

# Prednášky z Úvodu do deklaratívneho programovania

Ján Klúka

Letný semester 2012/2013

## Obsah

<b>I. Organizácia kurzu</b>	
<b>Deklaratívne programovanie v CL</b>	
<b>Explicitné definície</b>	<b>6</b>
<b>1. Organizácia</b>	<b>6</b>
1.1. Rozvrh a kontakty . . . . .	6
1.2. Pravidlá hodnotenia . . . . .	7
<b>2. Deklar. prog.</b>	<b>7</b>
2.1. Príklad . . . . .	8
<b>3. Jazyk CL</b>	<b>10</b>
3.1. Základy syntaxe . . . . .	11
3.2. Zabudované funkcie . . . . .	13
<b>4. Explicitné definície</b>	<b>13</b>
4.1. S jednou podmienkou . . . . .	14
4.2. S viacerými podmienkami . . . . .	16
4.3. Verifikácia . . . . .	17

<b>II. Rekurzia</b>	<b>19</b>
4.4. Opakovanie a prémiová domáca úloha . . . . .	19
<b>5. Rekurzívne definície</b>	<b>21</b>
5.1. Primitívna rekurzia . . . . .	21
5.2. Všeobecná (course-of-values) rekurzia . . . . .	23
5.3. Substitúcia v parametri . . . . .	24
5.4. Verifikácia . . . . .	25
<b>III. Priradujúce diskriminácie</b>	
<b>Chvostová rekurzia</b>	<b>27</b>
5.5. Opakovanie . . . . .	27
<b>6. Priradujúce diskriminácie</b>	<b>28</b>
6.1. Priradenie . . . . .	28
6.2. Monadická diskriminácia . . . . .	29
<b>7. Chvostová rekurzia</b>	<b>31</b>
7.1. Cyklus while . . . . .	31
7.2. Fibonacci efektívne . . . . .	35
7.3. Cyklus for . . . . .	37
<b>IV. Dyadická číselná sústava</b>	
<b>Dyadické reťazce</b>	<b>39</b>
7.4. Opakovanie . . . . .	39
<b>8. Dyadická sústava</b>	<b>41</b>
8.1. Konštruktory . . . . .	42
8.2. Diskriminácia . . . . .	43
8.3. Rekurzia . . . . .	44
8.4. Aritmetika . . . . .	45
<b>V. Symbolické operácie na dyadickom zápise.</b>	

<b>Predikáty. Párovanie.</b>	<b>49</b>
8.5. Symbolické operácie . . . . .	50
<b>9. Predikáty</b>	<b>52</b>
<b>10. Párovanie</b>	<b>54</b>
10.1. Kódovanie dátových štruktúr . . . . .	54
10.2. Párovacie funkcie . . . . .	55
10.2.1. Párovacia funkcia CL . . . . .	57
10.2.2. Programovanie s párovacou funkciou . . . . .	59
10.2.3. Tupling . . . . .	60
<b>VI. Celočíselná aritmetika. Opakovanie</b>	<b>62</b>
10.3. Kódovanie celých čísel párovaním . . . . .	62
10.3.1. Operácie na celých číslach . . . . .	64
10.3.2. Rozšírený euklidovský algoritmus . . . . .	65
<b>11. Opakovanie</b>	<b>66</b>
11.1. Chvostová rekurzia . . . . .	66
11.2. Dyadická aritmetika . . . . .	68
11.3. Symbolické operácie na dyadickom zápise . . . . .	69
<b>VII. Zoznamy.</b>	
<b>Chvostová rekurzia a tupling na zoznamoch</b>	<b>70</b>
<b>12. Zoznamy</b>	<b>70</b>
12.1. Kódovanie konečných postupností . . . . .	70
12.2. Programovanie so zoznamami . . . . .	72
12.2.1. Zrežazenie . . . . .	72
12.2.2. Porovnanie s Pascalom . . . . .	74
12.2.3. Obrátenie . . . . .	75
12.3. Chvostová rekurzia na zoznamoch . . . . .	76
12.3.1. Súčet prvkov zoznamu . . . . .	76
12.3.2. Obrátenie zoznamu . . . . .	77

