2, 4) = 2 \cdot 2 \cdot \text{Power}(2, 3) = 2 \cdot 2 \cdot 2 \cdot \text{Power}(2, 2) \\ &= 2 \cdot 2 \cdot 2 \cdot 2 \cdot \text{Power}(2, 1) = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot \text{Power}(2, 0) \\ &= 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 1 = 32; \end{aligned}$$

1. dosadíme za premenné
2. vyselektujeme (pomocou diskriminácii) vhodný riadok
3. prepíšeme výraz výrazom na prvej strane = vybratého riadku.

V CL-ku môžeme spraviť nasledovnú definíciu mocniny, pomocou ktorej môžeme mať podobný výpočet ako pri **while**-cykle:

začiatok	po 1.zbehu	po 2.zbehu	3.zbehu	4.zbehu	5.zbehu
$p = 1$	2	4	8	16	32
$i = 5$	4	3	2	1	0

Obrázok 2.1: Výpočet pre $x = 2$ a $n = 5$.

$$\begin{aligned} Power(x, i, m) &= Pow1(x, n, 1) \\ Pow1(x, 0, m) &= m \\ Pow1(x, n + 1, m) &= Pow1(x, i, x * m) \end{aligned}$$

Výpočet pre $x = 2$, $n = 5$:

$$\begin{aligned} Power(2, 5) &= Pow1(2, 5, 1) = Pow1(2, 4, 2) \\ &= Pow1(2, 3, 4) = Pow1(2, 2, 8) \\ &= Pow1(2, 1, 16) = Pow1(2, 0, 32) = 32. \end{aligned}$$

1. Vidíme, že výpočty sú analogické - rovnaká modifikácia i , p pri **while**-cykle a 2., 3. argumentu pri $Pow1$.
2. 3. argument, m , je miesto, kam si odkladáme medzivýsledky, takzvaný akumulátor, preto sa takáto technika nazýva technikou akumulátorovej premennej.
3. Tento typ rekurzie, keď po vyhodnotení rekurzívneho volania už nemusíme nič dovyhodnocovať v žiadnom prípade, nazveme chvostová rekurzia - *tail recursion*. Na rozdiel od prvej definície, kde sme museli ešte donásobovať dvoma, a ktorá je čisto rekurzívna.

Kompilátor dokáže rozpoznať, či ide o tail rekurziu a vtedy ju implementuje pomocou cyklu (efektívne). Čiže i vo funkcionálnom programovaní môže byť výpočet dlhý, neefektívny alebo naopak krátky a efektívny. Záleží na zručnosti programátora.

Do tretice pomocou **for**-cyklu:

```
p:=1;
for i=1 to n do p := x*p
```

Cyklus **for** vždy skončí, ak $n \geq 0$ uskutoční sa n -krát. Klesajúci parameter je $n - i + 1$. Pre $n \geq 0$ platí invariant $x^n = p \cdot x^{n-i+1}$:

1. Na začiatku máme $i = 1$, $p = 1$. Platí $x^n = 1 \cdot x^{n-1+1}$.
2. Pre $i \leq n$ dostávame $x^n = p \cdot x \cdot x^{n-i-1+1} = \bar{p} \cdot x^{n-\bar{i}+1}$.
3. Na konci máme $i = n + 1$, $p := x^n$. Teda $x^n = x^n \cdot x^{n-(n+1)+1} = x^n \cdot x^0 = x^n$

začiatok	po 1.zbehu	po 2.zbehu	3.zbehu	4.zbehu	5.zbehu
$p = 1$	2	4	8	16	32
$i = 1$	2	3	4	5	6

Obrázok 2.2: Výpočet pre $x = 2$ a $n = 5$.

$$Power(x, n) = Pow2(x, 1, n, 1)$$

$$Pow2(x, i, n, m) = m \leftarrow i > n$$

$$Pow2(x, i, n, m) = Pow2(x, i + 1, n, x \cdot m) \leftarrow i \leq n$$

Výpočet pre $x = 2$, $n = 5$:

$$\begin{aligned} Power(2, 5) &= Pow2(2, 1, 5, 1) = Pow2(2, 2, 5, 2) \\ &= Pow2(2, 3, 5, 4) = Pow2(2, 4, 5, 8) \\ &= Pow2(2, 6, 5, 32) = 32 . \end{aligned}$$

- Opäť sa využíva technika akumulátorovej premennej, navyše máme argument pre n - 'hornú hranicu cyklu'.
- Definícia je tail - rekurzívna. Kompilátor ju nahradí cyklom.
- Výpočet je analogický ako výpočet pomocou **for** cyklu. Nasledovné premenne si odpovedajú:
 - i , p vo **for** cykle a
 - i , $m - 2$. a 4. argument v $Pow2$.

2.1.2 Najväčší spoločný deliteľ. Nech $x, y \in \mathbb{N}$ chceme nájsť najväčšieho spoločného deliteľa z , kde z má spĺňať podmienku:

$$z|x \wedge z|y \wedge \forall d (d|x \wedge d|y \rightarrow d \leq z).$$

V prípade, že $x + y > 0$ ($x > 0$ alebo $y > 0$), z je určené jednoznačne. Pre prípad $x + y = 0$, pre $\forall z \in \mathbb{N}$ platí $z|0$, čiže najväčšie z s touto vlastnosťou neexistuje, preto defnitoricky položíme $Gcd(0, 0) = 0$.

Našou úlohou bude teda nájsť funkciu $Gcd(x, y)$, ktorá bude spĺňať nasledovnú špecifikáciu:

$$\begin{aligned} \forall x \forall y (x + y > 0 \rightarrow Gcd(x, y)|x \wedge Gcd(x, y)|y \wedge \\ \forall d [d|x \wedge d|y \rightarrow d \leq Gcd(x, y)]). \end{aligned}$$

Určili sme si, čo chceme počítať. Ako to počítať?

začiatok	po 1.zbehu	po 2.zbehu	3.zbehu	4.zbehu
$x_i = 12$	21	12	9	3
$y_i = 21$	12	9	3	0
$z_i = ?$	12	9	3	0

Obrázok 2.3: Výpočet pre $x = 12$ a $y = 21$.

Napríklad pomocou nasledovnej verzie Euklidovho algoritmu:

```

xi := x;
yi := y;
while yi > 0 do
  begin
    zi := xi mod yi;
    xi := yi;
    yi := zi
  end
gcd := xi;

```

Klesajúci parameter cyklu bude y_i (≥ 0).

1. Testujeme či $y_i > 0$.
2. Vieme, že nová hodnota y_i bude $x_i \bmod y_i <$ ako stará hodnota y_i . Čiže hodnota y_i sa zmenší po každom prechode cyklom. Cyklus nám skončí, vtedy $y_i = 0$.

Ak $x + y > 0$, invariant bude $gcd(x, y) = gcd(x_i, y_i)$.

1. Na začiatku máme $x = x_i$, $y = y_i$.
2. Vieme, že pre $y_i > 0$ platí:

$$gcd(x, y) = gcd(x_i, y_i)$$

$$gcd(y_i, x_i \bmod y_i) = gcd(\bar{x}_i, \bar{y}_i).$$

3. Na konci $y_i = 0$, čiže $gcd(x, y) = gcd(x_i, 0) = x_i = gcd$.

Ak $x + y = 0$ tak $gcd = 0$ (ako sme sa dohodli). Cyklus ani raz nezbehne.

Ako definovať v CL-ku Gcd ?

$$Gcd(x, 0) = x$$

$$Gcd(x, y + 1) = Gcd(y + 1, x \bmod (y + 1)) .$$

Definícia je tail - rekurzívna. Všimnime si analogický výpočet (nepotrebujeme pomocnú premennú z_i) pre $x = 12$, $y = 21$:

$$Gcd(12, 21) = Gcd(21, 12) = Gcd(12, 9) = Gcd(9, 3) = Gcd(3, 0) = 3.$$

Poznamenajme, že sme použili **ten istý** jazyk logiky na zápis špecifikácie (čo robiť) aj definície funkcie (ako to robiť). Z toho tiež vyplýva, že naše programy sú matematické.

2.1.3 Fibonacciho funkcia. Uvažujme Fibonacciho postupnosť:

$$0 \ 1 \ 1 \ 2 \ 3 \ 5 \ 8 \ 13 \ 21 \ 34 \ \dots$$

Z faktu, že nasledujúci člen postupnosti je súčtom predchádzajúcich dvoch, môžeme ihneď napísať rekurzívnu definíciu:

$$\begin{aligned} Fib(0) &= 0 \\ Fib(1) &= 1 \\ Fib(n+2) &= Fib(n+1) + Fib(n). \end{aligned}$$

Skúsme počítať pre $Fib(5)$:

$$\begin{aligned} Fib(5) &= Fib(4) + Fib(3) \\ &= Fib(3) + Fib(2) + Fib(2) + Fib(1) \\ &= Fib(2) + Fib(1) + Fib(1) + Fib(0) + Fib(1) + Fib(0) + 1 \\ &= Fib(1) + Fib(0) + 1 + 1 + 0 + 1 + 0 + 1 \\ &= 1 + 0 + 1 + 1 + 0 + 1 + 0 + 1 = 5. \end{aligned}$$

Dostávame nešikovný dlhý výpočet. Približne 2^n (pre $Fib(5)$ 8 volaní) rekurzívnych volaní treba pre výpočet $Fib(n)$. Skúsme to teda šikovnejšie, stačí nám pamätať si v pomocných premenných (v akumulátoroch) hodnoty pre dva predchádzajúce prípady a z nich vypočítať nový člen postupnosti. Dostávame definíciu:

$$\begin{aligned} Fiba(0, a, b) &= a \\ Fib(n) &= Fiba(n, 0, 1) \\ Fiba(n+1, a, b) &= Fiba(n, a+b, a) \quad . \end{aligned}$$

Definícia je tail - rekurzívna. Na porovnanie si ukážeme tiež výpočet $Fib(5)$.

$$\begin{aligned} Fib(5) &= Fiba(5, 0, 1) = Fiba(4, 1, 0) \\ &= Fiba(3, 1, 1) = Fiba(2, 2, 1) \\ &= Fiba(1, 3, 2) = Fiba(0, 5, 3) = 5. \end{aligned}$$

Vidíme, že je omnoho kratší. Pre ilustráciu si skúsme túto tail - rekurzívnu definíciu naprogramovať (odsimulovať) pomocou **while**-cyklu (predpokladáme, že $n \geq 0$).

```
a := 0;
b := 1;
i := n;
while i > 0 do
  begin
    c := a+b;
    b := a;
    a := c;
    i := i-1;
  end
fib := a;
```

začiatok	po 1.zbehu	po 2.zbehu	3.zbehu	4.zbehu	5.zbehu
a = 1	1	2	3	3	5
b = 1	0	1	1	2	3
c = ?	1	1	2	3	5
i = 5	4	3	2	1	0

Obrázok 2.4: Výpočet pre $n = 5$.

1. Klesajúci parameter je i ; testujeme, či $i > 0$; v cykle dekrementujeme $i := i - 1$; čiže cyklus vždy skončí.
2. Pre $n = 0$ platí $fib = Fib(0)$.
3. Pre $n > 0$ máme invariant

$$\begin{aligned} Fib(n - i) &= a \\ Fib(n - i - 1) &= b \end{aligned}$$

- (a) Na začiatku pre $i = n - 1$ dostávame

$$\begin{aligned} Fib(n - n + 1) &= Fib(1) = 1 = a \\ Fib(n - n) &= Fib(0) = 0 = b. \end{aligned}$$

Po vykonaní cyklu máme:

$$\begin{aligned} Fib(n - \bar{i}) &= Fib(n - i + 1) = Fib(n - i) + Fib(n - i - 1) \\ &= a + b = \bar{a} \\ Fib(n - \bar{i} - 1) &= Fib(n - i) = a = \bar{b}. \end{aligned}$$

- (b) Na konci platí, že

$$\begin{aligned} Fib(n - 0) &= Fib(n) = a = fib \\ Fib(n - 1) &= b. \end{aligned}$$

Výpočet pre $n = 5$ najdeme v tabuľke 2.4.

Teraz si skúsime verziu Fibonacciho funkcie pomocou **for**-cyklu.

```

if n=0 then fib := 0
else if n=1 then fib := 1
else
  begin
    a := 1;
    b := 0;
    for i=2 to n do
      begin
        c := a+b;
        b := a;
        a := c;
      end
    fib := a
  end

```

Overme si korektnosť nášho programu. Pre

$$\begin{aligned} n = 0 & \quad fib = 0 = Fib(0) \\ n = 1 & \quad fib = 1 = Fib(1). \end{aligned}$$

Pre $n \geq 2$ cyklus zbehne $n - 2 + 1$ -krát (klesajúci parameter je $n - i + 1$). Platí invariant

$$\begin{aligned} Fib(i - 1) &= a \\ Fib(i - 2) &= b. \end{aligned}$$

1. Na začiatku máme

$$\begin{aligned} Fib(2 - 1) &= Fib(1) = 1 = a \\ Fib(2 - 2) &= Fib(0) = 0 = b. \end{aligned}$$

2. Po zbehnutí cyklu dostávame:

$$\begin{aligned} Fib(\bar{i} - 1) &= Fib(i) = Fib(i - 1) + Fib(i - 2) \\ &= a + b = \bar{a} \\ Fib(\bar{i} - 2) &= Fib(i - 1) = a = \bar{b}. \end{aligned}$$

3. Na konci $i = n + 1$ a platí, že

$$\begin{aligned} Fib(n + 1 - 1) &= Fib(n) = a = fib \\ Fib(n + 1 - 2) &= Fib(n - 1) = b. \end{aligned}$$

Čiže náš program počíta Fibonacciho funkciu korektne.

začiatok	po 1.zbehu	po 2.zbehu	3.zbehu	4.zbehu
a = 1	1	2	3	5
b = 0	1	1	2	3
c = ?	1	2	3	5
i = 2	3	4	5	6

Obrázok 2.5: Výpočet pre $n = 5$.

V CL-ku môžeme simulovať tento výpočet nasledovne:

$$Fib(0) = 0$$

$$Fib(1) = 1$$

$$Fib(n + 2) = Fib2(2, n + 2, 1, 0)$$

$$Fib2(i, n, a, b) = a \leftarrow i > n$$

$$Fib2(i, n, a, b) = Fib2(i + 1, n, a + b, a) \leftarrow i \leq n \quad .$$

1. Definícia je tail - rekurzívna.
2. Netreba pomocnú premennú c .
3. Medzivýsledky sa ukladajú do akumulátorov $a = Fib(i-1)$, $b = Fib(i-2)$.

Pre porovnanie si ukážeme tiež výpočet pre $Fib(5)$:

$$\begin{aligned}
 Fib(5) &= Fib2(2, 5, 1, 0) = Fib2(3, 5, 1, 1) \\
 &= Fib2(4, 5, 2, 1) = Fib2(5, 5, 3, 2) \\
 &= Fib2(6, 5, 5, 3) = 5.
 \end{aligned}$$

Kapitola 3

Číselné reprezentácie

V tejto kapitole si zrekapitulujeme naše znalosti o číselných sústavách, ktoré poznáme zo základnej a strednej školy, z informatického hľadiska. Načrtne si implementáciu veľkej aritmetiky v rozličných číselných reprezentáciach. Ďalej sa zoznámime s p -adickými číselnými sústavami, kde našu pozornosť sústredíme na diadickú sústavu a jej reprezentáciu pomocou špeciálnych termov - numerálov. Nakoniec si rozoberieme implementáciu diadickej sústavy v jazyku CL.

3.1 Unárna sústava

Unárna sústava je asi prvá číselná sústava, s ktorou ste sa zoznámili v detstve (počítanie na prstoch, guľičky, čiarky). Prázdny reťazec čiarok (reprezentujúci

čísla	unárny zápis
0	\emptyset
1	
2	
3	
\vdots	\vdots

Obrázok 3.1: Unárna sústava.

0) v nej označíme ako \emptyset (pozri Obrázok 3.1). Čo je jej plus? Je jednoduchá - ľahko sa pričíta a odčíta jednotka.

3.1.1 Pričítanie a odčítanie jednotky. Dostávame nasledovné jednoduché klauzálne definície:

$$\text{Successor}(X) = X|$$

$Predecessor(\emptyset) = \emptyset$ (definitóricky)
 $Predecessor(X|) = X$,

Možeme ich zapísať i v tablovej verzii:

$$\frac{(+1) X}{X|} \quad \frac{(-1) \emptyset}{\emptyset} \quad \frac{(-1) X|}{X} .$$

Vzor (pattern) $X|$ naozaj matematicky znamená $X + 1$. Nakoniec si ukážme, že aj definície sčítania a odčítania nie sú oveľa zložitejšie.

3.1.2 Sčítanie.

$Addition(\emptyset, Y) = Y$
 $Addition(X|, Y) = Addition(X, Y|)$
alebo
 $Addition(X|, Y) = Addition(X, Y)|$,

tablová verzia:

$$\frac{\emptyset}{+ Y} \quad \frac{X|}{+ Y} \quad \frac{X|}{+ Y} .$$

3.1.3 Odčítanie.

$Subtraction(\emptyset, Y) = \emptyset$ (definitóricky)
 $Subtraction(X|, \emptyset) = X|$
 $Subtraction(X|, Y|) = Subtraction(X, Y)$,

tablová verzia:

$$\frac{\emptyset}{- Y} \quad \frac{X|}{- \emptyset} \quad \frac{X|}{- Y|} .$$

Zápis čísel v unárnej sústave má však aj svoje nevýhody. Dĺžka zápisu zodpovedá veľkosti čísla (napríklad na číslo 5 potrebujeme ||||| čiarok, znakov). Tiež rekúzia je typu $x + 1 \rightarrow x$, počet rekúzivných volaní zhruba odpovedá veľkosti čísla x . Preto sa používajú iné číselné sústavy, ktoré umožňujú 'ekonomickejší' (kratší zápis) a tiež rýchlejšiu rekúziu (menší počet rekúzivných volaní, miera rýchlejšie klesá).

3.2 Decimálna sústava

Ďalšou číselnou sústavou, s ktorou ste sa zoznámili v škole, bola decimálna (desiatková) sústava. Používa 10 cifier (znakov) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Číslo c v nej vyjadríme ako polynóm 10-vých mocnín

$$c = k_n 10^n + k_{n-1} 10^{n-1} + \dots + k_0 10^0$$

a pozične ho napíšeme ako k_n, \dots, k_0 . Zápis je výrazne kratší. Napríklad pre číslo tisíc - 1000 namiesto tisíc čiarok stačia 4 znaky, čo zhruba odpovedá $\log_{10} 10^3 = 3$; pre milión - 1000000 ($\log_{10} 10^6 = 6$) použijeme iba 7 znakov. Približne potrebujeme $\log_{10} c$ znakov na zápis čísla c , dokážeme ho teda zapísať v logaritmickú dĺžku. V prípade unárnej sústavy sme pre číslo c potrebovali až c znakov, zapisovali sme ho v lineárnej dĺžke, rozdiel dĺžok je veľmi veľký.

Ako je to s pričítaním a odčítaním jednotky? Dostávame nasledujúce jednoduché definície:

3.2.1 Pričítanie jednotky.

$$\begin{aligned} \text{Successor}(\emptyset) &= 1 \\ \text{Successor}(X0) &= X1 \\ \text{Successor}(X1) &= X2 \\ &\vdots \\ \text{Successor}(X9) &= \text{Successor}(X)0, \end{aligned}$$

tablová verzia:

$$\frac{(+1) \emptyset}{1} \quad \frac{(+1) X0}{X1} \quad \dots \quad \frac{(+1) X9}{((+1) X)0}.$$

3.2.2 Odčítanie jednotky.

$$\begin{aligned} \text{Predecessor}(\emptyset) &= \emptyset \quad (\text{definitorky}) \\ \text{Predecessor}(X0) &= \text{Predecessor}(X)9 \leftarrow X \neq \emptyset \\ \text{Predecessor} = 0 &\leftarrow X = \emptyset \\ \text{Predecessor}(X1) &= X0 \\ &\vdots \\ \text{Predecessor}(X9) &= X8, \end{aligned}$$

tablová verzia:

$$\frac{(-1) \emptyset}{\emptyset} \quad \frac{(-1) X0 (X \neq \emptyset)}{((-1) X)9} \quad \frac{(-1) X0 (X = \emptyset)}{0} \quad \frac{(-1) X1}{X0} \quad \dots \quad \frac{(-1) X9}{X8}.$$

Predchádzajúca definícia však v sebe skrýva problém, funguje, iba ak zápis neobsahuje vľavo neplatné nuly, ale sama túto zásadu nedodrží. Napríklad odčítaním jednotky od zápisu 10 čísla 10 dostaneme zápis 09, vygeneruje sa vľavo neplatná 0. Odčítaním jednotky od zápisu 00 čísla 0 získame dokonca chybný výsledok 09! Pri p-adických sústavách si ukážeme správne riešenie.

Pokúsme sa schématicky si zdefinovať sčítanie, odčítanie a násobenie.

3.2.3 Sčítanie.

$$\frac{\emptyset}{+Y} \quad \frac{X0}{(X+Y)0} \quad \dots \quad \frac{X9}{(X+Y+1)8}.$$

$$\text{Například: } \begin{array}{r} 347 \\ + 589 \\ \hline 936 \end{array}.$$

3.2.4 Odčítanie.

$$\begin{array}{r} \emptyset \\ - Y \\ \hline \emptyset \end{array} \quad \begin{array}{r} X0 \\ - \emptyset \\ \hline X0 \end{array} \quad \begin{array}{r} X0 \\ - Y0 \\ \hline (X - Y)0 \end{array} \quad \cdots \quad \begin{array}{r} X0 \\ - Y9 \\ \hline (X - Y - 1)1 \end{array} \quad \cdots \quad \begin{array}{r} X9 \\ - Y9 \\ \hline (X - Y)0 \end{array}.$$

$$\text{Například: } \begin{array}{r} 589 \\ - 499 \\ \hline 90 \end{array}.$$

Keby sme chceli presnejšie formalizovať, vznikli by problémy ako pri funkcii *Predecessor* kvôli nejednoznačnému zápisu (neplatné nuly vľavo).