12.4. Tupling zoznamových operácií . . . . .	79
12.4.1. Split = Take,Drop . . . . .	79
12.4.2. Zip, Unzip . . . . .	80
<b>IX. Triedenie zoznamov</b>	
<b>Zoznamová reprezentácia množín</b>	<b>81</b>
<b>13. Triedenie zoznamov</b>	<b>81</b>
13.1. Špecifikácia triedenia . . . . .	81
13.2. Triedenie vsúvaním . . . . .	84
13.3. Zlúčenie utriedených zoznamov . . . . .	86
13.4. Triedenie zlučováním . . . . .	87
<b>14. Zoznamová reprezentácia konečných množín</b>	<b>88</b>
<b>X. Kombinatorické predikáty a funkcie na zoznamoch</b>	<b>91</b>
<b>15. Kombinatorické predikáty a funkcie na zoznamoch</b>	<b>91</b>
15.1. Súvislé úseky . . . . .	92
15.1.1. Sufixy . . . . .	92
15.1.2. Prefixy . . . . .	94
15.1.3. Segmenty . . . . .	96
15.2. Podpostupnosti . . . . .	97
15.3. Permutácie . . . . .	99
15.4. Ďalšie úlohy . . . . .	100
<b>XI. Binárne stromy</b>	<b>101</b>
<b>16. Binárne stromy</b>	<b>101</b>
16.1. Kódovanie . . . . .	101
16.2. Operácie . . . . .	104
16.3. Prechody . . . . .	105
16.4. Vyhľadávacie stromy . . . . .	108

<b>XII.Číslovanie binárnych stromov</b>	
<b>Všeobecné stromy</b>	<b>112</b>
<b>16.Binárne stromy</b>	<b>112</b>
16.5. Číslovanie vrcholov v binárnych stromoch . . . . .	112
<b>17.Všeobecné stromy</b>	<b>115</b>
17.1. Kódovanie . . . . .	115
17.2. Programovanie so všeobecnými stromami . . . . .	118
17.3. Prechody všeobecnými stromami . . . . .	120
<b>XIIIPostfixový stroj a numerické výrazy</b>	<b>121</b>
<b>18.Postfixový stroj a numerické výrazy</b>	<b>121</b>
18.1. Postfixový stroj . . . . .	121
18.2. Numerické výrazy s premennými . . . . .	124
18.3. Kompilácia . . . . .	126
<b>19.Organizácia: Opravný test a skúška</b>	<b>131</b>
<b>XIVGenerovanie XML/XHTML</b>	<b>132</b>
<b>20.XML/XHTML</b>	<b>132</b>
20.1. Štruktúra . . . . .	132
20.2. Kódovanie . . . . .	134
20.3. Neformálne typovanie . . . . .	138
20.4. Generovanie . . . . .	138

## I. prednáška

# Organizácia kurzu

# Deklaratívne programovanie v CL

# Explicitné definície

11. februára 2013

## 1. Organizácia kurzu

### 1.1. Rozvrh a kontakty

#### 1.1 Organizácia kurzu — rozvrh a kontakty

---

**Prednášky** pondelok 9:50, poslucháreň A

**Cvičenia** pondelok 16:30 H6 1iai{1,3,5}  
pondelok 18:10 H6 1iai{2,4}

**Druháci a tretiaci: Prihláste sa** na termín podľa vlastného výberu  
(**povinne do 20. 2.**) (e-mail↓)

**Prváci:** Možná výmena človeka za človeka (e-mail↓)

Dodržiavajte termín cvičení podľa krúžku/prihlásenia, inak sa vám nemôžeme venovať.

Na test môžete prísť iba na svoj termín.

**Konzultácie** TBA

**Web** <http://dai.fmph.uniba.sk/courses/udp/>

**E-mail** [udp\(zavináč\)lists.dai.fmph.uniba.sk](mailto:udp(zavináč)lists.dai.fmph.uniba.sk)

## 1.2. Pravidlá hodnotenia

### 1.2 Organizácia kurzu – hodnotenie

---

**Semester** 2 testy po 30 bodov okolo 6. a 11. týždňa

neúčast' + *ospravedlnenie* (3 dni, Š. p. čl. 21)  $\Rightarrow$  náhradný termín (nasledujúci týždeň)

**Prémiové DÚ** Nepravidelne zadávané v cvičeniach

**Opravný test** Opravenie 1 semestrálneho testu podľa výberu (rovnaká časť učiva)

1. týždeň skúškového obdobia

Platí skóre z opravného testu, aj keby bolo nižšie!

**Podmienka pripustenia ku skúške:** zisk  $\geq 30$  bodov za semester

**Skúška** test za 40 bodov

**Známky** A  $\geq 90$  bodov

B  $\geq 80$  bodov

C  $\geq 70$  bodov

D  $\geq 60$  bodov

E  $\geq 50$  bodov

FX  $< 50$  bodov

## 2. Deklaratívne programovanie

### 1.3 Deklaratívne programovanie

---

- Spôsob (*paradigma*) programovania
- Dôraz na to, čo sa počíta, nie (len) na to, ako sa počíta
- Založené na matematickej logike
- Program je zapísaný v nejakom jazyku logiky

- Význam (*sémantika*) programu: matematický objekt Program *definuje* funkciu/reláciu
- Deklaratívne programovacie jazyky
  - funkcionálne (Lisp, ML, Haskell, ...)
  - logické (Prolog, Trilog, ASP, ...)
- *Výhody*
  - zameranie na riešený problém
  - veľká vyjadrovacia sila (expresivita)  $\implies$  stručnosť
  - ľahšia tvorba *správnych* programov

#### I.4 Tvorba správnych programov

---

Tvorba správneho programu má 3 časti:

**Špecifikácia** požadované vlastnosti programu, aké výstupy má počítať pre aké vstupy

**Implementácia** samotný program, podrobný návod na počítanie výstupov

**Verifikácia** overenie, že program má požadované vlastnosti (splňa špecifikáciu)

Deklaratívne programovanie:

- Všetky časti tvorby správneho programu sú realizované v *jednom jazyku* logiky

## 2.1. Príklad

#### I.5 Príklad — zadanie a špecifikácia

---

##### **Neformálne zadanie**

- Napíšte program, ktorý vypočíta *najväčší spoločný deliteľ* dvoch prirodzených čísel



## Špecifikácia

- Presný popis zadania v logickom jazyku
- Program v deklaratívnom jazyku bude definovať funkciu gcd s vlastnosťou:

$$x \neq 0 \vee y \neq 0 \rightarrow \text{gcd}(x, y) \mid x \wedge \text{gcd}(x, y) \mid y \wedge \\ \forall d(d \mid x \wedge d \mid y \rightarrow d \leq \text{gcd}(x, y))$$

### I.6 Príklad — implementácia

---

#### Implementácia

- Program realizuje euklidovský algoritmus
- Využíva vlastnosti deliteľnosti:

$$x \mid 0 \\ x \neq 0 \wedge d \mid x \rightarrow d \mid y \leftrightarrow d \mid y \bmod x$$

- Program v jazyku *klauzúl* (podmienенých rovností) s rekurziou:

$$\text{gcd}(x, y) = y \quad \leftarrow x = 0 \\ \text{gcd}(x, y) = \text{gcd}(y \bmod x, x) \leftarrow x \neq 0$$

- **Výpočet:** Zjednodušovanie výrazov podľa podmienенých rovností do základného tvaru (číslo)
- Napríklad  $\text{gcd}(21, 12) = \dots$
- Výpočet vždy skončí, lebo  $\dots$

**Špecifikácia** — požadovaná vlastnosť

$$x \neq 0 \vee y \neq 0 \rightarrow \text{gcd}(x, y) \mid x \wedge \text{gcd}(x, y) \mid y \wedge \\ \forall d(d \mid x \wedge d \mid y \rightarrow d \leq \text{gcd}(x, y))$$

**Implementácia** — program v jazyku klauzúl

$$\begin{aligned} \text{gcd}(x, y) &= y && \leftarrow x = 0 \\ \text{gcd}(x, y) &= \text{gcd}(y \bmod x, x) && \leftarrow x \neq 0 \end{aligned}$$

Má program požadovanú vlastnosť?

**Testovanie:** *Výpočtom* overíme pre *niektoré* vstupy

**Verifikácia:** *Dôkazom* overíme pre *všetky* vstupy

- Pre gcd: úplná matematická indukcia + vlastnosti deliteľnosti a celočíselného delenia

## 3. Jazyk CL

- Jednoduchý deklaratívny programovací jazyk
- Základ: logika prvého rádu s aritmetikou
- Syntax programov: podmienené rovnosti
- Význam programov: funkcie na prirodzených číslach
- Prostredie pre programovanie, testovanie, špecifikáciu a verifikáciu programov
- Autori: Pavol Voda, Ján Komara, Ján Klúka

### 3.1. Základy syntaxe

#### I.9 Základy syntaxe — premenné, funkčné symboly

---

**Premenná** neprázdna postupnosť malých písmen anglickej abecedy nasledovaná najviac dvoma číslicami

Napríklad:  $x$ ,  $y$ ,  $zvysok3$ ,  $zoznam99$ ,

ale nie:  ~~$X$~~ ,  ~~$velkePismeno$~~ ,  ~~$_pociar_kovnik$~~ ,  ~~$p123$~~ ,  ~~$kfdel$~~

**Funkčný symbol** postupnosť znakov začínajúca veľkým písmenom a pokračujúca ľubovoľnou postupnosťou malých písmen, číslic a podčiarkovníkov

Napríklad  $F$ ,  $Perms$ ,  $F\_n1\_m1$ ,  $Q54321$ ,

ale nie  ~~$funkcia$~~ ,  ~~$ZoznamZnakov$~~ ,  ~~$_pociarkovnik$~~ ,  ~~$Zvysok$~~

Každý f. s. má priradený počet argumentov — *aritu*

Funkčný symbol s aritou  $n$  nazývame  $n$ -árny, špeciálne:

$n = 0$  — nulárny f. s. alebo *konštanta*       $n = 2$  — binárny f. s.