3.2.5 Násobenie.

Celková schéma:

$$\begin{array}{r} X \\ \cdot Yk \\ \hline X \cdot k \\ + X \cdot Y \cdot 10 \end{array}.$$

$$\text{Príklad: } \begin{array}{r} 45 \\ \cdot 93 \\ \hline 135 \\ + 4050 \\ \hline 4185 \end{array}.$$

Využívame násobenie X jednou cifrou:

$$\begin{array}{r} Xk_1 \\ \cdot k_2 \\ \hline (X \cdot k_2 + \text{prenos})(k_1 \cdot k_2) \end{array}.$$

V decimálnej sústave vidíme, že sa ľahko realizuje násobenie základom 10 (posuv doľava + pridanie 0 vpravo) a celočíselné delenie 10-mi (posuv doprava + odrezanie cifry napravo).

3.3 Binárna sústava

V počítačoch sa používa na kódovanie (reprezentáciu, zápis) čísel často binárna sústava (popri hexadecimálnej a oktálnej). Používame dve cifry 0, 1. Číslo c

číslo	binárna sústava
0	0
1	1
2	10
3	11
4	100
⋮	⋮

Obrázok 3.2: Binárna sústava.

vyjadríme ako polynóm 2-vých mocnín

$$c = k_n \cdot 2^n + k_{n-1} \cdot 2^{n-1} + \dots + k_0 \cdot 2^0$$

a pozične zapíšeme ako $k_n \dots k_0$ (pozri Obrázok 3.2). Dĺžka zápisu zodpovedá $\log_2 c$ (analogicky ako pri desiatkovej sústave i tu dosahujeme logaritmicnú dĺžku zápisu). Napríklad: $(1024)_{10} = (10000000000)_2$, dĺžka je 11 znakov, čo približne zodpovedá $\log_2 1024 = 10$.

Základné aritmetické operácie odvodíme analogicky ako pri desiatkovej sústave. Sú vlastne ich zjednodušením z desiatich cifier (prípadov) na iba dve cifry (prípady).

3.3.1 Successor.

$$\frac{(+1) \emptyset}{1} \quad \frac{(+1) X0}{X1} \quad \frac{(+1) X1}{((+1) X)0}$$

3.3.2 Predecessor. Taktiež, ako pri desiatkovej sústave, funguje iba ak zápis neobsahuje vľavo neplatné nuly a pritom sám ich vyrába.

$$\frac{(-1) \emptyset}{\emptyset} \quad \frac{(-1) X0 (X \neq \emptyset)}{((-1) X)1} \quad \frac{(-1) X0 (X = \emptyset)}{0} \quad \frac{(-1) X1}{X0}$$

3.3.3 Sčítanie.

$$\begin{array}{ccccc} \emptyset & X0 & X0 & X1 & X1 \\ +Y & +Y0 & +Y1 & +Y0 & +Y1 \\ \hline Y & (X+Y)0 & (X+Y)1 & (X+Y)1 & (X+Y+1)0 \end{array}$$

Príklad:
$$\begin{array}{r} 1011 \\ +1111 \\ \hline 11010 \end{array}$$

3.3.4 Odčítanie.

$$\begin{array}{r}
 \emptyset \quad X0 \qquad \qquad X0 \qquad \qquad \qquad X0 \quad X1 \qquad \qquad X1 \qquad \qquad X1 \\
 Y \quad \emptyset \quad \quad \quad - Y0 \quad \quad \quad \quad - Y1 \quad \quad \quad \emptyset \quad \quad \quad - Y0 \quad \quad \quad - Y1 \\
 \hline
 \emptyset \quad X0 \quad \quad \quad (X - Y)0 \quad \quad \quad (X - Y - 1)1 \quad \quad \quad X1 \quad \quad \quad (X - Y)1 \quad \quad \quad (X - Y)0
 \end{array}$$

$$\begin{array}{r}
 1110 \\
 \text{Príklad: } \frac{-1010}{100}
 \end{array}$$

Keby sme chceli presnejšie definovať odčítanie, znova by vznikli problémy kvôli nejednoznačnému zápisu (neplatné nuly vľavo).

3.3.5 Násobenie.

$$\begin{array}{r}
 X \qquad \qquad X \\
 \cdot Y0 \quad \quad \cdot Y1 \\
 \hline
 X \cdot Y0 \quad \quad X \\
 + X \cdot Y0
 \end{array}$$

$$\begin{array}{r}
 10 \\
 \text{Príklad: } \frac{\cdot 11}{10} \\
 + 100 \\
 \hline
 110
 \end{array}$$

Podobne ako pri desiatkovej sústave, i tu sa ľahko realizuje (v konštantnom čase) násobenie 2-mi (posuv doľava + prídanie 0 vpravo) a celočíselne delenie 2-mi (posuv doprava + odrezanie cifry napravo).

3.4 P-adické číselné sústavy

Nejednoznačnosť zápisu v n -árnych sústavách je určité ich mínus. Vyplýva z toho, že umožňujeme ako koeficienty pri mocninách používať 0. Preto sa nám v zápise čísla môžu vľavo hromadiť neplatné nuly:

$$\begin{aligned}
 0 &= 00_{10} = 000_2 \\
 1 &= 01_{10} = 001_2 \\
 2 &= 02_{10} = 002_{10} = 0010_2 .
 \end{aligned}$$

To nám môže spôsobovať problémy vytvoriť elegantné definície aritmetických funkcií.

Ako si pomôcť? Jednoducho, zakážeme koeficienty 0. Nech p je priradené číslo > 0 . Ľubovoľné číslo $c \in \mathbb{N}$ je buď 0 alebo ak $c > 0$, tak sa dá zapísať ako polynóm p mocnín: $c = k_n \cdot p^n + \dots + k_0 \cdot p^0$, kde $k_i \in \{1, \dots, p\}$. Pozične ho zapíšeme ako k_n, \dots, k_0 . Takúto číselnú sústavu nazveme p -adickou. Dĺžka

číslo	monadická
0	0
1	1
2	11
3	111
4	1111
⋮	⋮

Obrázok 3.3: Monodická sústava.

zápisu c je logaritmická, približne $\log_p c$. Ak $p = 1$, sústavu voláme *monadická* (pozri Obrázok 3.3). Odpovedá vlastne našej unárnej sústave. Len miesto čiarok | píšeme 1-ky. Nebudeme ju ďalej rozvádzať, nakoľko o nej platí presne to, čo sme si povedali o unárnej sústave.

3.5 Diadická sústava

číslo	diadická
0	0
1	1
2	2
3	11
4	12
5	21
6	22
7	111
8	112
9	121
10	122
11	211
12	212
13	221
14	222
15	1111
⋮	⋮

Obrázok 3.4: Diadická sústava.

V ďalšom výklade sa zameriame na prípad $p = 2$ - na *diadickú* sústavu (pozri

Obrázok 3.4). Z polynomiálneho zápisu ihneď vidíme, že platí:

$$\overline{X1} = 2X + 1, \quad \overline{X2} = 2X + 2,$$

kde \overline{X} je pozičný zápis čísla X .

S využitím hore uvedených rovností si skúsme zadefinovať niektoré aritmetické operácie:

3.5.1 Successor.

$$\frac{(+1) 0}{1} \quad \frac{(+1) X1}{X2} \quad \frac{(+1) X2}{((+1) X)1}.$$

3.5.2 Predecessor.

$$\frac{(-1) 0}{0} \quad \frac{(-1) 1}{0} \quad \frac{(-1) X11}{((-1) X)2} \quad \frac{(-1) X21}{X12} \quad \frac{(-1) X2}{X1}$$

alebo

$$\frac{(-1) 0}{0} \quad \frac{(-1) X1(X = \emptyset)}{0} \quad \frac{(-1) X1(X \neq \emptyset)}{((-1) X)2} \quad \frac{(-1) X2}{X1}.$$

3.5.3 Sčítanie.

$$\begin{array}{ccccccc} 0 & X1 & X1 & X1 & X2 & X2 & X2 \\ +Y & +0 & +Y1 & +Y2 & +0 & +Y1 & +Y2 \\ \hline Y & X1 & (X+Y)2 & (X+Y+1)1 & X2 & (X+Y+1)1 & (X+Y+1)2 \end{array}.$$

3.5.4 Násobenie.

$$\begin{array}{ccc} 0 & X1 & X2 \\ \cdot Y & \cdot Y & \cdot Y \\ \hline 0 & Y & Y \\ + X \cdot Y & + Y & \\ + X \cdot Y & + X \cdot Y & \\ + X \cdot Y & + X \cdot Y & \end{array}.$$

3.5.5 Umocňovanie.

$$\begin{array}{ccc} 0 & X1 & X2 \\ Y & Y & Y \\ \hline 1 & Y \cdot Y^X \cdot Y^X & Y \cdot Y \cdot Y^X \cdot Y^X \end{array}.$$

3.6 Reprézentácia čísel pomocou numerálov

Vo funkcionálnom programovaní pracujeme s *výrazmi*, *termami*. Ako sme si už uviedli, výpočet je vyhodnocovanie výrazov, ich prepis, redukcia do základných, ďalej už nerozložiteľných, *irreducibilných* tvarov. Naším ďalším cieľom bude reprezentovať, implementovať, vyjadriť niektoré číselné sústavy pomocou špeciálnych termov - *numerálov*.

Začnime s monadickou sústavou. Použijeme

- 0 - konštantu, definovanú ako $0 \in \mathbb{N}$ a
- S - unárny funkčný symbol (successor), definovaný ako $S(x) = x + 1$.

Budeme mať nasledovnú korešpondenciu uvedenú na Obrázku 3.5.

číslo	monadická sústava	numerál
0	0	0
1	1	$S(0)$
2	11	$SS(0)$
3	111	$SSS(0)$
\vdots	\vdots	\vdots
n	$\underbrace{111 \dots 1}_n$	$\underbrace{SSS \dots S(0)}_n$

Obrázok 3.5: Reprézentácia čísel v unárnej sústave pomocou numerálov.

Pri diadickej sústave použijeme

- 0 - konštantu, definovanú ako $0 \in \mathbb{N}$ a
- $S1$, $S2$ - unárne funkčné symboly, definované nasledovne:

$$S1(x) = 2 \cdot x + 1,$$

$$S2(x) = 2 \cdot x + 2.$$

V CL-ku sú funkčné symboly $S1$ a $S2$ už preddefinované. Aby sme sa čo najviac priblížili pozičnému diadickému zápisu, budú v systéme Cl výrazy $S1(x)$ a $S2(x)$ zobrazované v postfixovom formáte x_1 a x_2 . Dostávame korešpondenciu uvedenú na Obrázku 3.6.

Všimnite si, že postupnosť 1, 2 a $S1$, $S2$ je zrkadlovo otočená. Vyplýva to z prefixovej notácie $S1(x)$ a $S2(x)$ a postfixovej notácie ich skratiek x_1 a x_2 . Napríklad: $4 = S2S1(0) = S2(0_1) = 0_{12}$. V CL-ku na zobrazenie čísel v diadickej sústave budeme používať formát $N2$. Napríklad, pre $5 = x : N2$ sa zobrazí x ako 0_{21} . Prilepí nám to 0 pred diadický zápis, čo vyplýva z postfixovej notácie skratiek pre $S1$, $S2$.

Preberieme si ďalší typ diskriminácie a to diadickú diskrimináciu. Vyzerá nasledovne:

číslo	diadická sústava	numerál	V CL-ku
0	0	0	0
1	1	$S1(0)$	0_1
2	2	$S2(0)$	0_2
3	11	$S1S1(0)$	0_{11}
4	12	$S2S1(0)$	0_{12}
5	21	$S1S2(0)$	0_{21}
6	22	$S2S2(0)$	0_{22}
7	111	$S1S1S1(0)$	0_{111}
8	112	$S2S1S1(0)$	0_{112}

Obrázok 3.6: Reprézntácia diadickej sústavy pomocou numerálov.

$$\begin{aligned} F(0) &= v_1 \\ F(S1(x)) &= v_2 \\ F(S2(x)) &= v_3 \end{aligned}$$

po kompilácii sa zobrazí ako:

$$\begin{aligned} F(0) &= v_1 \\ F(x_1) &= v_2 \\ F(x_2) &= v_3 \end{aligned}$$

Môžeme mať aj vnorenú diskrimináciu. Napríklad:

$$\begin{aligned} F(0) &= v_1 \\ F(S1(0)) &= v_2 \\ F(S1(S1(x))) &= v_3 \\ F(S1(S2(x))) &= v_4 \\ F(S2(0)) &= v_5 \\ F(S2(S1(x))) &= v_6 \\ F(S2(S2(x))) &= v_7 \end{aligned}$$

po kompilácii sa zobrazí ako

$$\begin{aligned} F(0) &= v_1 \\ F(0_1) &= v_2 \\ F(x_{11}) &= v_3 \\ F(x_{21}) &= v_4 \\ F(0_2) &= v_5 \\ F(x_{12}) &= v_6 \\ F(x_{22}) &= v_7 \end{aligned}$$

Dá sa používať diadická diskriminácia i v nasledovnom tvare:

$$\begin{aligned} F(0) &= v_1 \\ F(2 \cdot x + 1) &= v_2 \\ F(2 \cdot x + 2) &= v_3 \end{aligned}$$

Skúsme si teraz niektoré aritmetické funkcie nad diadickou sústavou klauzálné definovať:

3.6.1 Successor.

$$\begin{aligned} S(0) &= 1 \\ S(S(x_1)) &= x_2 \\ S(S(x_2)) &= S(x)1 \end{aligned}$$

alebo

$$\begin{aligned} S(0) &= 1 \\ S(2 \cdot x + 1) &= 2 \cdot x + 2 \\ S(2 \cdot x + 2) &= 2 \cdot S(x) + 1 . \end{aligned}$$

3.6.2 Predecessor.

Príklad na vnorenú diskrimináciu:

$$\begin{aligned} Pred(0) &= 0 \\ Pred(0_1) &= 0 \\ Pred(x_{11}) &= Pred(x_1)_2 \\ Pred(x_{21}) &= x_1 \\ Pred(x_2) &= x_1 . \end{aligned}$$

Verzia bez vnorenej diskriminácie:

$$\begin{aligned} Pred(0) &= 0 \\ Pred(x_1) &= 0 \leftarrow x = 0 \\ Pred(x_1) &= Pred(x)_2 \leftarrow x \neq 0 \\ Pred(x_2) &= x_1 . \end{aligned}$$

3.6.3 Sčítanie.

$$\begin{aligned} Add(0, y) &= y \\ Add(x_1, 0) &= x_1 \\ Add(x_1, y_1) &= Add(x, y)_2 \\ Add(x_1, y_2) &= S(Add(x, y))_1 \\ Add(x_2, 0) &= x_2 \\ Add(x_2, y_1) &= S(Add(x, y))_1 \\ Add(x_2, y_2) &= S(Add(x, y))_2 . \end{aligned}$$

3.6.4 Násobenie.

$$\begin{aligned} Mul(0, y) &= 0 \\ Mul(x_1, y) &= y + z + z \leftarrow Mul(x, y) = z \\ Mul(x_1, y) &= y + y + z + z \leftarrow Mul(x, y) = z . \end{aligned}$$

3.6.5 Umocňovanie.

$$\text{Exp}(x, 0) = 1$$

$$\text{Exp}(x, y_1) = x \cdot z \cdot z \cdot z \leftarrow \text{Exp}(x, y) = z$$

$$\text{Exp}(x, y_1) = x \cdot x \cdot z \cdot z \cdot z \leftarrow \text{Exp}(x, y) = z$$

Kapitola 4

Kódovanie dátových štruktúr do \mathbb{N}

Pri programovaní sa používa veľké množstvo rôznych dátových štruktúr: n -tice, vektory, matice, viacdimeziálne polia, reťazce, zoznamy, záznamy, zásobníky, fronty, tabuľky, stromy, lesy, grafy atď.. Otázka znie, ako tieto štruktúry implementovať do jazyka, ktorý umožňuje definovať funkcie iba nad prirodzenými číslami. Presnejšie ako zakódovať tieto štruktúry do \mathbb{N} . V nasledujúcom výklade si odpovieme na túto otázku.

4.1 Kódovanie konečných postupností nad konečnou abecedou

Ako prvý krok k našmu cieľu, sa zamyslíme nad kódovaním konečných postupností znakov (zoznamov, reťazcov) nad konečnou abecedou - množinou znakov. Doteraz sme využívali diadickú (p -adickú) číselnú sústavu na 'budovanie' veľkej aritmetiky. Definovali sme si niektoré základné aritmetické funkcie, ktoré sú schopné pracovať s ľubovoľne veľkým číslom bez obmedzenia. Naším jediným reálnym obmedzením je veľkosť pamäte v počítači. Ďalej si ukážeme, ako pomocou p -adickej sústavy môžeme kódovať postupnosti - zoznamy, reťazce znakov nad nejakou konečnou abecedou s počtom znakov p . Bez ujmy na všeobecnosti predpokladajme, že abeceda $\Sigma = \{1, 2, 3, 4, \dots, p\}$. (Naša abeceda sa skladá iba zo znakov - cifier.) Ľubovoľný reťazec znakov z tejto abecedy budeme kódovať číslom, ktorého pozičný zápis v p -adickej sústave zodpovedá danému reťazcu. Napríklad pre $p = 8$, reťazec 23871 budeme kódovať číslom, ktorého zápis v 8-adickej sústave je 0_{23871} . Prázdny reťazec kódujeme 0-ou.

Teraz si zdefinujme jednoduché operácie s reťazcami. Pre jednoduchosť ostaneme v diadickej sústave, budeme teda uvažovať iba reťazce zložené z 1-tiek a 2-ok.

4.1.1 Konkatenácia dvoch reťazcov.

$$\begin{aligned} \text{Con}(X, 0) &= X \\ \text{Con}(X, Y_1) &= \text{Con}(X, Y)_1 \\ \text{Con}(X, Y_2) &= \text{Con}(X, Y)_2 . \end{aligned}$$

Výpočet:

$$\text{Con}(0_{12}, 0_{22}) = \text{Con}(0_{12}, 0_2)_2 = \text{Con}(0_{12}, 0)_{22} = 0_{1222}.$$

4.1.2 Reverz (otočenie reťazca). Rekurzívna verzia:

$$\begin{aligned} \text{Rev}(0) &= 0 \\ \text{Rev}(X_1) &= \text{Con}(0_1, \text{Rev}(X)) \\ \text{Rev}(X_2) &= \text{Con}(0_2, \text{Rev}(X)) . \end{aligned}$$

Príklad:

$$\begin{aligned} \text{Rev}(0_{12}) &= \text{Con}(0_2, \text{Rev}(0_1)) = \text{Con}(0_2, \text{Con}(0_1, \text{Rev}(0))) \\ &= \text{Con}(0_2, \text{Con}(0_1, 0)) = \text{Con}(0_2, 0_1) \\ &= \text{Con}(0_2, 0)_1 = 0_{21}. \end{aligned}$$

Iteratívna (šikovnejšia verzia):

$$\begin{aligned} \text{Rev}(X) &= \text{Revi}(X, 0) \\ \text{Revi}(0, a) &= a \\ \text{Revi}(X_1, a) &= \text{Revi}(X, a_1) \\ \text{Revi}(X_2, a) &= \text{Revi}(X, a_2) . \end{aligned}$$

Príklad:

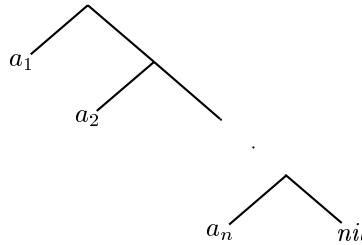
$$\begin{aligned} \text{Rev}(0_{12}) &= \text{Revi}(0_{12}, 0) = \text{Revi}(0_1, 0_2) \\ &= \text{Revi}(0, 0_{21}) = 0_{21}. \end{aligned}$$

Analogickým spôsobom by sa dali zdefinovať i ďalšie elegantné operácie nad diadickými reťazcami.

4.2 Teraz o kódovaní niečo všeobecnejšie

Naším ďalším cieľom bude navrhnúť kódovanie zoznamov (konečných postupností) nad nekonečnou abecedou, napríklad \mathbb{N} . Pozrime sa trochu do histórie funkcionálneho programovania. V jazyku LISP (SCHEME) sa dajú rôzne dátové štruktúry implementovať (kódovať) pomocou *s*-výrazov. *S*-výraz je buď

- atóm
 - numerický: 1, 2, 100,
 - symbolický: jano1, auto (reťazec písmen a číslíc začínajúci písmenom),

Obrázok 4.1: S -výraz kódujúci zoznam (a_1, a_2, \dots, a_n) .

– (špeciálny) preddefinovaný nil ;

- $cons(s_1, s_2)$ - zložený s -výraz (*pár*), kde s_1, s_2 sú nejaké s -výrazy. Budeme značiť aj ako $\bigwedge_{s_1 s_2}$. (V LISP-e sa používa zápis $s_1 . s_2$.)

Ako by sme mohli implementovať pomocou s -výrazov zoznamy - konečné postupnosti nejakých prvkov? Zoznam, označený ako

$$l = (a_1, a_2, \dots, a_n), \quad n \geq 0,$$

budeme kódovať s -výrazom načrtnutým na obrázku 4.1. Prázdny zoznam, ($n = 0$), označíme pomocou atomu nil . Čiže zakódovaný zoznam je

- buď tvaru nil - prázdny zoznam
- alebo tvaru $\bigwedge_{a_1 l_1}$ - neprázdny zoznam, kde a_1 je jeho prvý prvok a l_1 je podzoznam - zvyšok zoznamu.

4.3 Jednoduché funkcie nad zoznamami

Ako budú vyzeráť jednoduché funkcie nad zoznamami?

4.3.1 Prvý prvok zoznamu.

$$H(nil) = nil \quad (\text{definitoricky})$$

$$H\left(\bigwedge_{a_1 l_1}\right) = a_1 .$$

V LISP-e sa označuje ako car .

4.3.2 Zvyšok zoznamu.

$$T(nil) = nil \quad (\text{definitoricky})$$

$$T\left(\bigwedge_{a_1 l_1}\right) = l_1 .$$

V LISP-e sa označuje ako cdr .

4.3.3 Pridanie prvku na začiatok zoznamu.

$$C(a_1, l_1) = \bigwedge_{a_1} l_1 .$$

V LISP-e sa označuje ako *cons*.