$n = 1$  — unárny f. s.                                       $n = 3$  — ternárny f. s.

Dohoda:  $F$ ,  $G$  označujú funkčné symboly

#### I.10 Základy syntaxe — konštanty (literály)

---

**Číselná konštanta** (číselný literál) postupnosť číslic

Napríklad  $0$ ,  $8$ ,  $2013$ ,  $0123456789876543210$

**Znaková konštanta** (znakový lit.) ASCII znak v úvodzovkách (")

Napríklad  $"x"$ ,  $"7"$ ,  $"?"$ ,  $"\\"$ ,  $"'"$ ,  $"''"$ ,  ~~$"ô"$~~ ,  ~~$"znak"$~~

**Reťazcová konštanta** (reťazcový lit.) postupnosť znakov okrem apostrofu (') uzavretá v apostrofoch

Napríklad  $'x'$ ,  $'70 sukien mala'$ ,  $"Aha!"$ ,  $zvolal.'$ ,  ~~$'Tak 'toto' teda nie!'$~~

### I.11 Termy

---

Množina **termov** je najmenšia množina postupností symbolov spĺňajúca podmienky:

- premenná je term
- konštanta (ľubovoľného druhu) je term
- ak  $\tau$  a  $\sigma$  sú termy, tak aj *aplikácie zabudovaných binárnych funkcií*:

$$(\tau + \sigma) \quad (\tau \div \sigma) \quad (\tau \cdot \sigma) \quad (\tau \div \sigma) \quad (\tau \bmod \sigma) \quad (\tau, \sigma) \quad (\tau \oplus \sigma)$$

sú termy

- ak  $F$  je  $n$ -árny funkčný symbol,  $n \geq 1$  a  $\tau_1, \tau_2, \dots, \tau_n$  sú termy, tak aj *aplikácia funkčného symbolu  $F$*

$$F(\tau_1, \tau_2, \dots, \tau_n)$$

je term

Dohoda:  $\tau, \sigma$  označujú termy

### I.12 Termy – príklady a zjednodušenia

---

Príklady termov:

$$(a + b) \quad (((a \cdot a) + ((2 \cdot a) \cdot b)) + (b \cdot b)) \quad \text{Sq}(x)$$

$$\text{Sq}((x + (y \bmod \text{Zaklad}))) \quad \text{Min}(\text{Sq}(\text{Max}(a, b)), \text{Min}((x + 1), (y + 1)))$$

Sq je unárny f. s., Min a Max sú binárne f. s. a Zaklad je konštanta

Nemusíme písať všetky zátvorky:

$$a + b \quad a \cdot a + 2 \cdot a \cdot b + b \cdot b \quad \text{Sq}(x)$$

$$\text{Sq}(x + y \bmod \text{Zaklad}) \quad \text{Min}(\text{Sq} \text{Max}(a, b), \text{Min}(x + 1, y + 1))$$

Zabudované binárne funkcie:

	priorita	implicitné zátvorkovanie pri rovnakej priorite
$\cdot \quad \div \quad \bmod$	najvyššia	vľavo
$+$		vľavo
$\oplus$		vľavo
$,$	najnižšia	vpravo

## 3.2. Zabudované funkcie

### I.13 Niektoré funkcie zabudované v CL

---

Binárne aritmetické funkcie *na prirodzených číslach*:

+ sčítanie (v zdrojovom kóde +)

· násobenie (v zdrojovom kóde \*)

÷ *modifikované* odčítanie (v zdrojovom kóde -)

$$x \dot{-} y = \begin{cases} x - y & \text{ak } x \geq y, \\ 0 & \text{inak} \end{cases}$$

÷ delenie (/) a

mod zvyšok po delení (mod), ktoré majú nasledujúce vlastnosti:

$$x \div 0 = 0$$

$$x \bmod 0 = 0$$

$$y \neq 0 \rightarrow x = (x \div y) \cdot y + (x \bmod y) \wedge x \bmod y < y$$

Zabudovaných je aj niekoľko ďalších funkcií — neskôr

## 4. Explicitné definície

### I.14 Moduly a definície funkcií

---

CL *modul* je postupnosť definícií funkcií

*Definícia funkcie*

- množina podmienených rovností — *klauzúl* (clauses)

$$F(x_1, \dots, x_n) = \tau_1 \leftarrow \varphi_1$$

...

$$F(x_1, \dots, x_n) = \tau_k \leftarrow \varphi_k$$

skonštruovaná určitým spôsobom (postupne sa ho naučíme)