4.3.4 Dĺžka zoznamu.

$$L(\text{nil}) = 0$$

$$L(\bigwedge_a l) = L(l) + 1 .$$

Výpočet:

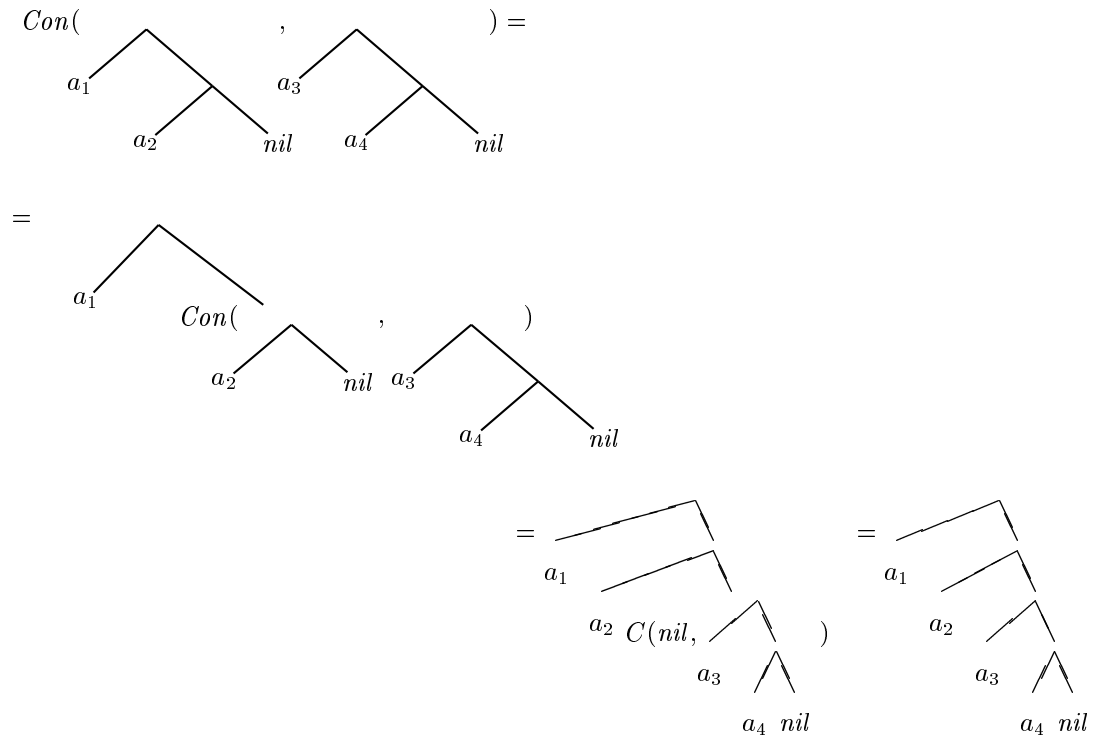
$$\begin{aligned}
 L(\bigwedge_{a_1} \bigwedge_{a_2} \bigwedge_{a_3} \text{nil}) &= L(\bigwedge_{a_2} \bigwedge_{a_3} \text{nil}) + 1 = \\
 &= L(\bigwedge_{a_3} \text{nil}) + 1 + 1 = L(\text{nil}) + 1 + 1 + 1 = 0 + 1 + 1 + 1 = 3.
 \end{aligned}$$

4.3.5 Konkatenácia dvoch zoznamov.

$$Con(\text{nil}, l_2) = l_2$$

$$Con(\bigwedge_a l_1, l_2) = \bigwedge_a Con(l_1, l_2) .$$

Výpočet:



4.4 Kódovanie dátových štruktúr

4.4.1 Párovacie funkcie. Je zrejmé, že pomocou s - výrazov sa budú dať priamočiaro reprezentovať binárne stromy:

1. nil - prázdny strom,
2. $\bigwedge_{\bar{t}_1 \bar{t}_2}$ - neprázdny binárny strom $\bigwedge_{t_1 t_2}$, kde \bar{t}_i je reprezentácia t_i .

O tom ako kódovať ľubovoľné stromy si povieme neskôr.

Poďme teraz spraviť ďalší krok k nášmu cieľu- implementácii, kódovania dátových štruktúr do \mathbb{N} : Budeme sa snažiť stotožniť množinu (*doménu*) s - výrazov s prirodzenými číslami. Môžeme si dovoliť nasledovné matematické zjednodušenie (abstrakcia) lisповských s -výrazov:

- miesto množiny rôznych atómov sa budeme snažiť vystačiť iba s jedným a to s 0 - ou,
- konštruktor- párovač *cons* budeme implemetovať pomocou vhodnej binárnej párovacej funkcie P nad \mathbb{N} , ktorá by mala spĺňať tieto vlastnosti:

1. $P(x, y) = P(v, w) \rightarrow x = v \wedge y = w$
2. $x < P(x, y) \wedge y < P(x, y)$
3. $x = 0 \vee \exists v \exists w x = P(v, w)$

1. vlastnosť nám zaručuje injektívnosť funkcie P , dvom rôznym "párom" x, y a v, w nemôžeme priradiť to isté číslo $P(x, y) = P(v, w)$;
2. vlastnosť sa jednak využije pri korektnosti párovej indukcii, ktorá slúži na dokazovanie správnosti programov- čo sa budete učiť 2. ročníku, a taktiež z nej vyplýva: $\forall x, y : 0 \leq x < P(x, y)$, čiže $\forall x, y : 0 \neq P(x, y)$; nula nie je v obore hodnôt funkcie $P(\text{range}(P))$, čiže: $\text{range}(P) \subseteq \mathbb{N} \setminus \{0\}$.
3. vlastnosť tvorí, že $\text{range}(P) = \mathbb{N} \setminus \{0\}$.

Pomocou 0 a binárneho funkčného symbolu P môžeme vytvárať nasledovné P -výrazy (párové výrazy). Zjednodušenie lisovských s -výrazov:

0; *jediný atóm*

$P(0, 0); P(P(0, 0), 0); P(P(0, 0), P(0, 0))$ atď... P - výrazy

Keď máme zložené P "fixované" nijakou vhodnou funkciou spĺňajúcou vlastnosti 1-3., môžeme každému prirodzenému číslu n jednoznačne priradiť P - výraz (jeho párový zápis), ktorého hodnota bude práve n . Čiže prirodzené čísla budeme vedieť reprezentovať pomocou P - výrazov. Takúto reprezentáciu budeme volať párová reprezentácia priradených čísel a P -výrazy budeme nazývať *P -numeralmi*, keďže reprezentujú prirodzené čísla. Pod párovou veľkosťou čísla n budeme rozumieť počet P -čok v jeho párovom zápise, označíme ju ako $|n|$.

4.4.2 Cantorova párovacia funkcia. Skúsme teraz pohľadať vhodných kandidátov na funkciu P spĺňajúcich podmienky 1., 2. a 3.. Z matematickej analýzy poznáte Cantorovu funkciu, ktorá bijektívne zobrazuje \mathbb{N}^2 na \mathbb{N} . Znázorníme si ju pomocou tabuľky 4.2. Môžeme si ju zmodifikovať nasledovne: šiftneme o

$C(x, y)$	0	1	2	3	4	...
0	0	1	3	6	10	...
1	2	4	7	11	16	...
2	5	8	12	17	23	...
3	9	13	18	24	31	...
4	14	19	25	32	40	...
⋮	⋮	⋮	⋮	⋮	⋮	

Obrázok 4.2: Cantorova párovacia funkcia

$J(x, y)$	0	1	2	3	4	...
0	1	2	4	7	11	...
1	3	5	8	12	17	...
2	6	9	13	18	24	...
3	10	14	19	25	32	...
4	15	20	26	33	41	...
⋮	⋮	⋮	⋮	⋮	⋮	

Obrázok 4.3: Modifikovaná Cantorova párovacia funkcia

jedna (pozri tabuľku 4.3) a dostaneme funkciu- bijekciu z \mathbb{N}^2 na $\mathbb{N} \setminus \{0\}$.
Explicitný vzorec:

$$J(x, y) = \frac{(x + y) \cdot (x + y + 1)}{2} + (x + 1).$$

Napríklad: $J(1, 2) = \frac{3 \cdot 4}{2} + 2 = 8$. Označme ju J . Funkcia J spĺňa podmienky 1-3.. Môžeme teda pomocou J - numerálov reprezentovať čísla. Funkcia J má

čísla	J - numerál
0	0
1	$J(0, 0)$
2	$J(0, 1) = J(0, J(0, 0))$
3	$J(1, 0) = J(J(0, 0), 0)$
4	$J(0, 2) = J(0, J(0, J(0, 0)))$
5	$J(1, 1) = J(J(0, 0), J(0, 0))$
6	$J(2, 0) = J(J(0, J(0, 0)), 0)$
7	$J(0, 3) = J(0, J(J(0, 0), 0))$
8	$J(1, 2) = J(J(0, 0), J(0, J(0, 0)))$
9	$J(2, 1) = J(J(0, J(0, 0)), J(0, 0))$
10	$J(3, 0) = J(J(J(0, 0), 0), 0)$

Obrázok 4.4: J -numerály


však určité nevýhody:

- čísla s rovnakou párovou veľkosťou nie sú spolu. Napríklad: čísla 7, 8, 9, 10 (pozri v tabuľke 4.3)
- dá sa ukázať, že zápis pomocou J - numerálov nie je "veľmi úsporný", obsahuje príliš veľa J -čok v párovom zápise nejakého čísla n .

4.4.3 Kódovanie pomocou binárnych stromov. Skúsme sa pozrieť po vhodnejšom kandidátovi, ktorý by nemal spomenuté nevýhody. Chceme, aby

čísla s rovnakou párovacou veľkosťou boli držané spolu. Všeobecne, na párový numeral pre nejaké číslo n sa môžeme pozrieť ako na binárny strom:

- 0- je reprezentované ako prázdny strom, označíme ho ako \bullet (bodka),
- $P(x, y)$ - zobrazíme ako $\bigwedge_{\bar{x} \bar{y}}$, kde \bar{x}, \bar{y} sú binárne stromy pre numerály x, y .

Napríklad: $P(P(0, 0), P(0, 0))$ zobrazíme ako .

Poznamenajme, že párová veľkosť zápisu, počet P -čok v párovom zápise, je rovná počtu vnútorných vrcholov v jeho stromovom zobrazení. Ľahko sa dá o tom presvedčiť z definície stromového zobrazenia párového numerálu.

binárne stromy s počtom vnútorných vrcholov 0.	binárne stromy s počtom vnútorných vrcholov 1.	atď ...
--	--	---------

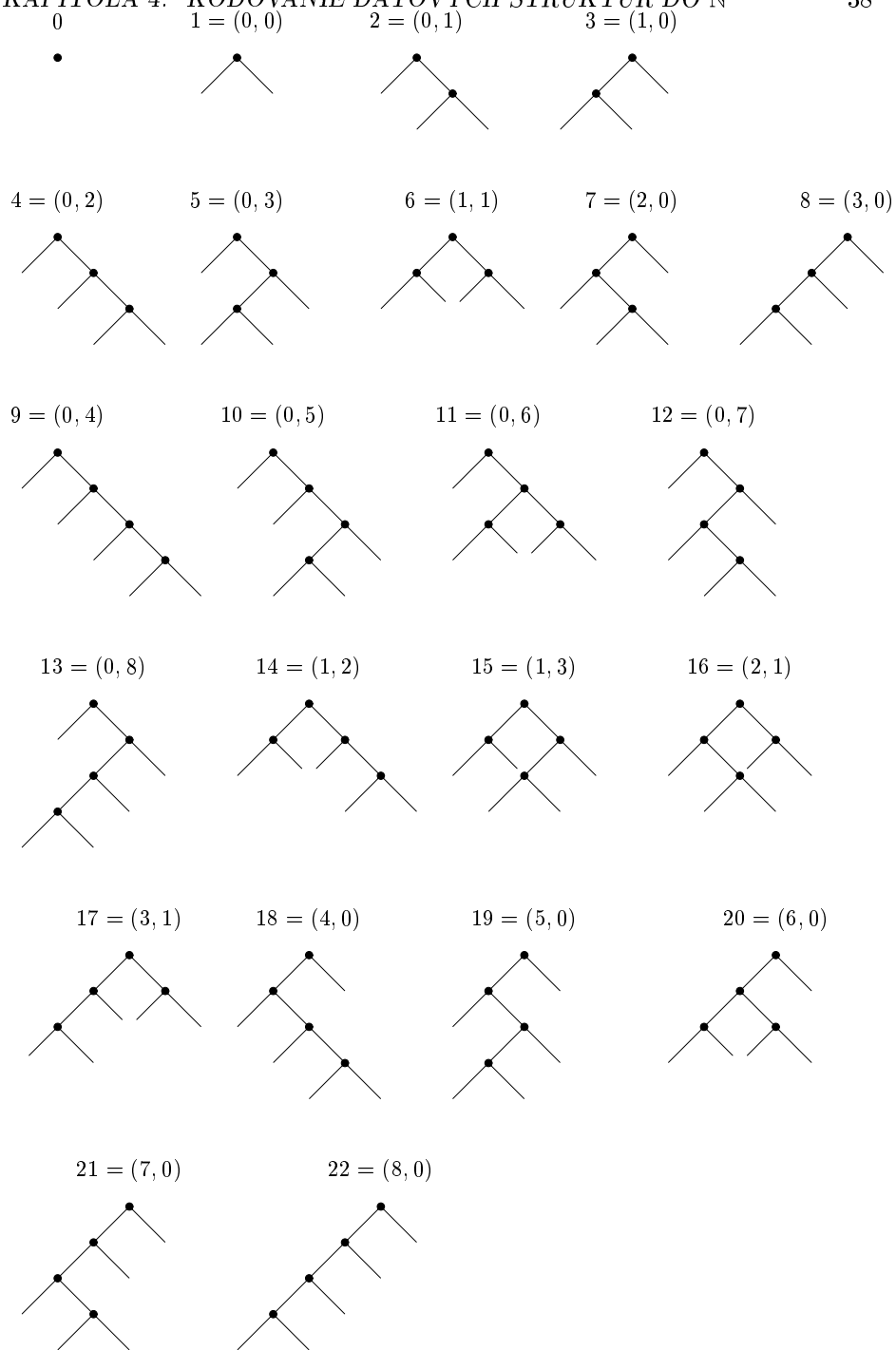
Obrázok 4.5: Binárne stromy

Vhodného kandidáta na párovaciu funkciu P dostaneme nasledovne: Bude-
me enumerovať všetky binárne stromy (stromové zobrazenia párových numerá-
lov) - vytvárať nekonečnú postupnosť binárnych stromov: b_0, b_1, b_2, \dots , (čiže 0
priradíme binárny strom b_0 (a tým odpovedajúci párový numeral), 1 zase b_1 atď
...) takým spôsobom, že binárne stromy s počtom vnútorných 0 (len prázdny
strom), potom s počtom vnútorných vrcholov 1, 2, 3 atď.. Tým dosiahneme, že
čísla rovnakou párovou veľkosťou budú držané spolu a stromy s menším počtom
vnútorných vrcholov budú predchádzať stromy s väčším počtom vnútorných vr-
cholov. Našu postupnosť bude znázorňuje Obrázok 4.5.

Ako teraz vymenovať v jednom bloku všetky binárne stromy s rovnakým poč-
tom vnútorných vrcholov? Zoradíme ich lexikograficky: nech t_1, t_2 sú binárne
stromy s rovnakým počtom vnútorných vrcholov, potom t_1 je pred t_2 ak ľavý
podstrom t_1 je pred ľavým podstromom t_2 alebo ak ľavé podstromy sú rovnaké
(zhodné, identické) tak pravý podstrom t_1 musí byť pred pravým podstromom
 t_2 . Tohoto kandidáta (na párovaciu funkciu) budeme označovať čiarkou " $,$ " a
používať *infixovú notáciu*. Napríklad: $(x, y); (0, (0, 0))$. Obrázok 4.6 znázorňuje
začiatok tejto postupnosti.

Ako vidíme z konštrukcie, táto postupnosť nám jednoznačne fixuje párovaciu
funkciu (x, y) . Pre dve čísla x, y vezmeme x -ty a y -ty binárny strom (počítajúc
od 0) t_1, t_2 . Potom pozícia binárneho stromu $\bigwedge_{t_1 t_2}$ bude určovať hodnotu (x, y) .

Dá sa ukázať, že takto definovaná funkcia " $,$ " spĺňa vlastnosti 1-3. a navyše
pre párovú veľkosť čísla n platí, že $|n|$ je približne $\log n$, čiže dostávame zápis s



Obrázok 4.6: Enumerácia binárnych stromov

logaritmickej dĺžkou, ktorý považujeme už za ekonomický (ako pri n -árnych a p -adických systémoch pre $n, p > 1$). Podarilo sa nám eliminovať obidve nevýhody funkcie J . Funkcia " \cdot " navyše spĺňa nasledujúce vlastnosti:

$$4. \ x < y \rightarrow |X| \leq |y|$$

$$5. \ |x| < |y| \rightarrow x < y$$

$$6. \ |(x_1, x_2)| = |(y_1, y_2)| \rightarrow ((x_1, x_2) < (y_1, y_2) \leftrightarrow x_1 < y_1 \vee x_1 = y_1 \wedge x_2 < y_2)$$

7. monotónnosť:

$$x_1 < x_2 \rightarrow (x_1, y) < (x_2, y)$$

$$y_1 < y_2 \rightarrow (x, y_1) < (x, y_2) .$$

4.4.4 Kontrakcia do unárnych funkcií. Teraz si ukážeme prirodzenú korešpondenciu medzi unárnymi a n -árnymi funkciami nad \mathbb{N} . Najskôr malá poznámka k značeniu, zápisu " \cdot "-numerálov, aby sme nemuseli zbytočne písať veľa zátvoriek " \cdot "- numerálov tvaru

$(x_1, (x_2, \dots, (x_{n-1}, x_n) \dots))$ budeme skracovať do tvaru: $x_1, x_2, \dots, x_{n-1}, x_n$.

Nech f je najaká n -árna funkcia nad \mathbb{N} , pokúsime sa ju "implementovať" pomocou unárnej funkcie-kontrakcie pre f , označnej ako $\langle f \rangle$, a párovacej funkcie " \cdot ".

Uvažujme nasledovnú definíciu:

$$\langle f \rangle (x) = \begin{cases} f(\underbrace{x_1, x_2, \dots, x_n}_{\text{oddelovače}}) & \text{if } x = \underbrace{x_1, \dots, x_n}_{\text{párovače}} \\ 0 & \text{otherwise} \end{cases} .$$

Z nej dostávame, že

$$f(\underbrace{x_1, x_2, \dots, x_n}_{\text{čiarky sú syntaktické oddelovače argumentov}}) = \langle f \rangle (\underbrace{x_1, \dots, x_n}_{\text{čiarky sú identifikátory pre našu párovaciu funkciu "\cdot"}})$$

Čiže bez ujmy na všeobecnosti sa môžeme zaoberať bez n -árnych funkcií a vystačíť iba s unárnymi a párovaciu binárnou funkciou " \cdot ". Poznámka: pre unárnu funkciu f je $\langle f \rangle = f$.

Príklady:

$\max(x, y)$

$$\langle \max \rangle (z) = \begin{cases} \max(x, y) & \text{if } z = x, y \\ 0 & \text{otherwise} \end{cases}$$

$x + y$

$$\langle + \rangle (z) = \begin{cases} x + y & \text{if } z = x, y \\ 0 & \text{otherwise} \end{cases}$$

4.4.5 Ako je to v CL ?. CL umožňuje definovať ľubovoľne n -árne funkcie. Napríklad: binárnu funkciu *Max* zadefinujeme:

fun/2Max

$Max(x, y) = x \leftarrow x > y$

$Max(x, y) = y \leftarrow x \leq y$.

Doteraz ste implicitne, nevedomky miesto n -árnych funkcií definovali ich kontrakcie. Miesto *fun/1* môžeme písať *fun*. Ako zistiť kedy ", " je párovacia funkcia a kedy oddeľovač argumentov ? Na zistenie jednoznačnosti zaveďme nasledovnú konvenciu. Nech f je n -árna funkcia a $m \geq n$ potom chápeme

$$f(x_1, \dots, x_{n-1}, x_n, \dots, x_m)$$

ako

$$f(\underbrace{x_1, \dots, x_{n-1}}_{\text{oddeľovače}}, \underbrace{(x_n, \dots, x_m)}_{\text{párovače}})$$

Čiže:

- x_1 je 1. argument
- x_{n-1} je $n - 1$. argument
- x_n, \dots, x_m je n . argument .

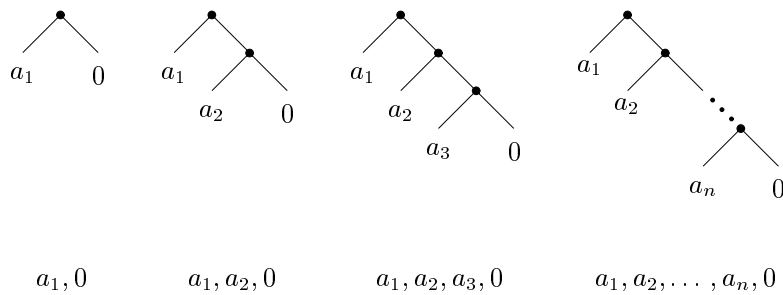
Ak by $m < n$, tak ide o zlý zápis aplikácie f . Pri unárnej funkcii všetky ", " sú párovacie funkcie.

Kapitola 5

Zoznamy

5.1 Kódovanie konečnej postupnosti

Na minulej prednáške sme si stručne načrtli implementáciu zoznamov v LISPE pomocou s - výrazov. Teraz zoznamy budeme implementovať pomocou $"$ - výrazov.



Obrázok 5.1: Zoznamy

Zoznam je konečná postupnosť prvkov (a_1, \dots, a_n) .

1. Ak $n = 0$ (prázdny zoznam), tak ho reprezentujeme ako 0.
2. Ak $n > 0$, tak ho reprezentujeme ako na Obrázke 5.1.

Podľa dohody $(a_1, (a_2, \dots, (a_n, 0) \dots))$ skracujeme na $a_1, \dots, a_n, 0$.

Dôležitá poznámka: p -adické systavy nám umožnili kódovanie konečných postupností nad konečnou abecedou $\{1, \dots, p\}$. Párová reprezentácia nám umožňuje omnoho všeobecnejšie kódovanie konečných postupností nad nekonečnou abecedou \mathbb{N} , a_i môže byť ľubovoľné prirodzené číslo. Čiže sú možné dva prípady zoznamov:

1. prázdny zoznam: 0

2. neprázdny zoznam: x, y ; kde x je prvý prvok a y je podzoznam-zvyšok zoznamu.

Všeobecne v párovej reprezentácii číslo n je tvaru 0 ak $n = 0$ alebo je tvaru x, y ; kde x, y sú nejaké ", "-numerály ak $n > 0$. Z toho dostávame v CL-ku párovú diskrimináciu:

v hlave

$$F(0) = v_1 \dots$$

$$F(x, y) = v_2 \dots$$

v tele

$$F(z) = v_1 \leftarrow z = 0$$

$$F(z) = v_2 \leftarrow z = x, y .$$

Môžeme mať i vnorené podprípady. Napríklad:

v vhlave

$$F(0) = v_1 \dots$$

$$F(x, 0) = v_2 \dots$$

$$F(x, y, z) = v_3 \dots$$

zmiešaná

$$F(0) = v_1$$

$$F(x, p) = v_2 \leftarrow p = 0$$

$$F(x, p) = v_3 \leftarrow p = y, z$$

v tele

$$F(r) = v_1 \leftarrow r = 0$$

$$F(r) = v_2 \leftarrow r = x, p \wedge p = 0$$

$$F(r) = v_3 \leftarrow r = x, p \wedge p = y, z$$

alebo

$$F(r) = v_1 \leftarrow r = 0$$

$$F(r) = v_2 \leftarrow r = x, 0$$

$$F(r) = v_3 \leftarrow r = x, y, z .$$

5.2 Základné operácie nad zoznamami

Precvičíme si ju na nasledujúcich príkladoch.

5.2.1 Prvý prvok.

$$H(0) = 0 \quad \text{dohoda}$$

$$H(x, y) = x$$

5.2.2 Zvyšok zoznamu.

$$T(0) = 0 \quad \text{dohoda}$$

$$T(x, y) = y$$

5.2.3 Dĺžka zoznamu.

$$L(0) = 0$$

$$L(x, y) = L(y) + 1$$

Výpočet :

$$\begin{aligned} L(0, 0, 0, 0) &= L(0, 0, 0) + 1 = L(0, 0) + 1 + 1 \\ &= L(0) + 1 + 1 + 1 = 0 + 1 + 1 + 1 = 3 . \end{aligned}$$

5.2.4 Konkatenácia dvoch zoznamov.

$$0 \oplus y = y$$

$$(x_1, x_2) \oplus = x_1, (x_2 \oplus y) = x_1, x_2 \oplus y$$

Výpočet:

$$\begin{aligned} (1, 2, 3, 0) \oplus (4, 5, 0) &= 1, (2, 3, 0) \oplus (4, 5, 0) = 1, 2, (3, 0) \oplus (4, 5, 0) = \\ &= 1, 2, 3, 0 \oplus (4, 5, 0) = 1, 2, 3, 4, 5, 0 . \end{aligned}$$

5.2.5 Reverz.

$$Rev(0) = 0$$

$$Rev(x, y) = Rev(y) \oplus (x, 0)$$

Výpočet:

$$\begin{aligned} Rev(1, 2, 3, 0) &= Rev(2, 3, 0) \oplus (1, 0) = (Rev(3, 0) \oplus (2, 0)) \oplus (1, 0) \\ &= ((Rev(0) \oplus (3, 0)) \oplus (2, 0)) \oplus (1, 0) \\ &= ((0 \oplus (3, 0)) \oplus (2, 0)) \oplus (1, 0) \\ &= ((3, 0) \oplus (2, 0)) \oplus (1, 0) = (3, 0 \oplus (2, 0)) \oplus (1, 0) \\ &= (3, 2, 0) \oplus (1, 0) = 3, (2, 0) \oplus (1, 0) \\ &= 3, 2, 0 \oplus (1, 0) = 3, 2, 1, 0 . \end{aligned}$$

Iteratívny:

$$Rev(x) = Revi(x, 0)$$

$$Revi(0, a) = a$$

$$Revi((x, y), a) = Revi(y, x, a) .$$

Výpočet:

$$\begin{aligned} Revi(1, 2, 3, 0) &= Revi((1, 2, 3, 0), 0) = Revi((2, 3, 0), 1, 0) \\ &= Revi((3, 0), 2, 1, 0) = Revi(0, 3, 2, 1, 0) \\ &= Revi(3, 2, 1, 0) . \end{aligned}$$

5.2.6 i -ty prvok zoznamu. Počítame od nuly.

$$\begin{aligned} \textit{Take}(0, i) &= 0 && \textit{dohoda} \\ \textit{Take}((x, y), 0) &= x \\ \textit{Take}((x, y), i + 1) &= \textit{Take}(y, i) \end{aligned}$$

Výpočet:

$$\begin{aligned} \textit{Take}((1, 2, 3, 4, 0), 2) &= \textit{Take}((2, 3, 4, 0), 1) \\ &= \textit{Take}((3, 4, 0), 0) = 3 . \end{aligned}$$

$$\begin{aligned} \textit{Take}(0, 0) &= 0 && \textit{dohoda} \\ \textit{Take}(0, x, y) &= x \\ \textit{Take}(i + 1, 0) &= 0 && \textit{dohoda} \\ \textit{Take}(i + 1, x + y) &= \textit{Take}(i, y) . \end{aligned}$$

Výpočet:

$$\begin{aligned} \textit{Take}(2, 1, 2, 3, 4, 0) &= \textit{Take}(1, 2, 3, 4, 0) \\ &= \textit{Take}(0, 3, 4, 0) = 3 . \end{aligned}$$

5.2.7 Vymaže prvých i prvkov.

$$\begin{aligned} \textit{Drop}(z, 0) &= z \\ \textit{Drop}(z, i + 1) &= 0 \leftarrow z = 0 \\ \textit{Drop}(z, i + 1) &= \textit{Drop}(y, i) \leftarrow z = x, y \end{aligned}$$

Výpočet:

$$\begin{aligned} \textit{Drop}((1, 2, 3, 0), 2) &= \textit{Drop}((2, 3, 0), 1) \\ &= \textit{Drop}((3, 0), 0) = 3, 0 . \end{aligned}$$

Ak vymeníme argumenty

$$\begin{aligned} \textit{Drop}(0, z) &= z \\ \textit{Drop}(i + 1, 0) &= 0 \\ \textit{Drop}(i + 1, x, y) &= \textit{Drop}(i, y) \end{aligned}$$

Výpočet:

$$\begin{aligned} \textit{Drop}(2, 1, 2, 3, 0) &= \textit{Drop}(1, 2, 3, 0) \\ &= \textit{Drop}(0, 3, 0) = 3, 0 . \end{aligned}$$

5.2.8 Párová veľkosť.

$$\begin{aligned} |0| &= 0 \\ |x, y| &= |x| + |y| + 1 \end{aligned}$$

Výpočet:

$$\begin{aligned} |(0, 0), (0, 0)| &= |0, 0| + |0, 0| + 1 \\ &= |0||0| + 1 + |0| + |0| + 1 + 1 \\ &= 0 + 0 + 1 + 0 + 0 + 1 + 1 = 3 . \end{aligned}$$

Kapitola 6

Predikáty

6.1 Definícia

Na prvej prednáške sme si charakterizovali jazyk logiky. Povedali sme si, že obsahuje symboly pre

- premenné: x, y, z, \dots
- logické spojky: $\neg, \wedge, \vee, \rightarrow, \leftarrow$
- kvantifikátory: \forall, \exists
- pomocné symboly: $(,)$

Tieto symboly súhrne nazveme logickými.

6.1.1 Funkčné symboly. Ďalej, že obsahuje funkčné symboly s *aritou* ≥ 0 . (0-árne funkčné symboly chápeme ako symboly pre konštanty) a predikátové symboly s aritou ≥ 1 . Tieto symboly súhrne nazveme špeciálnymi. Ak máme nijakú množinu (prvkov), doménu D môžeme funkčné symboly interpretovať, dať im zmysel, význam pomocou funkcií nad D : n -árnemu funkčnému symbolu, napr. f , priradíme n -árnu funkciu, značíme ju $f^I, f^I: D^n \mapsto D$. V našom výklade sme sa zamerali na $D = \mathbb{N}$ a pre funkcie nad N sme hľadali matematicky korektné definície, pomocou ktorých sme dokázali počítať, vyhodnocovať definované funkcie pre vstupné argumenty.

6.1.2 Predikátové symboly. Analogická situácia bude i pri predikátových symboloch. Vy ste sa už stretli s mnohými najmä binárnymi reláciami (predikátmi). Napr. na N : $<, \geq, =, |$. Pomocou binárneho predikátového symbolu $=$ ste označovali množinu-binárnu reláciu, predikát

$$\{(x, x) | x \in \mathbb{N}\} \subseteq \mathbb{N}^2$$

6.2.2 Pair. Predikát, ktorý platí keď x je pár.

Matematicky:

$$Pair(x) \leftrightarrow \exists x_1, x_2 (x_1, x_2 = x) .$$

Klauzálné:

$$Pair(x) \leftarrow x = x_1, x_2$$

alebo

$$Pair(x_1, x_2) .$$

6.2.3 Triple. Predikát, ktorý platí, keď x je trojica.

Matematicky:

$$Triple(x) \leftrightarrow \exists x_1, x_2, x_3 (x_1, x_2, x_3 = x) .$$

Klauzálné:

$$Triple(x) \leftarrow x = x_1, x_2, x_3$$

alebo

$$Triple(x_1, x_2, x_3) .$$

6.2.4 P. Predikát, ktorý platí, keď x je pár trojíc.

Matematicky:

$$Ptriple(x) \leftrightarrow \exists x_1, x_2 (x_1, x_2 = x \wedge Triple(x_1) \wedge Triple(x_2)) .$$

Klauzálné:

$$Ptriple \leftarrow x = x_1, x_2 \wedge Triple(x_1) \wedge Triple(x_2)$$

alebo

$$Ptriple(x_1, x_2) \leftarrow Triple(x_1) \wedge Triple(x_2) .$$

6.2.5 Zúplnenie predikátov. Od našich klauzálnych definícií predikátov budeme vyžadovať, aby podobne ako pri definíciách funkcií spĺňali

1. výlučnosť
2. úplnosť
3. podmienku regularity (existenciu miery).

Takisto tieto podmienky nám zaručia, že klauzálna definícia bude korektne definovať reláciu - relácia bude určená jednoznačne a bude existovať. Keď sa pozrieme na predchádzajúce definície, vidíme ihneď, že sú výlučné a rekurzívne aj regulárne. Ako je to s úplnosťou? Uvedené definície nie sú úplne. Zúplnime ich nasledovne:

$$\begin{aligned} \text{Even} &\leftarrow x \bmod 2 = 0 \\ \neg \text{Even} &\leftarrow x \bmod 2 \neq 0 \end{aligned}$$

$$\begin{aligned} \text{Even}(0) \\ \neg \text{Even}(1) \\ \text{Even}(x+2) &\leftarrow \text{Even}(x) \\ \neg \text{Even}(x+2) &\leftarrow \text{Even} \end{aligned}$$

$$\begin{aligned} 0 &\geq y \\ \neg x + 1 &\geq 0 \\ x + 1 &\geq y + 1 \leftarrow x \geq y \\ \neg x + 1 &\geq y + 1 \leftarrow \neg x \geq x \geq y \end{aligned}$$

$$\begin{aligned} \neg \text{Pair} &\leftarrow x = 0 \\ \text{Pair} &\leftarrow x = x_1, x_2 \end{aligned}$$

alebo

$$\begin{aligned} \neg \text{Pair}(0) \\ \text{Pair}(x_1, x_2) \end{aligned}$$

$$\begin{aligned} \neg \text{Triple}(x) &\leftarrow x = 0 \\ \neg \text{Triple}(x) &\leftarrow x = x_1, 0 \\ \text{Triple}(x) &\leftarrow x = x_1, x_2, x_3 \end{aligned}$$

alebo

$$\begin{aligned} \neg \text{Triple}(0) \\ \neg \text{Triple}(x_1, 0) \\ \text{Triple}(x_1, x_2, x_3) \end{aligned}$$

$$\begin{aligned} \neg \text{Ptriple}(x) &\leftarrow x = 0 \\ \neg \text{Ptriple}(x) &\leftarrow x = x_1, x_2 \wedge \neg \text{Triple}(x_1) \\ \neg \text{Ptriple}(x) &\leftarrow x = x_1, x_2 \wedge \text{Triple}(x_1) \wedge \neg \text{Triple}(x_2) \\ \text{Ptriple}(x) &\leftarrow x = x_1, x_2 \wedge \text{Triple}(x_1) \wedge \text{Triple}(x_2) \end{aligned}$$

alebo

$$\begin{aligned} \neg \text{Ptriple}(0) \\ \neg \text{Ptriple}(x_1, x_2) &\leftarrow \neg \text{Triple}(x_1) \\ \neg \text{Ptriple}(x_1, x_2) &\leftarrow \text{Triple}(x_1) \wedge \neg \text{Triple}(x_2) \\ \text{Ptriple}(x_1, x_2) &\leftarrow \text{Triple}(x_1) \wedge \text{Triple}(x_2) . \end{aligned}$$

Vidíme, že teraz naše doplnené klauzálne definície sú naozaj úplne - pre každý prípad existuje klauzula. Avšak všetky doplnené klauzuly sú tvaru

$$\neg \text{hlava} \leftarrow \dots ,$$

to jest sú negatívne. Podobne ako sme zúplňovali funkcie, pre všetky prípady, pre ktorú neexistovala klauzula, sme sa dohodli, že $F(.) = 0$, budeme zúplňovať i

predikátové definície: pre všetky prípady, pre ktoré neexistuje klauzula budeme uvažovať negatívne klauzuly $\neg \text{hlava} \leftarrow \dots$, t.j. , že v týchto prípadoch predikát neplatí. Preto v definíciach nikdy nebudeme písať negatívne klauzuly (v CL-ku sú dokonca zakázané). Na základe tejto konvencie sú naše predchádzajúce definície takto implicitne úplne.

Kapitola 7

Charakteristické funkcie

Teraz si povieme niečo o charakteristických funkciách k predikátom a o tom ako pomocou nich vyhodnocujeme, počítame predikáty.

Nech R je predikát, tak charakteristická funkcia k R , označená ako R_* musí spĺňať nasledujúce podmienky:

$$P1: R_* = 0 \vee R_*(x) = 1$$

$$P2: R(x) \leftrightarrow R_*(x) = 1 .$$

Z toho bezprostredne vyplýva:

$$R_*(x) = 1 \rightarrow R(x)$$

$$R_*(x) = 0 \rightarrow \neg R(x)$$

(z $\neg R(x) \leftarrow \neg R_*(x) = 1$ a $P1$). Poďme si nájsť k definovaným predikátom charakteristické funkcie:

$$Even_*(x) \leftarrow x \bmod 2 = 0$$

$Even_*(x) = 0 \leftarrow x \bmod 2 \neq 0$ je zúplnenie, ktoré je implicitné, lebo

$$Even_*(x) = 0 ,$$

preto ho nemusíme písať.

$$Even_*(0) = 1$$

$$Even_*(1) = 0 \quad \text{zúplnenie implicitné}$$

$$Even_*(x+2) = 1 \leftarrow Even_*(x) = 1$$

$$Even_*(x+2) = 0 \leftarrow Even_*(x) = 0 \quad \text{zúplnenie explicitné}$$

Všimnime si, že klauzuly tvaru $P_*(\bar{a}) = 0 \leftarrow \dots$ odpovedajú negatívnym klauzulám tvaru $\neg P(\bar{a})$ a na základe našich dvoch dohôd pre implicitné zúplnenie funkčných a predikátových definícií svorne vynacháme. Keďže nechceme zbytočne rozširovať náš výpočtový model a chceme naďalej zostať pri vyhodnocovaní funkcií, predikáty budeme vyhodnocovať pomocou ich charakteristických

funkcií. Ak charakteristická funkcia sa vyhodnotí do 1 pre daný vstup tak potom predikát pre daný vstup platí. A naopak ak sa char. funkcia vyhodnotí do 0, predikát pre daný vstup neplatí. Príklad:

$$\begin{aligned} 0 \leq_* y &= 1 \\ x + 1 \leq_* &= 0 && \text{zúplnenie} \\ x + 1 \leq_* y + 1 &= 1 \leftarrow x \leq_* y = 1 \\ x + 1 \leq_* y + 1 &= 0 \leftarrow x \leq_* y = 0 && \text{zúplnenie} \end{aligned}$$

$2 \leq 3$ je true

$$\begin{aligned} 2 \leq_* 3 &= 1 \\ 1 \leq_* 2 &= 1 \quad \swarrow \text{preto} \\ 0 \leq_* 1 &= 1 \quad \swarrow \text{preto} \end{aligned}$$

$3 \leq 2$ je false

$$\begin{aligned} 3 \leq_* 3 &= 0 \\ 2 \leq_* 1 &= 0 \quad \swarrow \text{preto} \\ 1 \leq_* 0 &= 1 \quad \swarrow \text{preto} \end{aligned}$$

Skúsme si nasledovný výpočet pre rekurzívnu definíciu $Even(x)$. Budeme počítať $Even(8)$:

$$\begin{aligned} Even_*(8) &= 1 \\ Even_*(6) &= 1 \quad \swarrow \text{preto} \\ Even_*(4) &= 1 \quad \swarrow \text{preto} \\ Even_*(2) &= 1 \quad \swarrow \text{preto} \\ Even_*(0) &= 1 \quad \swarrow \text{preto} \end{aligned}$$

preto je $Even(8)$ true.

Počítajme $Even(7)$:

$$\begin{aligned} Even_*(7) &= 1 \\ Even_*(5) &= 0 \quad \swarrow \text{preto} \\ Even_*(3) &= 0 \quad \swarrow \text{preto} \\ Even_*(1) &= 0 \quad \swarrow \text{preto} \end{aligned}$$

preto je $Even(7)$ false. Takže, predikát $Even$ sa pomocou jeho charakteristickej funkcie vyhodnocuje správne.

$$\begin{aligned} \times Pair_*(x) &= 0 \leftarrow x = 0 \\ Pair_*(x) &= 1 \leftarrow x = x_1, x_2 \end{aligned}$$

alebo

$$\begin{aligned} \times Pair_*(0) &= 0 \\ Pair_*(x_1, x_2) &= 1 \end{aligned}$$

$$\begin{aligned} \times Triple_*(x) &= 0 \leftarrow x = 0 \\ \times Triple_*(x) &= 0 \leftarrow x = x_1, 0 \\ Triple_*(x) &= 1 \leftarrow x = x_1, x_2, x_3 \end{aligned}$$

alebo

$$\begin{aligned}
& \times \text{Triple}_*(0) = 0 \\
& \times \text{Triple}_*(x_1, 0) = 0 \\
& \text{Triple}_*(x_1, x_2, x_3) = 1 \\
& \times \text{Ptriple}_*(x) = 0 \leftarrow x = 0 \\
& \times \text{Ptriple}_*(x) = 0 \leftarrow x = x_1, x_2 \wedge \text{Triple}_*(x_1) = 0 \\
& \times \text{Ptriple}_*(x) = 0 \leftarrow x = x_1, x_2 \wedge \text{Triple}_*(x_1) = 1 \wedge \text{Triple}_*(x_2) = 0 \\
& \text{Ptriple}_*(x) = 1 \leftarrow x = x_1, x_2 \wedge \text{Triple}_*(x_1) = 1 \wedge \text{Ptriple}_*(x_2) = 1 \\
& \times \text{Ptriple}_*(0) = 0 \\
& \times \text{Ptriple}_*(x_1, x_2) = 0 \leftarrow \text{Triple}_*(x_1) = 0 \\
& \times \text{Ptriple}_*(x_1, x_2) = 0 \leftarrow \text{Triple}_*(x_1) = 1 \wedge \text{Triple}_*(x_2) = 0 \\
& \text{Ptriple}_*(x_1, x_2) = 1 \leftarrow \text{Triple}_*(x_1) = 1 \wedge \text{Triple}_*(x_2) = 1
\end{aligned}$$

7.0.6 Predikát Eq. Zdefinujme si ďalší predikát:

$$\text{Eq}(x, y) \leftrightarrow x = y .$$

Pomocou párovej rekurzie:

$$\begin{aligned}
& \text{Eq}(0, 0) \\
& \text{Eq}((x_1, x_2), y_1, y_2) \leftarrow \text{Eq}(x_1, y_1) \wedge \text{Eq}(x_2, y_2) .
\end{aligned}$$

(využívame *PI* vlastnosť párovaciej funkcie "=", "-jej injektívnosť)

Úplná definícia:

$$\begin{aligned}
& \neg \text{Eq}(0) \\
& \text{Eq}(0, 0) \\
& \neg \text{Eq}(0, y_1, y_2) \\
& \neg \text{Eq}((x_1, x_2), 0) \\
& \neg \text{Eq}((x_1, x_2), y_1, y_2) \leftarrow \neg \text{Eq}(x_1, y_1) \\
& \neg \text{Eq}((x_1, x_2), y_1, y_2) \leftarrow \text{Eq}(x_1, y_1) \wedge \neg \text{Eq}(x_2, y_2) \\
& \neg \text{Eq}((x_1, x_2), y_1, y_2) \leftarrow \text{Eq}(x_2, y_2) .
\end{aligned}$$

K tomu priamočiara spravíme charakteristickú funkciu:

$$\begin{aligned}
& \text{Eq}_*(0) = 0 \\
& \text{Eq}_*(0, 0) = 0 \\
& \text{Eq}_*(0, y_1, y_2) = 0 \\
& \text{Eq}_*((x_1, x_2), 0) = 0 \\
& \text{Eq}_*((x_1, x_2), y_1, y_2) = 0 \leftarrow \text{Eq}_*(x_1, y_1) = 0 \\
& \text{Eq}_*((x_1, x_2), y_1, y_2) = 0 \leftarrow \text{Eq}_*(x_1, y_1) = 1 \wedge \text{Eq}_*(x_2, y_2) = 0 \\
& \text{Eq}_*((x_1, x_2), y_1, y_2) = 1 \leftarrow \text{Eq}_*(x_1, y_1) = 1 \wedge \text{Eq}_*(x_2, y_2) = 1 .
\end{aligned}$$

Podobne ako v minulých príkladoch, pozitívne klauzuly z definície predikátu odpovedajú klauzulám s hodnotou 1 v definícií charakteristickej funkcie a negatívne klauzuly-klauzulám s hodnotou 0. Čiže

$$\text{Eq}_*(\bar{x}) = 1 \quad \text{odpovedá} \quad \text{Eq}(\bar{x})$$

a analogicky

$$Eq_*(\bar{x}) = 0 \quad \text{odpovedá} \quad \neg Eq(\bar{x}) .$$

Na základe tohto faktu dokážeme syntaktycky priamočiaro prepisovať definíciu predikátov na definície ich charakteristických funkcií a naopak. Stačí prepísať

$$\begin{array}{ll} p(\bar{x}) & \text{na} \quad p_*(\bar{x}) = 1 \\ \neg p(\bar{x}) & \text{na} \quad p_*(\bar{x}) = 0 \end{array}$$

a naopak.