- *jednoznačne určuje význam* (sémantiku) funkčného symbolu na základe zabudovaných a predtým definovaných funkcií
- určuje *spôsob výpočtu* hodnoty funkcie pre ľubovoľné hodnoty jej argumentov

### I.15 Explicitné definície

---

#### Explicitná definícia funkcie

- klauzuly obsahujú iba argumenty funkcie (premenné), konštanty a *predtým* definované funkcie

Jednoduché explicitné definície majú iba jednu rovnosť, napríklad:

$$Z(x) = 0$$

$$\text{Twice}(x) = 2 \cdot x$$

$$\text{Square}(x) = x \cdot x$$

$$\text{Cube}(x) = x \cdot \text{Square}(x)$$

Vo všeobecnosti:

$$F(x_1, \dots, x_n) = \tau$$

- $F$  je nový  $n$ -árny funkčný symbol
- $\tau$  je term obsahujúci iba premenné  $x_1, \dots, x_n$ , konštanty a symboly predtým zadaných funkcií

## 4.1. Explicitné definície s jednou podmienkou

### I.16 Explicitné definície s podmienkami

---

- Zložitejšie definície majú viacero klauzúl

$$\text{Sgn}(x) = 0 \leftarrow x = 0$$

$$\text{Sgn}(x) = 1 \leftarrow x \neq 0$$

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

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

- Argumenty funkcie v rôznych klauzulách musia byť rovnaké

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

~~$$\text{Min}(a, b) = b \leftarrow a > b$$~~

- Podmienky v klauzulách musia byť *výlučné*
- Podmienky musia pokryť *všetky možné prípady*
- ▶ Potom sa vždy dá použiť práve jedna klauzula
- CL **nedovoľuje v klauzulách hocijaké podmienky**, nemôže vždy rozpoznať výlučnosť a pokrytie všetkých prípadov

### I.17 Diskriminácie

Dovolenú sadu podmienok voláme *diskriminácia*

Niektoré diskriminácie ( $\tau$  a  $\sigma$  sú ľubovoľné termy):

- diskriminácia na rovnosť

$$F(\dots) = \dots \leftarrow \tau = \sigma$$

$$F(\dots) = \dots \leftarrow \tau \neq \sigma$$

$$\neg(\tau = \sigma \wedge \tau \neq \sigma)$$

$$\tau = \sigma \vee \tau \neq \sigma$$

- dichotómia

$$F(\dots) = \dots \leftarrow \tau \leq \sigma$$

$$F(\dots) = \dots \leftarrow \tau > \sigma$$

$$\neg(\tau \leq \sigma \wedge \tau > \sigma)$$

$$\tau \leq \sigma \vee \tau > \sigma$$

Teda

- **je** dovolené napríklad:

$$F(x) = x \leftarrow x+7 = x \cdot 7$$

$$F(x) = x+1 \leftarrow x+7 \neq x \cdot 7$$

$$G(x, y) = x \cdot y \leftarrow y > x$$

$$G(x, y) = x+y \leftarrow y \leq x$$

- **nie je** dovolené napríklad:

$$F(x, y) = 0 \leftarrow x = y$$

$$F(x, y) = 1 \leftarrow y \neq x$$

$$G(x, y, z) = y+z \leftarrow y \leq x$$

$$G(x, y, z) = x+z \leftarrow y \geq x$$

## 4.2. Explicitné definície s viacerými podmienkami

### I.18 Kombinované podmienky

---

- Podmienky v klauzulách možno kombinovať *iba* logickou spojkou  $\wedge$
- Dovoľené sú len niektoré kombinácie podmienok
- CL musí rozpoznať, že podmienky sú vzájomne vylučné a pokrývajú všetky prípady
- Nie je dovoľené napríklad:

$$\text{Min}_3(x, y, z) = x \leftarrow x \leq y \wedge x \leq z$$

$$\text{Min}_3(x, y, z) = y \leftarrow y \leq z \wedge y \leq x$$

$$\text{Min}_3(x, y, z) = z \leftarrow z \leq x \wedge z \leq y$$

### I.19 Ako kombinovať podmienky

---

- CL rozpozná vzájomnú vylučnosť podmienok, ak vzniknú vnorením diskriminácií
- Napríklad podmienky v definícii:

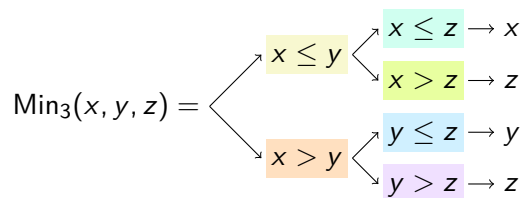
$$\text{Min}_3(x, y, z) = x \leftarrow \begin{matrix} x \leq y \\ x \leq z \end{matrix}$$

$$\text{Min}_3(x, y, z) = z \leftarrow \begin{matrix} x \leq y \\ x > z \end{matrix}$$

$$\text{Min}_3(x, y, z) = y \leftarrow \begin{matrix} x > y \\ y \leq z \end{matrix}$$

$$\text{Min}_3(x, y, z) = z \leftarrow \begin{matrix} x > y \\ y > z \end{matrix}$$

vzniknú vnorením diskriminácií:





## I.20 Ako kombinovať podmienky — recept

Zaručene správna konštrukcia podmienok:

kým nevieme určiť výsledok, vnárame dovolené diskriminácie

$$\text{Min}_3(x, y, z) = ? \leftarrow ?$$

$$\text{Min}_3(x, y, z) = ? \leftarrow x \leq y$$

$$\text{Min}_3(x, y, z) = ? \leftarrow x > y$$

$$\text{Min}_3(x, y, z) = x \leftarrow x \leq y \wedge x \leq z$$

$$\text{Min}_3(x, y, z) = z \leftarrow x \leq y \wedge x > z$$

$$\text{Min}_3(x, y, z) = ? \leftarrow x > y$$

$$\text{Min}_3(x, y, z) = x \leftarrow x \leq y \wedge x \leq z$$

$$\text{Min}_3(x, y, z) = z \leftarrow x \leq y \wedge x > z$$

$$\text{Min}_3(x, y, z) = y \leftarrow x > y \wedge y \geq z$$

$$\text{Min}_3(x, y, z) = z \leftarrow x > y \wedge y > z$$

## 4.3. Verifikácia explicitne definovaných funkcií

### I.21 Špecifikácia minima troch

Hodnotou funkcie  $\text{Min}_3$  je najmenší z jej argumentov, teda hodnota  $\text{Min}_3$  sa

- rovná niektorému z argumentov:

$$\text{Min}_3(x, y, z) = x \vee \text{Min}_3(x, y, z) = y \vee \text{Min}_3(x, y, z) = z$$

- nanajvýš rovná každému argumentu:

$$\text{Min}_3(x, y, z) \leq x \wedge \text{Min}_3(x, y, z) \leq y \wedge \text{Min}_3(x, y, z) \leq z$$

*Dokážme — verifikujme, že tieto vlastnosti platia pre našu definíciu*

$$\text{Min}_3(x, y, z) = x \leftarrow x \leq y \wedge x \leq z \tag{1}$$

$$\text{Min}_3(x, y, z) = z \leftarrow x \leq y \wedge x > z \tag{2}$$

$$\text{Min}_3(x, y, z) = y \leftarrow x > y \wedge y \leq z \tag{3}$$

$$\text{Min}_3(x, y, z) = z \leftarrow x > y \wedge y > z \tag{4}$$

Ako na to?

$$\text{Min}_3(x, y, z) \leq x \wedge \text{Min}_3(x, y, z) \leq y \wedge \text{Min}_3(x, y, z) \leq z \quad (*)$$

Priamy dôkaz: Nech  $x, y, z$  sú ľubovoľné čísla.

Na určenie hodnoty  $\text{Min}_3(x, y, z)$  podľa našej definície musíme porovnať  $x$  a  $y$  – tak, ako v samotnej definícii – a *analyzovať prípady*, ktoré môžu nastať:

- Ak  $x \leq y$ , hodnotu  $\text{Min}_3$  určí niektorá z klauzúl (1) a (2) na základe porovnania  $x$  a  $z$  – vnorená analýza prípadov:
  - Ak  $x \leq z$ , podľa (1) je  $\text{Min}_3(x, y, z) = x$ . Ľahko overíme, že platia všetky konjunkty dokazovanej vlastnosti (\*): zrejme  $x \leq x$  a podľa aktuálnych predpokladov  $x \leq y$  a  $x \leq z$ .
  - Ak  $x > z$ , podľa (2) je  $\text{Min}_3(x, y, z) = z$ . Opäť overíme platnosť všetkých konjunktov (\*): Podľa aktuálnych predpokladov  $z < x \leq y$  a zrejme  $z \leq z$ .
- Ak  $x > y$ , hodnotu  $\text{Min}_3$  určíme porovnaním  $y$  a  $z$ . Postup je podobný ako v predchádzajúcom prípade.