Výpočet $Eq_*((0, 0), 0, 0)$:

$$\begin{aligned} Eq_*((0, 0), 0, 0) &= 1 \\ Eq_*(0, 0) &= 1 \wedge Eq_*(0, 0) = 1 \quad \leftarrow \text{preto} \end{aligned}$$

čiže $Eq((0, 0), 0, 0)$ je true.

Výpočet $Eq((0, 0), 0, 0, 0)$:

$$\begin{aligned} Eq_*((0, 0), 0, 0, 0) &= 1 \\ Eq_*(0, 0) &= 1 \wedge Eq_*(0, 0, 0) = 0 \quad \leftarrow \text{preto} \end{aligned}$$

čiže $Eq((0, 0), 0, 0, 0)$ je false.

7.0.7 Byť prvkom. Skúsme si ďalší predikát na zoznamoch: $x \varepsilon y - x$ je prvkom zoznamu y . Matematicky:

$$x \varepsilon y \leftrightarrow \exists z_1 \exists z_2 (z_1 \oplus (x, z_2) = y) .$$

Klauzálna definícia:

$$\begin{aligned} pred / 2 \varepsilon \\ x \varepsilon (v, y) &\leftarrow x = v \\ x \varepsilon (v, y) &\leftarrow x \neq v \wedge x \varepsilon y . \end{aligned}$$

Úplná definícia:

$$\begin{aligned} x \notin 0 \\ x \varepsilon (v, y) &\leftarrow x = v \\ x \varepsilon (v, y) &\leftarrow x \neq v \wedge x \varepsilon y \\ x \notin (v, y) &\leftarrow x \neq v \wedge x \notin y . \end{aligned}$$

Charakteristická funkcia:

$$\begin{aligned} fun / 2 \varepsilon_* \\ x \varepsilon_* 0 \\ x \varepsilon_* (v, y) &= 1 \leftarrow x = v \\ x \varepsilon_* (v, y) &= 1 \leftarrow x \neq v \wedge x \varepsilon_* y = 1 \\ x \varepsilon_* (v, y) &= 0 \leftarrow x \neq v \wedge x \varepsilon_* y = 0 . \end{aligned}$$

Skúsme si nijaký výpočet.

Výpočet $2 \varepsilon (1, 2, 3, 0)$:

$$\begin{aligned} 2 \varepsilon_*(1, 2, 3, 0) &= 1 \\ 2 \neq 1 \wedge 2 \varepsilon_*(2, 3, 0) &= 1 \quad \swarrow_{\text{preto}} \\ 2 &= 2 \quad \swarrow_{\text{preto}}, \end{aligned}$$

čiže $2 \varepsilon (1, 2, 3, 0)$ je true.

Výpočet pre $4 \varepsilon_* (1, 2, 3, 0)$:

$$\begin{aligned} 4 \varepsilon_*(1, 2, 3, 0) &= 0 \\ 4 \neq 1 \wedge 4 \varepsilon_*(2, 3, 0) &= 0 \quad \swarrow_{\text{preto}} \\ 4 \neq 2 \wedge 4 \varepsilon_*(3, 0) &= 0 \quad \swarrow_{\text{preto}} \\ 4 \neq 3 \wedge 4 \varepsilon_* 0 &= 0 \quad \swarrow_{\text{preto}}, \end{aligned}$$

čiže $4 \varepsilon_* (1, 2, 3, 0)$ je false.

Poznámka: predikát ε je v CL-ku už preddefinovaný, ASCII označenie: in pre ε , lin pre ℓ . *Prefix*(x, y) je začiatočným podreťazcom y . Matematicky:

$$\text{Prefix}(x, y) \leftrightarrow \exists z(x \oplus z = y) .$$

Klauzálne *Prefix*:

pred / 2 *Prefix*

Prefix(0, y)

Prefix((x_1, x_2), y_1, y_2) $\leftarrow x_1 = y_1 \wedge \text{Prefix}(x_1, y_2)$.

Úplná definícia:

Prefix(0, y)

$\neg \text{Prefix}((x_1, x_2), 0)$

$\neg \text{Prefix}((x_1, x_2), y_1, y_2) \leftarrow x_1 \neq y_1$

$\neg \text{Prefix}((x_1, x_2), y_1, y_2) \leftarrow x_1 = y_1 \wedge \neg \text{Prefix}(x_2, y_2)$

Prefix((x_1, x_2), y_1, y_2) $\leftarrow x_1 = y_1 \wedge \text{Prefix}(x_2, y_2)$.

Charakteristická funkcia:

fun / 2 *Prefix**

*Prefix**(0, y) = 1

*Prefix**((x_1, x_2), 0) = 0

*Prefix**((x_1, x_2), y_1, y_2) = 0 $\leftarrow x_1 \neq y_1$

*Prefix**((x_1, x_2), y_1, y_2) = 0 $\leftarrow x_1 = y_1 \wedge \text{Prefix}_*(x_2, y_2) = 0$

*Prefix**((x_1, x_2), y_1, y_2) = 1 $\leftarrow x_1 = y_1 \wedge \text{Prefix}_*(x_2, y_2) = 1$.

Výpočet *Prefix*((1, 2, 0), 1, 2, 3, 0):

$$\begin{aligned} \text{Prefix}_*((1, 2, 0), 1, 2, 3, 0) &= 1 \\ 1 &= 1 \wedge \text{Prefix}_*((2, 0), (2, 3, 0)) = 1 \quad \swarrow_{\text{preto}} \\ 2 &= 2 \wedge \text{Prefix}_*(0, 3, 0) = 1 \quad \swarrow_{\text{preto}} \end{aligned}$$

čiže $Prefix((1, 2, 0), 1, 2, 3, 0)$ je true.

Výpočet $Prefix((1, 3, 0), 1, 2, 3, 0)$:

$$Prefix_*((1, 3, 0), 1, 2, 3, 0) = 0$$

$$1 = 1 \wedge Prefix_*((3, 0), 2, 3, 0) = 0 \quad \swarrow_{\text{preto}}$$

$$3 \neq 2 \quad \swarrow_{\text{preto}}$$

čiže $Prefix((1, 3, 0), 1, 2, 3, 0)$ je false.

7.0.8 Suffix. $Suffix(x, y)$ - x je koncovým podreťazcom y . Matematicky:

$$Suffix(x, y) \leftrightarrow \exists z(z \oplus x = y) .$$

Klauzálné:

pred / 2 Suffix

$$Suffix(x, 0) \leftarrow x = 0$$

$$Suffix(x, y_1, y_2) \leftarrow x = y_1, y_2$$

$$Suffix(x, y_1, y_2) \leftarrow x \neq y_1, y_2 \wedge Suffix(x, y_2) .$$

Úplná definícia:

$$Suffix(x, 0) \leftarrow x = 0$$

$$\neg Suffix(x, 0) \leftarrow x \neq 0$$

$$Suffix(x, y_1, y_2) \leftarrow x = y_1, y_2$$

$$Suffix(x, y_1, y_2) \leftarrow x \neq y_1, y_2 \wedge Suffix(x, y_2)$$

$$\neg Suffix(x, y_1, y_2) \leftarrow x \neq y_1, y_2 \wedge \neg Suffix(x, y_2) .$$

Charakteristická funkcia $Suffix_*$:

*fun / 2 Suffix_**

$$Suffix_*(x, 0) = 1 \leftarrow x = 0$$

$$Suffix_*(x, 0) = 0 \leftarrow x \neq 0$$

$$Suffix_*(x, y_1, y_2) = 1 \leftarrow x = y_1, y_2$$

$$Suffix_*(x, y_1, y_2) = 1 \leftarrow x \neq y_1, y_2 \wedge Suffix_*(x, y_2) = 1$$

$$Suffix_*(x, y_1, y_2) = 0 \leftarrow x \neq y_1, y_2 \wedge Suffix_*(x, y_2) = 0 .$$

Výpočet $Suffix(2, 3, 0), 1, 2, 3, 0)$:

$$Suffix_*((2, 3, 0), 1, 2, 3, 0) = 1$$

$$2, 3, 0 \neq 1, 2, 3, 0 \wedge Suffix_*((2, 3, 0), 2, 3, 0) = 1 \quad \swarrow_{\text{preto}}$$

$$2, 3, 0 = 2, 3, 0 \quad \swarrow_{\text{preto}}$$

čiže $Suffix((2, 3, 0), 1, 2, 3, 0)$ je true.

Výpočet $Suffix((1, 3, 0), 1, 2, 3, 0)$:

$$Suffix_*((1, 3, 0), 1, 2, 3, 0) = 0$$

$$1, 3, 0 \neq 1, 2, 3, 0 \wedge Suffix_*((1, 3, 0), 2, 3, 0) = 0 \wedge \quad \swarrow_{\text{preto}}$$

$$1, 3, 0 \neq 2, 3, 0 \wedge Suffix_*((1, 3, 0), 3, 0) = 0 \quad \swarrow_{\text{preto}}$$

$$1, 3, 0 \neq 3, 0 \wedge Suffix_*((1, 3, 0), 0) = 0 \quad \swarrow_{\text{preto}}$$

$$1, 3, 0 \neq 0 \quad \swarrow_{\text{preto}}$$

čiže $Suffix((2, 3, 0), 1, 2, 3, 0)$ je false.

7.0.9 Podreťazec. $Substr(x, y)$ - x je podreťazcom y .

Matematicky:

$$Substr(x, y) \leftrightarrow \exists z_1, z_2 (z_1 \oplus x \oplus z_2 = y) .$$

Klauzálné:

pred / 2 *Substr*

$$Substr(x, 0) \leftarrow 0$$

$$Substr(x, y_1, y_2) \leftarrow P_{t}prefix(x, y_1, y_2)$$

$$Substr(x, y_1, y_2) \leftarrow \neg Prefix(x, y_1, y_2) \wedge Substr(x, y_2) .$$

Úplná definícia:

$$Substr(x, 0) \leftarrow x = 0$$

$$\neg Substr(x, 0) \leftarrow x \neq 0$$

$$Substr(x, y_1, y_2) \leftarrow Prefix(x, y_1, y_2)$$

$$\neg Substr(x, y_1, y_2) \leftarrow \neg Prefix(x, y_1, y_2) \wedge \neg Substr(x, y_2) .$$

Charakteristická funkcia:

fun / 2 *Substr*

$$Substr_*(x, 0) = 1 \leftarrow x = 0$$

$$Substr_*(x, 0) = 0 \leftarrow x \neq 0$$

$$Substr_*(x, y_1, y_2) = 1 \leftarrow Prefix(x, y_1, y_2)$$

$$Substr_*(x, y_1, y_2) = 1 \leftarrow \neg Prefix(x, y_1, y_2) \wedge Substr_*(x, y_2) = 1$$

$$Substr_*(x, y_1, y_2) = 0 \leftarrow \neg Prefix(x, y_1, y_2) \wedge Substr_*(x, y_2) = 0 .$$

Výpočet $Substr((1, 2, 0), 3, 4, 1, 2, 3, 0)$:

$$Substr_*((1, 2, 0), 3, 4, 1, 2, 3, 0) = 1$$

$$\neg Prefix((1, 2, 0), 3, 4, 1, 2, 3, 0) \wedge Substr_*((1, 2, 0), 4, 1, 2, 3, 0) = 1 \quad \swarrow \text{preto}$$

$$\neg Prefix((1, 2, 0), 4, 1, 2, 3, 0) \wedge Substr_*((1, 2, 0), 1, 2, 3, 0) = 1 \quad \swarrow \text{preto}$$

$$Prefix((1, 2, 0), 1, 2, 3, 0) \quad \swarrow \text{preto}$$

čiže $Substr((1, 2, 0), 3, 4, 2, 3, 0)$ je true.

Výpočet $Substr((1, 2, 0), 3, 4, 3, 0)$:

$$Substr_*((1, 2, 0), 3, 4, 3, 0) = 0$$

$$\neg Prefix((1, 2, 0), 3, 4, 3, 0) \wedge Substr_*((1, 2, 0), 4, 3, 0) = 0 \quad \swarrow \text{preto}$$

$$\neg Prefix((1, 2, 0), 4, 3, 0) \wedge Substr_*((1, 2, 0), 3, 0) = 0 \quad \swarrow \text{preto}$$

$$\neg Prefix((1, 2, 0), 3, 0) \wedge Substr_*((1, 2, 0), 0) = 0 \quad \swarrow \text{preto}$$

$$Prefix((1, 2, 0), 1, 2, 3, 0) \quad \swarrow \text{preto}$$

čiže $Substr((1, 2, 0), 3, 4, 3, 0)$ je false.

Kapitola 8

Unárne predikáty verzus n -árne

Podobne ako pri funkciách aj pomocou unárnych predikátov a párovacej funkcie, dokážeme vyjadriť n -árne predikáty. Nech p je n -árny predikát, budeme k nemu definovať unárnu kontrakciu $\langle p \rangle$ nasledovne:

$$\langle p \rangle (x) \leftrightarrow \exists x_1, \dots, x_n (x = \underbrace{x_1, \dots, x_n}_{\text{párovače}} \wedge p(\underbrace{x_1, \dots, x_n}_{\text{oddelovače}})) .$$

Potom platí:

$$\langle p \rangle (\underbrace{(x_1, \dots, x_n)}_{\text{párovače}}) \leftrightarrow p(\underbrace{x_1, \dots, x_n}_{\text{oddelovače}}) .$$

Ak p je unárny predikát tak potom $\langle p \rangle = p$. CL-ko umožňuje definovať ľubovoľné n -árne predikáty: $pred/n/p$; ak zapíšeme iba $pred/p$ tak je to skratka za $pred\ pred/1/p$. Konvencia: nech p je n -árny predikát a $m \geq n$ tak potom

$$p(x_1, \dots, x_{n-1}, x_n, x_{n+1}, \dots, x_m)$$

chápeme ako

$$p(\underbrace{x_1, \dots, x_{n-1}}_{\text{oddelovače}}, \underbrace{(x_n, x_{n+1}, \dots, x_m)}_{\text{párovače}}) ;$$

Ak $m < n$ tak ide o syntakticky chybný zápis.

1. argument je x_1

$n - 1$. argument je x_{n-1}

n . argument je $(x_n, x_{n+1}, \dots, x_m)$.

8.1 Preddefinované predikáty (formáty)

8.1.1 $N(x)$.

$N(x) \leftrightarrow x$ je prirodzené číslo

Kedže uvažujeme prirodzené čísla, vždy pravdivý.

Klauzálna definícia:

$N(0)$

$N(x+1) \leftarrow N(x)$.

Má bočný zobrazovací efekt: keď napíšeme v okienku **Query** $\tau=x:N$ (použije **N** ako formát), v okienku **Results** dostaneme

true for:

x=decimálna konštanta ,

čiže hodnota x -prirodzené číslo sa sformátuje a zobrazí do decimálnej konštanty.

8.1.2 $N2(x)$.

$N2(x) \leftrightarrow x$ je prirodzené číslo

Klauzálna definícia:

$N2(0)$

$N2(x_1) \leftarrow N2(x)$

$N2(x_2) \leftarrow N2(x)$,

kde

$S1(x) = 2x + 1 = x_1$

$S2(x) = 2x + 2 = x_2$.

Má bočný zobrazovací efekt:

- **Query:** $\tau=x:N2$

- **Results:**

true for:

x=diadická konštanta

Napríklad: $8=x:N2 \rightarrow x=0_{112}$.

8.1.3 $P(x)$.

$P(x) \leftrightarrow x$ je prirodzené číslo

Klauzálna definícia:

$P(0)$
 $P(x, y) \leftarrow P \wedge P(y)$.

Bočný zobrazovací efekt:

- Query: $\tau=x:P$
- Results:
 - true for:
 - x =párový, čiarkový numeral (čiarková konštanta)

Napríklad: $8=x:P \rightarrow x=((0,0),0),0$.

8.1.4 $M(x)$.

$M(x) \leftrightarrow x$ je prirodzené číslo

Klauzálna definícia:

$M(x)$.

Vždy platí, pre ľubovoľné x .

Bočný zobrazovací efekt:

- Query: $\tau=x:M$
- Results:
 - true for:
 - x =mixovaná čiarkovodefinovaná konštanta

Napríklad.: $8,7=x:M \rightarrow x=8,7$ alebo $8,S1S2(0)=x:M \rightarrow x=8,5$.

8.1.5 $Ch(x)$.

$Ch(x) \leftrightarrow x < 256$

Klauzálna definícia:

$Ch(x) \leftarrow x < 256$.

Bočný zobrazovací efekt:

- Query: $\tau=x:Ch$
- Results:
 - true for:
 - (ak hodnota $x < 256$) x ="znak s ASCII hodnotou x "

Napríklad: $100=x:Ch \rightarrow x="d"$ alebo $300=x:Ch \rightarrow x=?C?(300)$.

Napríklad: $"d"=x(:M) \rightarrow x=100$.

8.1.6 $Ln(x)$.

$Ln(x) \leftrightarrow x$ je zoznam
 x je prirodzené číslo

Klauzálna definícia:

$Ln(0)$
 $Ln(x, y) \leftarrow N(x) \wedge Ln(y)$.

Bočný zobrazovací efekt:

- Query: $\tau=x:Ln$

- Results:

true for:
 $x=x_1, x_2, \dots, x_k, 0$; kde x_i sú decimálne konštanty

Napríklad: $100=x:Ln \rightarrow x=0, 16, 0$ alebo $9=x:Ln \rightarrow x=0, 0, 0, 0, 0$.

8.1.7 $Str(x)$.

$Str(x) \leftrightarrow x$ je reťazec, zoznam charov $x = x_1, \dots, x_n, 0$; kde $x_i < 256$

Klauzálna definícia:

$Str(0)$
 $Str(x, y) \leftarrow Ch(x) \wedge Str(y)$.

Bočný zobrazovací efekt:

- Query: $\tau=x:Str$

- Results:

true for:
 $x = 'c_1 \dots c_k'$, kde $\tau=x:Ln \rightarrow x=x_1, \dots, x_k, 0$ a x_i je ASCII hodnota
pre znak c_i

Napríklad:

- $300, 0 = x:Str \rightarrow x=?S?(300, 0)$

- $100, 101, 102, 0 = x:Str \rightarrow 'def'$

- $'def' = x(:ln) (:M) \rightarrow 100, 101, 102, 0$

- konštantu- reťazec $'c_1 \dots c_k'$ môžeme používať, je skratkou za zoznam $x_1, \dots, x_k, 0$; kde x_i je ASCII hodnota c_i

- v reťazci môžete zadať znak c_i aj ako x_i , kde x_i je ASCII hodnota (3 miestna) c_i .
- V module `Standard` sú zadefinované štandardné definície. Dostane sa tam tak, že sa nastavíte na čiarku nad `incl Standard` a stlačíte F4 (rozbalíte aj zbalíte okno).
- môžeme rozbyť i zložené ciele: $\tau_1=x:F_1$ & $\tau_2=y:F_2 \dots$

8.1.8 Možnosť definovania predikátov. Môžeme si aj my definovať predikáty- formáty typu *Ln* či *Str*. Všeobecná schéma:

$Lt(0)$
 $Lt(x, y) \leftarrow T(x) \wedge Lt(y)$

kde T je nejaký predikát s bočným zobrazovacím efektom. Potom ak zadáme v Query $\tau=x:Lt$, dostaneme v Results

true for:
 $x=x_1:T, \dots, x_k:T, 0$; kde $x = x_1, \dots, x_k, 0$.

Napríklad:

$Ln2(0)$
 $Ln2(x, y) \leftarrow N2(x) \wedge Ln2(y)$

- $x:Ln2$ sa zapíše ako zoznam diadických konštánt
- $4,5,6,0 = x:Ln2 \leftarrow 0_{12}, 0_{21}, 0_{22}, 0$

$Lln(0)$
 $Lln(x, y) \leftarrow Ln(x) \wedge Lln(y)$

- $x:Lln$ sa zapíše ako zoznam, zoznamov (s decimálnymi konštantami).
- $4,5,6=x:Lln \rightarrow (0,0,0,0), (0,1,0), (1,0,0), 0$
- $1,2,4,9,0=x:Lln \rightarrow (0,0), (0,0,0), (0,0,0,0), (0,0,0,0,0), 0$

$Lln2(0)$
 $Lln2(x, y) \leftarrow Ln2(x) \wedge Lln2(y)$

- $x:Lln2$ sa zapíše ako zoznam, zoznamov s diadickými konštantami.
- $(4,5,0), (6,7,0)=x:Lln2 \rightarrow (0_{12}, 0_{21}, 0), (0_{22}, 0_{111}, 0), 0$

$Lstr(0)$
 $Lstr(x, y) \leftarrow Str(x) \wedge Lstr(y)$

- $x:Lstr$ sa zapíše ako zoznam reťazcov
- $(97,98,0), (99,100,0), 0=x:Lstr \rightarrow 'ab', 'cd', 0$

Taktiež môžeme definovať i kartézsky súčin typov:

$$R(x_1, \dots, x_n) \rightarrow T_1(x_1) \wedge \dots \wedge T_n(x_n),$$

kde T_i je nijaký predikát s bočným zobrazovacím efektom. Potom ak zadáme v Query $T = x : R$ dostaneme v Results

true for:

$x_1 : T_1, \dots, x_k : T_k$; kde $x = x_1, \dots, x_k$.