## II. prednáška

# Rekurzia

18. februára 2013

### 4.4. Opakovanie a prémiová domáca úloha

#### II.1 Opakovanie 1. prednášky

---

- Explicitné definície — skladanie už definovaných funkcií

$$\text{Square}(x) = x \cdot x$$

$$\text{Cube}(x) = x \cdot \text{Square}(x)$$

- Explicitné klauzálne definície — podmienené výpočty

$$\min(x, y) = x \leftarrow x \leq y$$

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

Diskriminácie — dovolené kombinácie podmienok

$$\dots \leftarrow s \leq \tau$$

$$\dots \leftarrow s = \tau$$

$$\dots \leftarrow s > \tau$$

$$\dots \leftarrow s \neq \tau$$

vzájomne výlučné, pokrývajúce všetky prípady

$$\neg(s \leq \tau \wedge s > \tau)$$

$$\neg(s = \tau \wedge s \neq \tau)$$

$$(s \leq \tau \vee s > \tau)$$

$$(s = \tau \vee s \neq \tau)$$

- ▶ Chýba nám *opakovanie*

## II.2 Prémiová domáca úloha

---

$$\text{Median}(x, y, z) = y \leftarrow x \leq y \wedge y \leq z$$

$$\text{Median}(x, y, z) = z \leftarrow x \leq y \wedge y > z \wedge x \leq z$$

$$\text{Median}(x, y, z) = x \leftarrow x \leq y \wedge y > z \wedge x > z$$

$$\text{Median}(x, y, z) = y \leftarrow x > y \wedge y > z$$

$$\text{Median}(x, y, z) = x \leftarrow x > y \wedge y \leq z \wedge x \leq z$$

$$\text{Median}(x, y, z) = z \leftarrow x > y \wedge y \leq z \wedge x > z$$

Dokážte, že

$$\text{Median}(x, y, z) = \text{Median}(y, x, z) \quad (\text{Median\_perm1})$$

Pozor!

- Vychádzame z našej definície funkcie, musíme ju uviesť.
- Nemôžeme predpokladať, že vlastnosť platí, keď ju máme dokázať. (Dôkaz kruhom.)
- Dokazujeme rovnostnú vlastnosť Median\_perm1, teda musíme ukázať, že za každých okolností sú hodnoty na ľavej a pravej strane identické. Nestačí zistiť, čomu sa rovná ľavá strana.

Ako má teda dôkaz vyzerieť?

## II.3 Prémiová domáca úloha — dôkaz

---

$$\text{Median}(a, b, c) = b \leftarrow a \leq b \wedge b \leq c \quad (1)$$

$$\text{Median}(a, b, c) = c \leftarrow a \leq b \wedge b > c \wedge a \leq c \quad (2)$$

$$\text{Median}(a, b, c) = a \leftarrow a \leq b \wedge b > c \wedge a > c \quad (3)$$

$$\text{Median}(a, b, c) = b \leftarrow a > b \wedge b > c \quad (4)$$

$$\text{Median}(a, b, c) = a \leftarrow a > b \wedge b \leq c \wedge a \leq c \quad (5)$$

$$\text{Median}(a, b, c) = c \leftarrow a > b \wedge b \leq c \wedge a > c \quad (6)$$

---

$$\text{Median}(x, y, z) = \text{Median}(y, x, z) \quad (\text{Median\_perm1})$$

---

- Ak  $x \leq y$  a  $y \leq z$ , tak  $\text{Median}(x, y, z) = y$  podľa (1).  
Čomu sa rovná  $\text{Median}(y, x, z)$  — pravá strana dokazovanej rovnosti (Median\_perm1)?  
Dosadíme  $a := y$ ,  $b := x$ ,  $c := z$  do (1)–(6).  
Platí  $b \leq a \leq c$ . Hodnotu  $\text{Median}(a, b, c) = \text{Median}(y, x, z)$  určí až ďalšia analýza prípadov:
  - Ak  $b = a$ , tak  $a = b \leq c$  a  $\text{Median}(a, b, c) \stackrel{(1)}{=} b = a$ .
  - Ak  $b \neq a$ , teda  $b < a$  musíme rozhodnúť medzi klauzulami (4)–(6):

\* Prípád  $b > c$  nemôže nastať (spor s predpokladom).

\* Ak  $b \leq c$ ,  $\text{Median}(a, b, c) \stackrel{(5)}{=} a$ .

V oboch prípadoch  $\text{Median}(y, x, z) = \text{Median}(a, b, c) = a = y =$   
 $= \text{Median}(x, y, z)$ , a teda ( $\text{Median\_perm1}$ ) platí.

- V ďalších prípadoch postupujeme podobne...