Napríklad:

$$Nch(x, y) \leftarrow N(x) \wedge Ch(x)$$

99,100=x:Nch→99,"d"

$$Nchd(x, y, z) \leftarrow N(x) \wedge Ch(y) \wedge N2(z)$$

99,100,5=x:Nchd→99,"d",0₂₁ .

Kapitola 9

Množiny

Na mimulých prednáškach sme si ukázali ako implementovať, kódovať zoznamy, zložitejší dátový typ, pomocou prirodzených čísel a párovacej funkcie " ". Teraz sa budeme zaoberať ďalším typom - konečnými množinami prirodzených čísel. Budeme ich reprezentovať pomocou ostro usporiadaných zoznamov. Číslo $a = x_1, \dots, x_n, 0$, kde $x_1 < x_2 < \dots < x_n$, bude kódom konečnej množiny prirodzených čísel $A = \{x_1, \dots, x_n\}$. Predikát $Ord(x)$, ktorý platí ak x je ostro usporiadaný zoznam, má nasledovnú definíciu.

Matematicky:

$$Ord(x) \leftrightarrow \forall y, a, b, z (x = y \oplus (a, b, z) \rightarrow a < b) .$$

Klauzálné:

$$Ord(0)$$

$$Ord(x, 0)$$

$$Ord(x, y, z) \leftarrow x < y \wedge Ord(y, z) .$$

Za množinu budeme považovať ostro usporiadnaý zoznam. Matematicky

$$Set(x) \leftrightarrow Ord(x) .$$

Klauzálné

$$Set(x) \leftarrow Ord(x) .$$

9.1 Operácie na množinách

Podme si naprogramovať základné operácie a predikáty na množinách.

9.1.1 Prázdna množina. Matematicky:

$$Set \rightarrow Empty(x) \leftrightarrow x = 0 .$$

Klauzálné:

$$Empty(0) .$$

9.1.2 Byť prvkom nožiny. Matematicky:

$$Set \rightarrow x \in y \leftrightarrow x \varepsilon y .$$

Klauzálné:

$$x \in y \leftarrow x \varepsilon y .$$

Bez ε využijúc usporiadanosť y :

$$x \in (y_1, y_2) \leftarrow x = y_1$$

$$x \in (y_1, y_2) \leftarrow x > y_1 \wedge x \in y_2 .$$

9.1.3 Rovnosť dvoch množín.

1. pomocou zabudovaného =

Matematicky:

$$Set(x) \wedge Set(y) \rightarrow x \equiv y \leftrightarrow x = y .$$

Klauzálné:

$$x \equiv y \leftarrow x = y .$$

2. využívajúc vlastnosť- axiómu množín, že dve množiny sa rovnajú, ak majú rovnaké prvky:

Matematicky:

$$Set(x) \wedge Set(y) \rightarrow x \equiv y \leftrightarrow \forall z(z \in x \leftrightarrow z \in y) .$$

Klauzálné (využijeme usporiadanosť zoznamov reprezentajúcich množiny):

$$0 \equiv 0$$

$$(x_1, x_2) \equiv (y_1, y_2) \leftarrow x_1 = y_1 \wedge x_2 \equiv y_2 .$$

9.1.4 Podmnožina. Matematicky:

$$Set(x) \wedge Set(y) \rightarrow x \leq y \leftrightarrow \forall z(z \in x \rightarrow z \in y) .$$

Klauzálné:

$$0 \subseteq (y_1, y_2) \leftarrow x_1 = y_1 \wedge x_2 \subseteq y_2$$

$$(x_1, x_2) \subseteq (y_1, y_2) \leftarrow x_1 > y_1 \wedge (x_1, x_2) \subseteq y_2 .$$

9.1.5 Prienik dvoch množín. Matematicky:

$$Set(x) \wedge Set(y) \rightarrow Set(x \cap y)$$

$$Set(x) \wedge Set(y) \rightarrow \forall z(z \in x \cap y \leftrightarrow z \in x \wedge z \in y) .$$

Klauzálné:

$$0 \cap y = 0$$

$$(x_1, x_2) \cap 0 = 0$$

$$(x_1, x_2) \cap (y_1, y_2) = x_1, x_2 \cap y_2 \leftarrow x_1 = y_1$$

$$(x_1, x_2) \cap (y_1, y_2) = x_2 \cap (y_1, y_2) \leftarrow x_1 < y_1$$

$$(x_1, x_2) \cap (y_1, y_2) = (x_1, x_2) \cap y_2 \leftarrow x_1 > y_1 .$$

9.1.6 Rozdiel dvoch množín. Matematicky:

$$\begin{aligned} & Set \wedge Set(y) \rightarrow Set(x \setminus y) \\ & Set \wedge Set(y) \rightarrow \forall z(z \in x \setminus y \leftrightarrow z \in x \wedge z \in y) . \end{aligned}$$

Klauzálna definícia:

$$\begin{aligned} 0 \setminus y &= 0 \\ (x_1, x_2) \setminus 0 &= x_1, x_2 \\ (x_1, x_2) \setminus (y_1, y_2) &= x_2 \setminus y_2 \leftarrow x_1 = y_1 \\ (x_1, x_2) \setminus (y_1, y_2) &= x_1, x_2 \setminus (y_1, y_2) \leftarrow x_1 < y_1 \\ (x_1, x_2) \setminus (y_1, y_2) &= (x_1, x_2) \setminus y_2 \leftarrow x_1 > y_1 . \end{aligned}$$

9.1.7 Zjednotenie dvoch množín. Matematicky:

$$\begin{aligned} & Set(x) \wedge Set(y) \rightarrow Set(x \cup y) \\ & Set(x) \wedge Set(y) \rightarrow \forall z(z \in x \cup y \leftrightarrow z \in x \vee z \in y) . \end{aligned}$$

Klauzálna:

$$\begin{aligned} 0 \cup y &= 0 \\ (x_1, x_2) \cup 0 &= x_1, x_2 \\ (x_1, x_2) \cup (y_1, y_2) &= x_1, x_2 \cup y_2 \leftarrow x_1 = y_1 \\ (x_1, x_2) \cup (y_1, y_2) &= x_1, x_2 \cup (y_1, y_2) \leftarrow x_1 < y_1 \\ (x_1, x_2) \cup (y_1, y_2) &= y_1, (x_1, x_2) \cup y_2 \leftarrow x_1 > y_1 . \end{aligned}$$

9.1.8 Symetrická definícia dvoch množín. Matematická definícia:

$$\begin{aligned} & Set(x) \wedge Set(y) \rightarrow Set(x \Delta y) \\ & Set(x) \wedge Set(y) \rightarrow x \Delta y = (x \setminus y) \cup (y \setminus x) . \end{aligned}$$

Klauzálna definícia:

$$\begin{aligned} 0 \Delta y &= y \\ (x_1, x_2) \Delta 0 &= x_1, x_2 \\ (x_1, x_2) \Delta (y_1, y_2) &= x_2 \Delta y_2 \leftarrow x_1 = y_1 \\ (x_1, x_2) \Delta (y_1, y_2) &= x_1, x_2 \Delta (y_1, y_2) \leftarrow x_1 < y_1 \\ (x_1, x_2) \Delta (y_1, y_2) &= y_1, (x_1, x_2) \Delta \leftarrow x_1 > y_1 . \end{aligned}$$

9.1.9 Disjunktné množiny. Matematická definícia:

$$Set(x) \wedge Set(y) \rightarrow \forall x \diamond y \leftrightarrow x \cap y = 0$$

alebo

$$Set(x) \wedge Set(y) \rightarrow \forall x \diamond y \leftrightarrow x \cup y = x \Delta y .$$

Vieme, že $x \cup y \supseteq x \Delta y$. Klauzálna definícia:

$$\begin{aligned} 0 \diamond y &= 0 \\ (x_1, x_2) \diamond 0 &= 0 \\ (x_1, x_2) \diamond (y_1, y_2) &\leftarrow x_1 < y_1 \wedge x_2 \diamond (y_1, y_2) \\ (x_1, x_2) \diamond (y_1, y_2) &\leftarrow x_1 > y_1 \wedge (x_1, x_2) \diamond y_2 . \end{aligned}$$

Kapitola 10

Kombinatorické úlohy na zoznamoch

10.1 Základné operácie

10.1.1 Interleave. Skúsme si zdefinovať funkciu $Interleave(x, y)$; kde x je nejaký prvok; y je zoznam; ktorá nám vráti zoznam všetkých vložení prvku x do zoznamu y . Napríklad:

$$Interleave(1, 2, 3, 0) = (1, 2, 3, 0), (2, 1, 3, 0), (2, 3, 1, 0), 0.$$

Matematická špecifikácia:

$$z \in Interleave(x, y) \leftrightarrow \exists y_1 \exists y_2 (y = y_1 \oplus y_2 \wedge z = y_1 \oplus (x, y_2)) .$$

Klauzálna definícia:

$$\begin{aligned} Interleave(x, 0) &= (x, 0), 0 \\ Interleave(x, y_1, y_2) &= (x, y_1, y_2), Map_1(y_1, Interleave(x, y_2)) , \end{aligned}$$

kde

$$\begin{aligned} Map_1(x, 0) &= 0 \\ Map_1(x, y_1, y_2) &= (x, y_1), Map_1(x, y_2) . \end{aligned}$$

10.1.2 Subsequence. Zdefinujme si predikát $Subsequence$, ktorý platí ak zoznam x je podpostupnosťou (vybranou postupnosťou) zoznamy y . Matematická definícia:

$$\begin{aligned} Subsequence(x, y) \leftrightarrow \exists i (Ord(i) \wedge L(i) = L(x) \wedge i \neq 0 \rightarrow Last(i) < L(y) \wedge \\ Maxl(i) < L(y) \wedge \\ \forall j < L(x) (Take(j, x) = Take(Take(j, i), y))) . \end{aligned}$$

Čiže existuje korektná rastúca postupnosť indexov i taká, že

$$\forall j < L(i) : x_j = y_j .$$

Klauzálna definícia:

$$\text{Subsequence}(0, y)$$

$$\text{Subsequence}((x_1, x_2), y_1, y_2) \leftarrow x_1 = y_1 \wedge \text{Subsequence}(x_2, y_2)$$

$$\text{Subsequence}((x_1, x_2), y_1, y_2) \leftarrow x_1 \neq y_1 \wedge \text{Subsequence}((x_1, x_2), y_2) .$$

10.1.3 Subseqlist. Teraz skúsme pre daný zoznam y vytvoriť zoznam všetkých podpostupností y -nu.

Matematicky:

$$x \in \text{Subseqlist}(y) \leftrightarrow \text{Subseqlist}(x, y) .$$

Klauzálna:

$$\text{Subseqlist}(0) = 0, 0$$

$$\text{Subseqlist}(y_1, y_2) = \text{Map}_2(y_1, \text{Subseqlist}(y_2)) ,$$

kde

$$\text{Map}_2(x, 0) = 0$$

$$\text{Map}_2(x, y_1, y_2) = (x, y_1), y_1, \text{Map}_2(x, y_2) .$$

10.1.4 Zoznam všetkých k -prvkových podpostupností y -nu. k -prvková podpostupnosť x postupnosti y je definovaná nasledovne:

$$\text{Subsequence}_k(k, x, y) \leftrightarrow \text{Subsequence}(x, y) \wedge L(x) = k .$$

Klauzálna:

$$\text{Subsequence}_k(k, x, y) \leftarrow L(x) = k \wedge \text{Subsequence}_k$$

$$\text{Subsequence}_k(k + 1, (x_1, x_2), y_1, y_2) \leftarrow x_1 = y_1 \wedge \text{Subsequence}_k(k, x_2, y_2)$$

$$\text{Subsequence}_k(k + 1, (x_1, x_2), y_1, y_2) \leftarrow x_1 \neq y_1 \wedge \text{Subsequence}_k(k, (x_1, x_2), y_2) .$$

Skúsme pre dané k a zoznam y vytvoriť zoznam všetkých k -prvkových podpostupností y -nu- $\text{Subsequence}_k(k, y)$.

Matematicky:

$$x \in \text{Subseqlist}_k(k, y) \leftrightarrow \text{Subsequence}_k(k, x, y) .$$

Klauzálna:

$$\text{Subseqlist}_k(0, y) = 0, 0$$

$$\text{Subseqlist}_k(k + 1, 0) = 0$$

$$\text{Subseqlist}_k(k + 1, y_1, y_2) = \text{Map}_3(y_1, \text{Subseqlist}_k(k, y_2) \oplus \text{Subseqlist}_k(k + 1, y_2)$$

$$\text{Map}_3(x, 0) = 0$$

$$\text{Map}_3(x, y_1, y_2) = (x, y_1), \text{Map}_3(x, y_2) .$$

10.1.5 Partition. Zdefinujme si predikát $Partition(x, y)$, ktorý platí ak $x = x_1, \dots, x_n, 0$ ($x_i \neq 0$) a $x_1 \oplus \dots \oplus x_n = y$. Čiže x je akýsi "rozsekanie" zoznamu y na neprázdne časti x_i , ktoré dokopy po konkatenácii dávajú opäť zoznam y . Matematicky:

$$Partition(x, y) \leftrightarrow Union(x) = y \wedge 0 \notin x ,$$

kde

$$\begin{aligned} Union(0) &= 0 \\ Union(x_1, x_2) &= x_1 \oplus Union(x_2) . \end{aligned}$$

Skúsme si teraz zdefinovať funkciu $Parts(y)$, ktorá nám vráti zoznam všetkých "rozsekaní" y -nu na neprázdne časti. Matematicky:

$$x \in Parts(y) \leftrightarrow Partition(x, y) .$$

Klauzálné:

$$\begin{aligned} Parts &= 0, 0 \\ Parts(y_1, y_2) &= Map_4(y_1, Parts(y_2)) , \end{aligned}$$

kde

$$\begin{aligned} Map_4(x, 0) &= 0 \\ Map_4(x, 0, y_2) &= ((x, 0), 0), Map_4(x, y_2) \\ Map_4(x, (y_{11}, y_{12}), y_2) &= ((x, 0), y_{11}, y_{12}), ((x, y_{11}), y_{12}), Map_4(x, y_2) \end{aligned}$$

$$\begin{aligned} Parts(0) &= 0, 0 \\ Parts(y_1, 0) &= ((y_1, 0), 0) \\ Parts(y_1, y_2, y_3) &= Map_4(y_1, Parts(y_2, y_3)) . \end{aligned}$$

Pripomenme, že konečná postupnosť prirodzených čísel 0 dĺžke $n \geq 0$ je unárna funkcia tvaru $f : \{0, \dots, n-1\} \rightarrow \mathbb{N}$. Konečnú postupnosť prirodzených čísel x_0, \dots, x_{n-1} o dĺžke n môžeme priamo kódovať číslom, zoznamom $\bar{x} = x_0, \dots, x_{n-1}, 0$ ($L = n$). Prázdnu postupnosť, $n = 0$, je teda kódovaná prázdny zoznamom.

10.1.6 Permutácia. Permutáciou o dĺžke n budeme rozumieť konečnú postupnosť, ktorá je bijekciou na množine $\{0, 1, \dots, n-1\}$. Skúsme si zdefinovať funkciu $Perms(n)$, ktorá vráti zoznam všetkých permutácií o dĺžke n :

$$\begin{aligned} Perms(0) &= 0, 0 \\ Perms(n+1) &= Map_5(n, Perms(n)) \\ Map_5(x, 0) &= 0 \\ Map_5(x, y_1, y_2) &= Interleave \oplus Map_5(x, y_2) . \end{aligned}$$

10.2 Triedenia

Teraz sa budeme zaoberať skupinou funkcií, ktoré nám dokážu vstupný neusporiadaný zoznam usporiadať podľa veľkosti prvkov-čísel.

Najskôr si zadefinujeme predikát $Ordl(x)$, ktorý platí ak zoznam x je (neostro) usporiadaný:

Matematická definícia:

$$Ordl(x) \leftrightarrow \forall y, a, b, z (x = y \oplus (a, b, z) \rightarrow a \leq b) .$$

Klauzálna definícia:

$$Ordl(0)$$

$$Ordl(x, 0)$$

$$Ordl(x, y, z) \leftarrow x \leq y \wedge Ordl(y, z) .$$

Okrem usporiadanosti od výstupného zoznamu budeme ešte vyžadovať, aby bol preusporiadaním- permutáciou vstupného zoznamu. Definujeme si predikát $Perm(x, y)$, ktorý platí ak zoznam x je permutáciou zoznamu y .
matematicky:

$$Perm(0, y) \leftrightarrow y = 0$$

$$Perm((x_1, x_2), y) \leftrightarrow \exists y_1, y_2 (y = y_1 \oplus (x_1, y_2) \wedge Perm(x_2, y_1 \oplus y_2)) .$$

Z toho dostávame nasledovnú klauzálnu definíciu:

$$Perm(0, 0)$$

$$Perm((x_1, x_2), y) \leftarrow Mem(x_1, y) = y_1, y_2 \wedge Perm(x_2, y_1 \oplus y_2) ,$$

kde pomocná funkcia $Mem(x, y)$ má nasledovné vlastnosti:

$$x \varepsilon y \rightarrow \exists y_1, y_2 (Mem(x, y) = y_1, y_2 \wedge y = y_1 \oplus x, y_2)$$

$$x \notin y \rightarrow Mem(x, y) = 0 .$$

Klauzálna definícia:

$$Mem(x, 0) = 0$$

$$Mem(x, y_1, y_2) = 0, y_2 \leftarrow x = y_1$$

$$Mem(x, y_1, y_2) = (y_1, v_1), v_2 \leftarrow x \neq y_1 \wedge Mem(x, y_2) = v_1, v_2 .$$

Teraz môžeme triedenia formálnejšie charakterizovať. Funkcia-triedenie $Sort(x)$ musí spĺňať tieto dve vlastnosti:

1. $Ordl(Sort(x))$
2. $Perm(x, Sort(x))$

V ďalšej časti budeme implementovať 3 triediace metódy: *Insertsort*, *Mergesort* a *Quicksort*.

10.2.1 Insertsort. Táto metóda triedania ja založená na vkladání prvku do už usporiadaného zoznamu, tak aby usporiadanie nového zoznamu ostalo zachované. Môžeme si zdefinovať pomocnú funkciu $Insert(x, y)$, ktorá vloží prvok x do usporiadaného zoznamu y a zachová usporiadanie. Matematicky by sme ju mohli charakterizovať týmito vlastnosťami:

1. $Ordl(y) \rightarrow Ordl(Insert(x, y))$
2. $Ordl(y) \rightarrow Perm((x, y), Insert(x, y))$.

Klauzálna definícia:

$$\begin{aligned} Insert(x, 0) &= x, 0 \\ Insert(x, y_1, y_2) &= x, y_1, y_2 \leftarrow x \leq y_1 \\ Insert(x, y_1, y_2) &= y_1, Insert(x, y_2) \leftarrow x > y_1 . \end{aligned}$$

Celé triedenie vyzerá potom tak, že najskôr utriedime rekurzívne zvyšok vstupného zoznamu bez prvého prvku a ten potom vložíme (pomocou funkcie $Insert$) do už utriedaného zvyšku zoznamu.

$$\begin{aligned} Inert(0) &= 0 \\ Insert(x, y) &= Insert(x, Insert(y)) . \end{aligned}$$

10.2.2 Mergesort. Triedenie je založené na nasledovnom princípe:

1. vstupný zoznam sa rozdelí na dva približne rovnake dlhé podzoznamy, t.j. ich dĺžky sa rovnajú alebo líšia o jedna.
2. tie sa rekurzívne utriedia
3. a nakoniec spoja - *merge* do výsledného usporiadaného zoznamu.

1.krok sa realizuje pomocou funkcie $Msplit(x)$, ktorú môžeme nasledovne formalizovať:

$$\begin{aligned} \exists y, z (Msplit(x) = y, z \wedge L(x) = L(y) + L(z) \wedge \\ (L(y) = L(z) \vee L(y) = L(z) + 1) \wedge \\ \forall i(2 \cdot i < L(x) \rightarrow Take(2 \cdot i, x) = Take(i, y)) \wedge \\ \forall i(2 \cdot i + 1 < L(x) \rightarrow Take(2 \cdot i + 1, x) = Take(i, z))) . \end{aligned}$$

Klauzálna definícia:

$$\begin{aligned} Msplit(0) &= 0, 0 \\ Msplit(x, 0) &= (x, 0), 0 \\ Msplit(x, y, z) &= (x, v_1), y, v_2 \leftarrow Msplit(z) = v_1, v_2 . \end{aligned}$$

2.krok spojenie(*mergovanie*) dvoch usporiadaných zoznamov do opäť utriedeného zoznamu naimplementujeme pomocou funkcie *Merge*, ktorá má nasledovné vlastnosti:

1. $Ordl(x) \wedge \leftarrow Ordl(y) \rightarrow Ordl(Merge(x, y))$,
2. $Perm(x \oplus y, Merge(x, y))$.

Klauzálna definícia:

$$\begin{aligned}
 Merge(0, y) &= y \\
 Merge((x_1, x_2), 0) &= x_1, x_2 \\
 Merge((x_1, x_2), y_1, y_2) &= x_1, y_1, Merge(x_2, y_2) \leftarrow x_1 = y_1 \\
 Merge((x_1, x_2), y_1, y_2) &= y_1, Merge((x_1, x_2), y_2) \leftarrow x_1 > y_1 \\
 Merge((x_1, x_2), y_1, y_2) &= x_1, Merge(x_2, (y_1, y_2)) \leftarrow x_1 < y_1 .
 \end{aligned}$$

Celé triedenie môžeme zdefinovať nasledovne:

$$\begin{aligned}
 Msort(0) &= 0 \\
 Msort(x, 0) &= x, 0 \\
 Msort(x, y, z) &= Merge(Msort(v_1), Msort(v_2)) \leftarrow Msplit(x, y, z) = v_1, v_2 .
 \end{aligned}$$

10.2.3 Quicksort. Triedenie ja postavené na nasledujúcej metóde:

1. zo vstupného zoznamu vyberieme jeden prvok-*pivot* (v našej implementácii sa obmedzíme na prvý prvok zoznamu pre jednoduchosť),
2. zoznam rozdelíme na dva podzoznamy, v prvom podzozname budú prvky menšie alebo rovné ako pivot a v druhom zase väčšie,
3. podzoznamy rekurzívne utriedime a spojíme-*skonkatenujeme* do výsledného zoznamu.

1. a 2. krok zrealizujeme pomocou funkcie $Qsplit(p, x)$, kde p -pivot je prvým prvkom zoznamu a x jeho zvyškom. Funkcia má tieto vlastnosti:

$$\begin{aligned}
 \exists y, z(Qsplit(p, x) = y, z \wedge \forall v(v \in y \rightarrow v \leq p) \wedge \\
 \forall v(v \in z \rightarrow v > p) \wedge Perm(x, y \oplus z)) .
 \end{aligned}$$

Klauzálna definícia:

$$\begin{aligned}
 Qsplit(p, 0) &= 0, 0 \\
 Qsplit(p, x, y) &= (x, v_1), v_2 \leftarrow x \leq p \wedge Qsplit(p, y) = v_1, v_2 \\
 Qsplit(p, x, y) &= v_1, x, v_2 \leftarrow x > p \wedge Qsplit(p, y) = v_1, v_2 .
 \end{aligned}$$

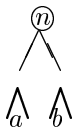
Celé triedenie vyzerá nasledovne:

$$\begin{aligned}
 Qsort(0) &= 0 \\
 Qsort(x, y) &= Qsort(v_1) \oplus (x, Qsort(v_2)) \leftarrow Qsort(x, y) = v_1, v_2 .
 \end{aligned}$$

Kapitola 11

Binárne stromy

Zoznámime sa ďalšou dátovou štruktúrou - binárnymi stromami. Pod binárnym

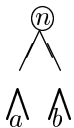
stromom budeme schematicky rozumieť nasledovnú štruktúru: , kde n je

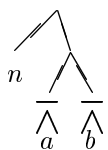
vrchol(koreň) s hodnotou $n \in \mathbb{N}$ a A je ľavý binárny podstrom, B je pravý binárny podstrom. Prázdny binárny strom môžeme značiť ako \bullet .

11.1 Kódovanie binárnych stromov


Otázka znie, ako budeme kódovať binárne stromy do prirodzených čísel?

11.1.1 1.spôsob. Prvý jednoduchý spôsob, ktorý nás ihneď napadne, je nasle-

dovný. Prázdny binárny strom \bullet zakódujeme 0 a binárny strom tvaru  bu-

deme kódovať pomocou trojice:  = n, \bar{A}, \bar{B} , kde \bar{A}, \bar{B} sú kódy pre

binárne podstromy A, B .

Napríklad: binárny strom  zakódujeme do $2, (1, 0, 0), 0 = 382$.

Môžeme si zdefinovať predikát byť binárnym stromom:

$$Bt(0)$$

$$Bt(n, a, b) \leftarrow N(n) \wedge Bt(a) \wedge Bt(b) .$$

Ďalej si skúsme funkciu *Size*, ktorá nám zistí počet vnútorných vrcholov binárneho stromu t :

$$Size(0) = 0$$

$$Size(n, a, b) = Size(a) + Size(b) + 1$$

alebo predikát *Memb*(x, t), ktorý platí ak v t sa nachádza vrchol s hodnotou x :

$$Memb(x, y, a, b) \leftarrow x = y$$

$$Memb(x, y, a, b) \leftarrow x \neq y \wedge Memb(x, a)$$

$$Memb(x, y, a, b) \leftarrow x \neq y \wedge \neg Memb(x, a) \wedge Memb(x, b) .$$

11.1.2 2.spôsob. Druhý spôsob kódovania binárnych stromov je o niečo zložitejší, ale na druhej strane všeobecnejší. Vo funkcionálnom programovaní sa používajú typové (aj rekurzívne) rovnice na špecifikáciu nejakého typu. Napríklad typ *výraz-expression* môžeme definovať nasledovne.

Expression je buď:

1. konštanta $\in C(N)$
2. premenná $\in V(N)$
3. $Add(exp_1, exp_2)$
4. $Mul(exp_1, exp_2)$

a zapíšeme pomocou typovej rovnice

$$Exp = C(\mathbb{N}) | V(\mathbb{N}) | Add(Exp, Exp) | Mul(Exp, Exp) ,$$

kde položky oddelené zvislou čiarou nazývame varianty a C, V, Add, Mul voláme *tagy* (označenia, nálepky, značky), pre konštruktory (funkcie).

Náš typ binárny strom môžeme definovať nasledovne.

Binárny strom je buď:

1. prázdny strom, značme ako E
2. tvaru $Nd(N, t_1, t_2)$,

a zapísať pomocou typovej rovnice:

$$Bt = E|Nd(N, Bt, Bt) .$$

Význam typových rovníc môžeme definovať algebraicky:

1. Exp je najmenšia množina, ktorá spĺňa:

$$\begin{aligned} C(n) &= \{C(i) | i \in \mathbb{N}\} \subseteq Exp \\ V(n) &= \{V(i) | i \in \mathbb{N}\} \subseteq Exp \\ exp_1, exp_2 \in Exp &\rightarrow Add(exp_1, exp_2) \in Exp \\ exp_1, exp_2 \in Exp &\rightarrow Mul(exp_1, exp_2) \in Exp . \end{aligned}$$

Dá sa ukázať, že takto je Exp definovaná jednoznačne: Nech Exp_1, Exp_2 sú dve také množiny, ktoré spĺňajú vyššie uvedené podmienky, potom $Exp_1 \subseteq Exp_2$ (Exp_1 je najmenšia taká) a $Exp_1 \supseteq Exp_2$ (Exp_2 je najmenšia taká), čiže $Exp_1 = Exp_2$. Podobne pre typ Bt .

2. Bt je najmenšia množina, ktorá spĺňa:

$$\begin{aligned} E &\in Bt \\ t_1, t_2 \in Bt \wedge n \in \mathbb{N} &\rightarrow Nd(n, t_1, t_2) \in Bt . \end{aligned}$$

Náš druhý spôsob kódovania binárnych stromov bude teda vychádzať z typových rovníc. Všimnime si, že každý variant z typovej rovnici začína nejakým *tagom*-značkou, nálepkou pre konštruktor

- v prvom prípade C, V, Add, Mul ;
- v druhom prípade E, Nd ; pričom za tagom môže byť ďalšie parametre, vtedy ide o tag pre konštruktor - funkcie (Nd), alebo nemusia, vtedy ide o tag pre konštruktor- konštantu (E).

V Cl budeme konštruktory- konštanty definovať kódovať ako pár

$$Const = t, 0 \quad (0, 0; 1, 0; 2, 0; 3, 0 \dots)$$

a konštruktory- funkcie ako pár

$$F(x) = t, x \quad (0, x; 1, x; 2, x; 3, x \dots) ,$$

kde $t \in \mathbb{N}$ bude vlastne tag pre daný konštruktor. (Definície sú farebne odlišne, $Const, F$ sú zelené.) Pre náš prvý prípad by sme mohli zdefinovať konštruktory:

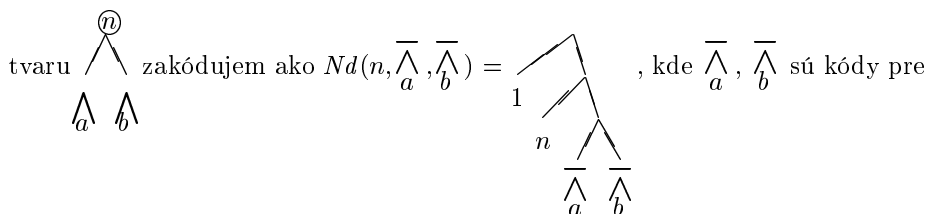
$$\begin{aligned} C(x) &= 0, x & Add(x) &= 2, x \\ V(x) &= 1, x & Mul(x) &= 3, x \end{aligned}$$

(0, 1, 2, 3 - sú ich tagy).

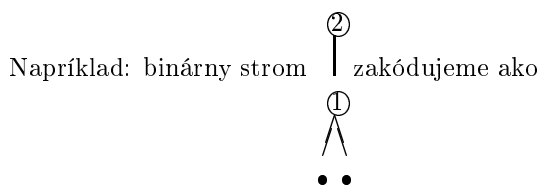
Pre binárne stromy použijeme:

- konštruktor - konštantu $E = 0, 0$ (pre prázdny strom) a
- konštruktor - funkciu $Nd = 1, x$.

Výsledné kódovanie bude vyzerat' nasledovne: Prázdny binárny strom • zakódujeme pomocou konštruktora- konštanty $E = 0, 0 = \bigwedge_{0\ 0}$. Binárny strom



bin. podstromy \bigwedge_a, \bigwedge_b .



$$Nd(2, Nd(1, E, E), E) = 1, 2, (1, (0, 0), 0, 0), 0, 0 = 1855045 .$$

Môžeme si zdefinovať predikát byť binárnym stromom:

$$Bt(E)$$

$$Bt(Nd(n, a, b)) \leftarrow N(n) \wedge Bt(a) \wedge Bt(b) .$$

Malá poznámka ku formátom: Okrem formátov kartézsky súčin a zoznamová schéma môžeme používať definovať predikáty- formáty s konštruktormi. Jeden z príkladov takých formátov je definícia Bt (Bt bude vysvietení žltou). Zobrazovací efekt bude taký, že výsledný tvar termu sa zobrazí pomocou konštruktov. Napríklad:

$$1, 2, (1, (0, 0), 0, 0), 0, 0 = x : Bt$$

$$x = Nd(2, Nd(1, E, E), E)$$

$$1855045 = x : Bt$$

$$x = Nd(2, Nd(1, E, E), E)$$

11.2 Funkcie a predikáty na binárnych stromoch

Podme si teraz zdefinovať nejaké ďalšie predikáty a funkcie na binárnych stromoch

11.2.1 Member. Matematicky:

$$\begin{aligned} \text{Size}(E) &= 0 \\ \text{Size}(Nd(n, l, r)) &= \text{Size}(l) + \text{Size}(r) + 1 . \end{aligned}$$

Klauzálné:

$$\begin{aligned} \text{Memb}(x, Nd(y, l, r)) &\leftarrow x = y \\ \text{Memb}(x, Nd(y, l, r)) &\leftarrow x \neq y \wedge \text{Memb}(x, l) \\ \text{Memb}(x, Nd(y, l, r)) &\leftarrow x \neq y \wedge \neg \text{Memb}(x, l) \wedge \text{Memb}(x, r) . \end{aligned}$$

11.2.2 Inorder. Ďalej si zdefinujeme funkciu, ktorá nám vráti pre daný binárny strom zoznam vnútorných vrcholov v inorderovom usporiadaní:

$$\begin{aligned} \text{Inorder}(E) &= 0 \\ \text{Inorder}(Nd(x, l, r)) &= \text{Inorder}(l) \oplus (x, \text{Inorder}(r)) . \end{aligned}$$

11.2.3 Update. Funkcia $\text{Update}(x, y, t)$ vráti strom t' , ktorý vznikne z t nahradením u všetkých vrcholov x s hodnotou y .

$$\begin{aligned} \text{Update}(x, y, E) &= E \\ \text{Update}(x, y, Nd(z, l, r)) &= Nd(y, \text{Update}(x, y, l), \text{Update}(x, y, r)) \leftarrow z = x \\ \text{Update}(x, y, Nd(z, l, r)) &= Nd(z, \text{Update}(x, y, l), \text{Update}(x, y, r)) \leftarrow z \neq x . \end{aligned}$$

11.3 Binárne vyhľadávacie stromy

Binárne vyhľadávacie stromy budú špeciálnym podtypom binárnych stromov. Matematicky ich môžeme definovať (pomocou Memb) takto:

$$\begin{aligned} \text{Bst}(E) \\ \text{Bst}(Nd(x, l, r)) &\leftrightarrow N(x) \wedge \forall z (\text{Memb}(z, l) \rightarrow z < x) \wedge \\ &\forall z (\text{Memb}(z, r) \rightarrow z > x) \wedge \text{Bst}(l) \wedge \text{Bst}(r) . \end{aligned}$$

Na klauzálnu definíciu budeme potrebovať dva pomocné predikáty $\text{Less}(t, z)$, ktorá platí ak každá hodnota v binárnom strome t je menšia ako z , a $\text{Grater}(t, z)$, ktorý platí ak každá hodnota v binárnom strome t je väčšia ako z .

Matematicky:

$$\begin{aligned} \text{Bt}(t) \rightarrow \text{Less}(t, z) &\leftrightarrow \forall v (\text{Memb}(v, t) \rightarrow v < z) \\ \text{Bt}(t) \rightarrow \text{Grater}(t, z) &\leftrightarrow \forall v (\text{Memb}(v, t) \rightarrow v > z) . \end{aligned}$$

Klauzálné:

$$\begin{aligned} \text{Less}(E, z) \\ \text{Less}(Nd(x, l, r), z) &\leftarrow x < z \wedge \text{Less}(l, z) \wedge \text{Less}(r, z) \end{aligned}$$

$$\begin{aligned} & \text{Greater}(E, z) \\ & \text{Greater}(Nd(x, l, r), z) \leftarrow x > z \wedge \text{Greater}(l, z) \wedge \text{Greater}(r, z) . \end{aligned}$$

Celá definícia vyzerá takto:

$$\begin{aligned} & \text{Bst}(E) \\ & \text{Bst}(Nd(x, l, r)) \leftarrow N(x) \wedge \text{Less}(l, x) \wedge \text{Greater}(r, x) \wedge \text{Bst}(l) \wedge \text{Bst}(r) . \end{aligned}$$

11.3.1 Member. Zdefinujme si predikát $\text{Member}(x, t)$, pre ktorý platí:

$$\text{Bst}(t) \rightarrow \text{Member}(x, t) \leftrightarrow \text{Memb}(x, t) .$$

Klauzálné:

$$\begin{aligned} & \text{Member}(x, Nd(y, l, r)) \leftarrow x = y \\ & \text{Member}(x, Nd(y, l, r)) \leftarrow x > y \wedge \text{Member}(x, r) \\ & \text{Member}(x, Nd(y, l, r)) \leftarrow x < y \wedge \text{Member}(x, l) . \end{aligned}$$

11.3.2 Insert. Ďalej si zdefinujeme funkciu $\text{Insert}(x, t)$, ktorá vloží do binárneho vyhľadávacieho stromu vrchol s hodnotou x , tak aby výsledný strom ostal binárnym stromom.

Matematicky:

$$\begin{aligned} & \text{Bst}(t) \rightarrow \text{Bst}(\text{Insert}(x, t)) \\ & \text{Bst}(t) \rightarrow \text{Member}(z, \text{Insert}(x, t)) \leftrightarrow z = x \vee \text{Member}(z, t) . \end{aligned}$$

Klauzálné:

$$\begin{aligned} & \text{Insert}(x, E) = Nd(x, E, E) \\ & \text{Insert}(x, Nd(y, l, z)) = Nd(y, l, r) \leftarrow x = y \\ & \text{Insert}(x, Nd(y, l, z)) = Nd(y, \text{Insert}(x, l), r) \leftarrow x < y \\ & \text{Insert}(x, Nd(y, l, z)) = Nd(y, l, \text{Insert}(x, r)) \leftarrow x > y . \end{aligned}$$

11.3.3 Delete. Nakoniec si zdefinujeme funkciu $\text{Delete}(x, t)$, ktorá vymaže vrchol s hodnotou x z t :

Matematicky:

$$\begin{aligned} & \text{Bst}(t) \rightarrow \text{Bst}(\text{Delete}(x, t)) \\ & \text{Bst}(t) \rightarrow \text{Members}(z, \text{Delete}(x, t)) \leftrightarrow z \neq x \wedge \text{Member}(z, t) . \end{aligned}$$

Pre klauzálnu definíciu potrebujeme dve pomocné funkcie; ktoré nám pre dvojicu l, r bin. vyhl. stromov nájdú najväčší prvok x z l a vytvoria binárny vyhľadávací strom l_1 bez najväčšieho prvku x a napokon skonštruujú binárny vyhľadávací strom $Nd(x, l_1, r)$.

$$\begin{aligned} & \text{Join}(l, r) = r \leftarrow l = E \\ & \text{Join}(l, r) = Nd(x, l_1, r) \leftarrow l \neq E \wedge \text{Split}(l) = x, l_1 \\ & \text{Split}(Nd(x, l, r)) = x, l \leftarrow r = E \\ & \text{Split}(Nd(x, l, r)) = x_1, Nd(x, l, r_1) \leftarrow r \neq E \wedge \text{Split}(r) = x_1, r_1 . \end{aligned}$$

Celá definícia $Delete(x, t)$ vyzerá nasledovne:

$$Delete(x, E) = E$$

$$Delete(x, Nd(y, l, r)) = Join(l, r) \leftarrow x = y$$

$$Delete(x, Nd(y, l, r)) = Nd(y, Delete(x, l), r) \leftarrow x < y$$

$$Delete(x, Nd(y, l, r)) = Nd(y, l, Delete(x, r)) \leftarrow x > y .$$

Matematicky:

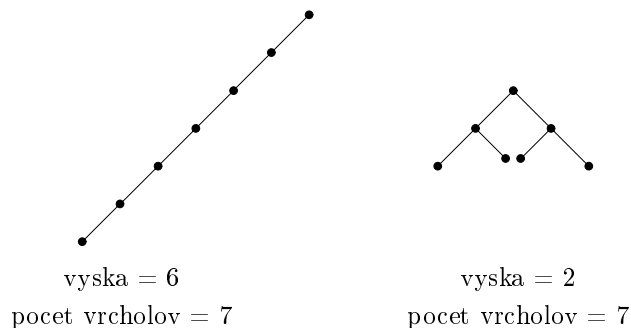
$$Bst(l) \wedge Bst(r) \rightarrow Bst(Join(l, r))$$

$$Bst(l) \wedge Bst(r) \rightarrow Member(z, Join(l, r)) \leftrightarrow Member(z, l) \vee Member(z, r) .$$

Kapitola 12

Perfektne vyvážené stromy

Binárne stromy predstavujú dôležitý nástroj na implementáciu mnohých rozšírených dátových typov, napríklad: množín, frônt, atď.. Nad týmito typmi vykonávame hlavne operácie, ktoré by sme mohli zhruba charakterizovať ako: pridanie prvku, $Insert(p, t)$, odobratie prvku, $Delete(p, t)$, a vyhľadanie prvku, $Lookup(p, t)$. Našou snahou je, aby uvedené operácie boli vykonávané čo najefektívnejšie. Keď sa pozrieme bližšie na realizáciu týchto operácií nad binárnymi vyhľadávacími stromami, tak ľahko zistíme, že zložitosť, počet krokov, ktoré musíme vykonať, keď pridávame, vyberáme alebo vyhľadávame nijaký prvok, odpovedá výške binárneho vyhľadávacieho stromu. Čiže našou snahou bude udržiavať binárny strom čo najmešou výškou pri čo najväčšom počte vrcholov. Kedy nám môžu vzniknúť patologické prípady - stromy s veľkou výškou a malým počtom vrcholov? Pri pohľade na obrázok 12.1, ktorý predstavuje takýto



Obrázok 12.1: Stromy

strom, ľahko usúdime, že ak budeme nevyvážene pridávať nové vrcholy iba do jedného podstromu (v našom prípade ľavého podstromu) a počet vrcholov v jednom podstrome bude veľmi rozdielny, tak sa dopracujeme k stromu s veľkou výškou a malým počtom vrcholov - štíhlemu stromu. Naším idealom bude malý "bucľaty" strom, kde všetky vrcholy nebudú mať ďaleko od koreňa a ich počet

bude čo najväčší. To dosiahneme tým, že nedopustíme aby jeden podstrom mal omnoho viac vrcholov ako druhý (lebo tým na rýchlo narastie výška stromu), čiže budeme nové vrcholy pridávať do ľavého a pravého podstromu vyvážené, aby počet vrcholov ľavého podstromu sa rovalnal počtu vrcholov pravého podstromu, poažme bol o jeden väčší. Túto podmienku ďalej budeme uplatňovať aj na podstromy ľavého a pravého podstromu. Touto úvahou sme sa prepracovali až k matematickej definícii binárneho perfektne vyváženého stromu:

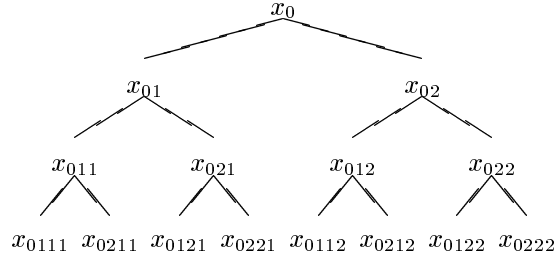
$$\begin{aligned}
 & Pbt(E) \\
 Pbt(Nd(x, l, r)) & \leftrightarrow N(x) \wedge (Size(l) = Size(r) \vee Size(l) = Size(r) + 1) \wedge \\
 & Pbt(l) \wedge Pbt(r)
 \end{aligned}$$

Klauzálné:

$$\begin{aligned}
 & Pbt(E) \\
 Pbt(Nd(x, l, r)) & \leftarrow Size(l) = Size(r) \wedge Pbt(l) \wedge Pbt(r) \\
 Pbt(Nd(x, l, r)) & \leftarrow Size(l) \neq Size(r) \wedge Size(l) = Size(r) + 1 \wedge Pbt(l) \wedge Pbt(r) .
 \end{aligned}$$

Je zrejmé, že $Pbt(t) \rightarrow Bt(t)$, binárne perfektne vyvážený strom je aj binárnym stromom. Ďalšia otázka, ktorú musíme zodpovedať, je ako budeme indexovať prvky v takto perfektne vyváženom strome. Čiže ak máme n prvkov x_0, \dots, x_{n-1} , tak akým spôsobom ich uložíme do stromu. Jednoduché a veľmi elegantné riešenie nám ponúka diadická sústava. Predstavme si indexy $0, \dots, n-1$ zapísané v diadickej sústave. Môžeme ich rozdeliť do troch skupín. Prvá skupina bude obsahovať iba 0. Druhá skupina bude tvorená tými indexami, ktorých statický zápis sa končí 1 a tretia tými, ktorých diadický zápis sa končí 2. Ľahko vidno, že v druhej skupine je rovnaký alebo o jeden väčší počet indexov ako v tretej skupine (v diadických zápisoch čísel $1, \dots, n-1$ sa nám na poslednom mieste strieda 1 a 2, pričom sa začína 1; $0_1, 0_2, 0_{11}, 0_{12}, 0_{21}, \dots$). Perfektne vyvážený strom budeme vytvárať tak, že prvok x_0 dáme do koreňa, prvky s indexami z prvej skupiny pôjdu do ľavého podstromu a prvky s indexami z druhej skupiny zase do pravého podstromu, čiže vyváženosť bude platiť v ľavom podstrome bude rovnaký alebo o jeden väčší počet vrcholov ako v pravom podstrome. Uvedenú metódu ďalej uplatníme na podstromy. Druhú triedu indexov končiacich na 1 opäť rozdelíme na tri podskupiny podľa toho či na predposlednom mieste v diadickom je 0, 1 alebo 2. Prvok s indexom z prvej podskupiny pôjde do vrchola ľavého podstromu. Prvky s indexami z druhej podskupiny zase do ľavého podstromu ľavého podstromu a z tretej podskupiny zase do pravého podstromu ľavého podstromu. Opäť v diadických zápisoch indexom z druhej skupiny okrem indexu 0_1 (z prvej podskupiny) sa na predposlednom mieste bude striedať 1 a 2, pričom začína sa 1: $0_{11}, 0_{21}, 0_{111}, 0_{121}, \dots$, čiže v druhej podskupine bude rovnaký alebo o jeden väčší počet indexov ako v tretej podskupine a taktiež ostane zachovaná vyváženosť pre ľavý a pravý podstrom ľavého podstromu. Analogicky postupujeme pre pravý podstrom, tretiu skupinu indexov končiacich sa na 2, rozdelíme ju opäť na tri podskupiny indexov, podľa toho či na predposlednom mieste ich diadických zápisov je 0, 1 alebo 2 a sformujeme kus pravého podstromu. Takto vygenerujeme celý perfektne vyvážený

strom pre prvky x_0, \dots, x_{n-1} . Metódu si ilustrujeme nasledovným príkladom. Perfektne vyvážený strom pre prvky $x_0, \dots, x_{14} = 0_{222}$ vyzerá nasledovne:



Pri generovaní stromu môžeme využiť nasledovný vzťah: ak máme "otca" $x_{0\alpha}$ tak potom jeho ľavý "syn" bude $x_{01\alpha}$ a pravý "syn" bude $x_{02\alpha}$; ktorý priamo vyplýva z navrhutej metódy: skutočne skupina indexov končiacich sa na reťazec α sa rozpadne do troch podskupín: $\{\alpha\}$, podskupiny 1α a podskupiny 2α . $0_{1\alpha}$ bude prvý prvok skupiny 1α a člen prvej podskupiny 1α a $0_{2\alpha}$ bude prvý prvok skupiny 2α a člen prvej podskupiny 2α a. Uvedenú metódu môžeme sformalizovať do klauzálnej definície, ktorá nám zo zoznamu prvkov x_0, \dots, x_{n-1} vytvorí perfektne vyvážený strom.

$$Ln2pbt(0) = E$$

$$Ln2pbt(x, y) = Nd(x, Ln2pbt(y_1), Ln2pbt(y_2)) \leftarrow Split(y) = y_1, y_2$$

$$Split(0, 0) = 0$$

$$Split(x, 0) = (x, 0), 0$$

$$Split(x_1, x_2, z) = (z_1, z_1), x_2, z_2 \leftarrow Split = z_1, z_2$$

12.0.4 Lookup. Z uvedenej metódy ľahko vyčítame ako najdme prvok x_i v strome. Ak $i = 0$ tak x_0 je priamo hodnota koreňa. Ak $i = i'1$ tak hľadáme v ľavom podstrome prvok $x_{i'}$. Ak $i = i'2$ tak hľadáme v pravom podstrome prvok $x_{i'}$. Môžeme si zapísať klauzálnu definíciu:

$$Lookup(Nd(x, l, r), 0) = x$$

$$Lookup(Nd(x, l, r), i_1) = Lookup(l, i)$$

$$Lookup(Nd(x, l, r), i_2) = Lookup(r, i) .$$

12.0.5 Pbt2ln. Vytvoríme si funkciu, ktorá nám zo stromu t s prvkami x_0, \dots, x_{n-1} vytvorí zoznam s x_0, \dots, x_{n-1} . Jej matematická charakterizácia je

$$Pbt \rightarrow L(Pbt2ln(t)) = Size(t) \wedge$$

$$\forall i (i < Size(t) \rightarrow Take(Pbt2ln(t), i) = Lookup(t, i)) .$$

Klauzálna definícia:

$$Pbt2ln(E) = 0$$

$$Pbt2ln(Nd(x, l, r)) = x, Zip(Pbt2ln(l), Pbt2ln(r))$$

$$Zip(0, 0) = 0$$

$$Zip((x, 0), 0) = x, 0$$

$$Zip((x, x_s), y, y_s) = x, y, Zip(x_s, y_s) .$$

12.0.6 Update. Ďalej si môžeme zadefinovať funkciu, ktorá nám prepíše v danom strome t prvok s indexom i hodnotou y . $Update(t, i, y)$ funkcia spĺňa nasledovnú špecifikáciu:

$$\begin{aligned} Pbt \wedge i < Size(t) &\rightarrow Pbt(Update(t, i, y)) \wedge \\ &Size(Update(t, i, y)) = Size(t) \wedge \\ &Lookup(Update(t, i, y), i) = y \wedge \\ &\forall j(j < Size(t) \wedge j \neq i \rightarrow \\ &Lookup(Update(t, i, y), j) = Lookup(t, j)) . \end{aligned}$$

Klauzálné:

$$\begin{aligned} Update(Nd(x, l, r), 0, y) &= Nd(y, l, r) \\ Update(Nd(x, l, r), i_1, y) &= Nd(x, Update(l, i, y), r) \\ Update(Nd(x, l, r), i_2, y) &= Nd(y, l, Update(r, i, y)) . \end{aligned}$$

12.0.7 Inslast. Teraz si zadefinujeme funkciu $Inslast(t, n, x)$, ktorá nám vloží x ako prvok s indexom n do stromu t o veľkosti n (teda s prvkami x_0, \dots, x_{n-1}). Funkcia má nasledovnú špecifikáciu:

$$\begin{aligned} Pbt(t) \wedge Size(t) = n &\rightarrow Pbt(Inslast(t, n, x)) \wedge \\ &Size(Inslast(t, n, x)) = n + 1 \wedge \\ &Lookup(Inslast(t, n, x), n) = x \wedge \\ &\forall i(i < n \rightarrow Lookup(Inslast(t, n, x), i) = Lookup(t, i)) . \end{aligned}$$

Klauzálné:

$$\begin{aligned} Inslast(E, 0, x) &= Nd(x, E, E) \\ Inslast(Nd(y, l, r), n_1, x) &= Nd(y, Inslast(l, n, x), r) \\ Inslast(Nd(y, l, r), n_2, x) &= Nd(y, l, Inslast(l, n, x), r) . \end{aligned}$$

12.0.8 Dellast. Zadefinujeme si inverznú funkciu $Dellast(t, n)$, ktorá vymaže z t o veľkosti $n + 1$ prvok x_n :

$$\begin{aligned} Pbt(t) \wedge Size(t) = n + 1 &\rightarrow Pbt(Dellast(t, n)) \wedge Size(Dellast(t, n)) = n \wedge \\ &\forall i(i < n \rightarrow Lookup(Dellast(t, n), i) = Lookup(t, i)) . \end{aligned}$$

Klauzálné:

$$\begin{aligned} Dellast(Nd(x, l, r), 0) &= E \\ Dellast(Nd(x, l, r), n_1) &= Nd(x, Dellast(l, n), r) \\ Dellast(Nd(x, l, r), n_2) &= Nd(x, l, Dellast(l, n)) . \end{aligned}$$

12.0.9 Infirst. Teraz si skúsime zadefinovať funkciu $Infirst(t, x)$, ktorá nám do stromu t vloží x ako prvok s indexom 0 a ostatné prvky s indexom

$0 \leq i < Size(t)$ uloží na miesta $0 < i + 1 < Size(t) + 1$. Matematická definícia:

$$Pbt(t) \rightarrow Pbt(Insfirst(t, x)) \wedge \\ Pbt2ln(Insfirst(t, x) = x, Pbt2ln(t)) .$$

Klauzálna definícia:

$$Insfirst(E, x) = Nd(x, E, E) \\ Insfirst(Nd(y, l, r), x) = Nd(x, Insfirst(r, y), l) .$$

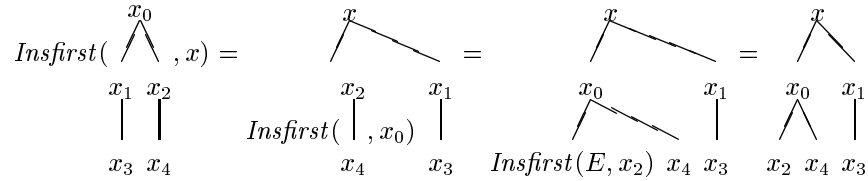
12.0.10 Delfirst. Nakoniec si zdefinujeme funkciu $Delfirst(t)$ pre neprázdne t , ktorá vymaže prvok x_0 a ostatné prvky s indexom $0 < i < Size(t)$ uloží na miesta s indexom $0 \leq i - 1 < Size(t) - 1$. Číže:

$$Pbt(t) \wedge Size(t) > 0 \rightarrow Pbt(Delfirst(t)) \wedge \\ \exists x(Pbt2ln(t) = x, Pbt2ln(Delfirst(t))) .$$

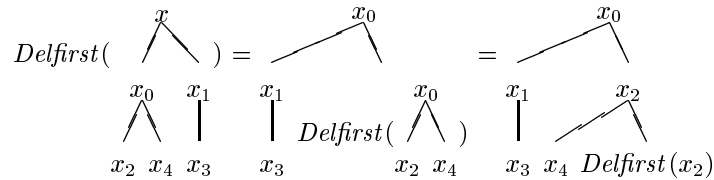
$$Delfirst(Nd(x, l, r)) = E \leftarrow l = E \\ Delfirst(Nd(x, l, r)) = Nd(y, r, Delfirst(l)) \leftarrow l = Nd(y, l_1, r_1)$$

12.1 Príklady

12.1.1 Insfirst. $x_0, x_1, x_2, x_3, x_4 \rightarrow x, x_0, x_1, x_2, x_3, x_4$



12.1.2 Delfirst. $x, x_0, x_1, x_2, x_3, x_4 \rightarrow x_0, x_1, x_2, x_3, x_4$



Kapitola 13

Aritmetické výrazy

Teraz sa budeme zaoberať ďalším dátovým typom, aritmetickými výrazmi. Ukážeme si ako sa dá tento typ zakódovať pomocou konštruktorov do prírodných čísel, zdefinujeme si funkciu *Den*, ktorá nám vypočíta hodnotu aritmetického výrazu, funkciu *Comp*, ktorá pretransformuje výraz do zoznamu inštrukcií-programu pre zásobníkový automat a nakoniec navrhne funkciu *Run*, ktorá bude simulovať zásobníkový automat pre daný program. Ak program vznikol pomocou funkcie *Comp* aplikovanej na nijaký výraz e , tak *Eval* spustená na tento program nám vráti hodnotu výrazu e . Presnejšie, budeme uvažovať aritmetické výrazy, ktoré obsahujú $+$, $*$, číselné konštanty a premenné tvaru Vn .

konštantu číslo n zakódujeme ako $Ct(n)$

premennú Vn zakódujeme ako $Vt(n)$

výraz $e_1 + e_2$ zakódujeme ako $At(\bar{e}_1, \bar{e}_2)$

výraz $e_1 * e_2$ zakódujeme ako $Mt(\bar{e}_1, \bar{e}_2)$

Aritmetický výraz môžeme formálne špecifikovať pomocou nasledujúcej typovej rovnice:

$$Exp = Ct(\mathbb{N}) | At(Exp, Exp) | Mt(Exp, Exp) | Vt(\mathbb{N})$$

Do prírodných čísel budeme tento typ kódovať nasledovne.

- budeme mať *konštruktory*:

$$Ct(x) = 0, x$$

$$At(x) = 1, x$$

$$Mt(x) = 2, x$$

$$Vt(x) = 3, x,$$

- a *predikát- formát Exp*:

$$\begin{aligned}
&Exp(Ct(n)) \\
&Exp(Vt(n)) \\
&Exp(At(e_1, e_2)) \leftarrow Exp(e_1) \wedge Exp(e_2) \\
&Exp(Mt(e_1, e_2)) \leftarrow Exp(e_1) \wedge Exp(e_2) .
\end{aligned}$$

Napríklad výraz: $2 + (3 * V4)$ zakódujeme ako

$$At(At(2), Mt(At(3), Vt(4))) = 1, (0, 2), 2, (0, 3), 3, 4 = 1184358093.$$

Hodnotu výrazu z premenných vypočítame jednoducho pomocou funkcie *Den*:

$$\begin{aligned}
Den(Ct(n)) &= n \\
Den(At(e_1, e_2)) &= Den(e_1) + Den(e_2) \\
Den(Mt(e_1, e_2)) &= Den(e_1) * Den(e_2)
\end{aligned}$$

Ak chceme zistiť, hodnotu výrazu s premennými v tvare $V0, V1, \dots, Vn, \dots$, budeme k tomu potrebovať zoznam hodnôt, ktoré chceme dosadiť za premenné. Budeme ho nazývať prostredie. Nech *env* je takéto prostredie, tak potom *Take(env, n)* bude predstavovať hodnotu pre premennú Vn . Všimnime si, že ak dĺžka $env \leq n$ a chceme zistiť hodnotu pre premennú Vn , tak $Take(env, n) = 0$. Takto sú dodefinované všetky premenné tvaru $Vn; n \in \mathbb{N}$. Na základe tohto môžeme zovšeobecniť funkciu *Den* na výraz s premennými: *Den(t, env)* nám vráti hodnotu výrazu *t* v prostredí *env*:

$$\begin{aligned}
Den(Ct(n), env) &= n \\
Den(Vt(n), env) &= Take(env, n) \\
Den(At(e_1, e_2), env) &= Den(e_1, env) + Den(e_2, env) \\
Den(Mt(e_1, e_2), env) &= Den(e_1, env) * Den(e_2, env)
\end{aligned}$$

kde

$$\begin{aligned}
Take((x_1, x_2), 0) &= x_1 \\
Take((x_1, x_2), n + 1) &= Take(x_2, n)
\end{aligned}$$

13.1 Zásobníkový automat

Teraz si povieme niečo o zásobníkovom automате. Zásobníkový automat je zariadenie, ktoré sa skladá z "pásky", ktorá obsahuje program- zoznam inštrukcií, z "pásky"- prostredia, zoznam hodnôt pre premenné zásobníka, zásobníka LIFO pre premenné a z dvoch čítacích hláv, ktoré sú nastavené nad najakými políčkami pásky. Automat si prečíta a vykoná inštrukciu v políčku pod hlavou programovej pásky, ktorá vykoná nejaké čítania a zmeny na zásobníku a čítania v prostredí, potom posunie programovú hlavu o jedno políčko doprava. Keď dočíta a vykoná všetky inštrukcie na programovej páske, presunie sa až na pravý okraj pásky, zastaví sa a vráti hodnotu, ktorá je uložená na vrchole (top) zásobníka. Na začiatku je programová hlava nastavená nad najlajším políčkom pásky, hlava pre prostredie môže byť hociakde a zásobník je prázdny.

My si teraz zdefinujeme funkciu Ren , ktorá bude simulovať prácu takéhoto zásobníkového automatu pre nejaký program, prostredie. Budeme predpokladať, že náš automat vykonáva inštrukcie nasledovného typu:

$Cx(n)$ ulož na vrchol zásobníka číslo n ,

$Vx(n)$ ulož na vrchol zásobníka hodnotu n -tého ľavého políčka (od nuly) z prostredia

Ax zober a vymaž dve vrchné políčka zo zásobníka a ulož na vrchol zásobníka ich súčet

Mx zober a vymaž dve vrchné políčka zo zásobníka a ulož na vrchol zásobníka ich súčin.

Zásobník budeme kódovať pomocou zoznamu. Vrchol zásobníka bude prvý prvok zoznamu. Keďže hlava sa posúva zľava doprava po programovej páske a v tomto poradí sa vykonávajú aj inštrukcie. Môžeme programovú pásku a hlavu simulovať tiež pomocou zoznamu, pričom jeho prvý prvok bude predstavovať to políčko nad ktorým je čítacia hlava. Keď sa hlava po vykonaní inštrukcie posunie doprava, zoznam skrátime o prvý prvok- zoberieme zvyšok zoznamu. Môžeme si to dovoliť, lebo hlava sa pohybuje vždy iba doprava a na opustené políčko sa už nikdy nevráti, preto ho zahadzujeme. Pásku- prostredie budeme tiež simulovať pomocou zoznamu.

Inštrukcie budeme kódovať pomocou konštruktorov:

$$\begin{aligned}Cx(x) &= 0, x \\Ax &= 1, 0 \\Mx &= 2, 0 \\Vx(x) &= 3, x .\end{aligned}$$

Môžeme si zdefinovať predikát *Instruction*, ktorý platí keď:

$$\begin{aligned}Inst(Cx(n)) \\Inst(Vx(n)) \\Inst(Ax) \\Inst(Mx) .\end{aligned}$$

A taktiež predikát- formát program- zoznam inštrukcií:

$$\begin{aligned}Program(0) \\Program(i, p) \leftarrow Inst(i) \wedge Program(p) .\end{aligned}$$

Celá simulácia bude vyzeráť nasledovne:

$Run(p, env, stack)$ má tri argumenty:

p je programová páska ,

env je prostredie a

$stack$ je zásobník.

$$Run(0, env, k, s) = k$$

$$Run((Cx(n), p), env, s) = Run(p, env, n, s)$$

$$Run((Vx(n), p), env, s) = Run(p, env, Take(env, n), s)$$

$$Run((Ax(n), p), env, s_1, s_2, s) = Run(p, env, s_1 + s_2, s)$$

$$Run((Mx(n), p), env, s_1, s_2, s) = Run(p, env, s_1 * s_2, s)$$

Našou ďalšou úlohou bude definovať funkciu *Comp*- kompilátor, ktorá aritmetický výraz e preloží do programu, tak aby zásobníkový automat po vykonaní programu v prostredí env vrátil hodnotu výrazu e v env $Den(e, env)$.

Matematicky:

$$Exp(e) \rightarrow Eval(Comp(e), env) = Den(e, env) ,$$

kde

$$Eval(p, env) = Run(p, env, 0) .$$

13.1.1 Kompilácia. Kompilácia bude vyzerat nasledovne. Funkciu *Comp* navrhujeme tak, aby nám platilo, že

$$Exp(e) \rightarrow Run(Comp(e) \oplus p, env, s) = Run(p, env, Den(e, env), s) \quad (1)$$

potom, pre $p = 0, s = 0$ dostaneme, že

$$\begin{aligned} Exp(e) \rightarrow Eval(Comp(e), env) &\stackrel{\text{def}}{=} Run(Comp(e) \oplus 0, env, 0) \\ &= Run(0, env, Den(e, env), s) \\ &\stackrel{\text{def}}{=} Den(e, env) \end{aligned}$$

z čoho už vyplýva naše tvrdenie-špecifikácia. Funkcia $Comp(e)$ bude nasledovná:

$$Comp(Ct(n)) = Cx(n), 0$$

$$Comp(Vt(n)) = Vx(n), 0$$

$$Comp(At(e_1, e_2)) = Comp(e_1) \oplus Comp(e_2) \oplus (Ax, 0)$$

$$Comp(Mt(e_1, e_2)) = Comp(e_1) \oplus Comp(e_2) \oplus (Mx, 0)$$

To, že vyhovuje (1) si môžeme dokázať indukciou na term e . Ak (1) bude platiť, pre ľubovoľnú konštantu $Ct(n)$ a ľubovoľnú premennú $Vt(n)$. A za predpokladu, že (1) platí pre ľubovoľné e_1, e_2 vieme ukázať, že (1) platí aj pre $At(e_1, e_2), Mt(e_1, e_2)$, tak potom bude (1) platiť pre ľubovoľný výraz e , z čoho už vyplýva naša špecifikácia.

Nech $e = Ct(n)$:

$$\begin{aligned} Run(Comp(Ct(n)) \oplus p, env, s) &= Run(Cx(n), 0) \oplus p, env, s) \\ &= Run(Cx(n), p), env, s) \\ &= Run(p, env, n, s) \\ &= Run(p, env, Den(Ct(n), env), s) . \end{aligned}$$

Nech $e = Vt(n)$:

$$\begin{aligned} Run(Comp(Vt(n)) \oplus p, env, s) &= Run(Vx(n), p), env, s) \\ &= Run(p, env, Take(env, n), s) \\ &= Run(p, env, Den(Vt(n), env), s) . \end{aligned}$$

Nech $e = At(e_1, e_2)$ a pre e_1, e_2 , už (1) platí:

$$\begin{aligned} Run(Comp(At(e_1, e_2)) \oplus p, env, s) &= Run(Comp(e_1) \oplus Comp(e_2) \oplus (Ax, 0) \oplus p, env, s) \\ &\stackrel{\text{IP pre } e_1}{=} Run(Comp(e_2) \oplus (Ax, 0) \oplus p, env, Den(e_1, env), s) \\ &\stackrel{\text{IP pre } e_2}{=} Run((Ax, p), env, Den(e_2, env), Den(e_1, env), s) \\ &= Run(p, env, Den(e_2, env) + Den(e_1, env), s) \\ &= Run(p, env, Den(At(e_1, e_2), env), s) \end{aligned}$$

Nech $e = Mt(e_1, e_2)$ a pre e_1, e_2 , už (1) platí:

$$\begin{aligned} Run(Comp(Mt(e_1, e_2)) \oplus p, env, s) &= Run(Comp(e_1) \oplus Comp(e_2) \oplus (Mx, 0) \oplus p, env, s) \\ &\stackrel{\text{IP pre } e_1}{=} Run(Comp(e_2) \oplus (Mx, 0) \oplus p, env, Den(e_1, env), s) \\ &\stackrel{\text{IP pre } e_2}{=} Run((Mx, p), env, Den(e_2, env), Den(e_1, env), s) \\ &= Run(p, env, Den(e_2, env) * Den(e_1, env), s) \\ &= Run(p, env, Den(Mt(e_1, e_2), env), s) \end{aligned}$$

Takže (1) pre ľubovoľný výraz e platí a tým aj naša špecifikácia pre $Comp(e)$.