## 5. Rekurzívne definície

### II.4 Rekurgia — princíp

---

- Opakovanie v deklaratívnom programovaní — rekurgia
- *Rekuzívna* definícia vyjadruje hodnotu funkcie pre nejaké argumenty pomocou hodnoty *tej istej funkcie* pre *jednoduchšie argumenty*

$$0! = 1$$

$$(n + 1)! = (n + 1) \cdot n!$$

- Rôzne druhy rekuzie — rôzne zjednodušenia rekuzívnych argumentov

### 5.1. Primitívna rekurgia

#### II.5 Primitívna rekurgia

---

- *Primitívna rekurgia*: rekuzívny argument je o 1 menší
- Súčet čísel od 0 po  $n$ :

$$\sum_{i=0}^n i = \begin{cases} 0 & \text{ak } n = 0, \\ n + \sum_{i=0}^{n-1} i & \text{inak} \end{cases}$$

- CL definícia funkcie  $\text{Sum}(n) = \sum_{i=0}^n i$  pomocou diskriminácie na rovnosť:

$$\begin{aligned} \sum_{i=0}^n i &= 0 && \leftarrow n = 0 \\ \sum_{i=0}^n i &= n + \sum_{i=0}^{n-1} i && \leftarrow n \neq 0 \end{aligned}$$

- Výpočet:

$$\begin{aligned} \sum_{i=0}^5 i &= 5 + \sum_{i=0}^4 i = 5 + (4 + \sum_{i=0}^3 i) = \dots \\ &= 5 + (4 + (3 + (2 + (1 + \sum_{i=0}^0 i)))) \\ &= 5 + (4 + (3 + (2 + (1 + 0)))) \\ &= 5 + (4 + (3 + (2 + 1))) = 5 + (4 + (3 + 3)) \\ &= 5 + (4 + 6) = 5 + 10 = 15 \end{aligned}$$

- Čo je potrebné na počítačovú realizáciu výpočtu?

*Zásobník*

## II.6 Aritmetika primitívnou rekurziou

---

- Predpokladajme, že máme zabudované iba sčítanie a odčítanie
- Definícia násobenia  $\text{Mul}(x, y) = x \cdot y$  primitívnou rekurziou:

$$\begin{aligned} x \cdot y &= 0 && \leftarrow x = 0 \\ x \cdot y &= \dots (x \div 1) \cdot y \dots && \leftarrow x \neq 0 \end{aligned}$$

- Všetky programy na tomto predmete budú *primitívne rekurzívne*
  - ▶ Definovateľné primitívnou rekurziou a skladaním funkcií z nuly  $Z(x) = 0$ , nasledovníka  $S(x) = x+1$  a projekcií  $I_k^n(x_1, \dots, x_k, \dots, x_n) = x_k$
- Programovať iba primitívnou rekurziou by bolo neefektívne a nepohodlné
  - ▶ Príklad: hľadanie celočíselného podielu  $x$  a  $y$  skúšaním všetkých čísel od  $x$  po 0.

## 5.2. Všeobecná (course-of-values) rekurzia

### II.7 Všeobecná (course-of-values) rekurzia

---

- Na pripomenutie:
  - Rekurzia vyjadruje hodnotu funkcie pre nejaký argument pomocou hodnoty tej istej funkcie pre *jednoduchšie argumenty*
  - Primitívna rekurzia: rekurzívny argument je o 1 menší
- *Všeobecná (course-of-values) rekurzia*: rekurzívny argument je *ľubovoľné menšie* číslo
- Príklad: Vieme iba sčítavať, odčítavať, násobiť, chceme delenie  $\text{Div}(x, y) = x \div y$ :

$$y \neq 0 \rightarrow \exists r(x = (x \div y) \cdot y + r \wedge r < y)$$

- Definícia pomocou všeobecnej rekurzie:

$$\begin{aligned}x \div y &= 0 && \leftarrow y = 0 \\x \div y &= 0 && \leftarrow y \neq 0 \wedge x < y \\x \div y &= ((x \div y) \div y) + 1 && \leftarrow y \neq 0 \wedge x \geq y\end{aligned}$$

Je zaručené zmenšovanie rekurzívneho argumentu?

$$y \neq 0 \wedge x \geq y \rightarrow x \div y < x$$

### II.8 Všeobecná rekurzia s viacerými rekurzívnymi volaniami

---

- Fibonacciho postupnosť 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

$$\begin{aligned}\text{fib}_0 &= 0 \\ \text{fib}_1 &= 1 \\ \text{fib}_{n+2} &= \text{fib}_{n+1} + \text{fib}_n\end{aligned}$$

- Funkcia  $\text{Fib}(n) = \text{fib}_n$  počíta  $n$ -tý prvok Fibonacciho postupnosti

- Definícia všeobecnej rekúriou
  - s *dvoma* rekurzívnymi volaniami, a
  - so zloženými podmienkami:

$$\text{fib}_n = \dots \leftarrow \dots$$

## 5.3. Všeobecná rekúzia so substitúciou v parametri

### II.9 Substitúcia v parametri

---

- Najväčší spoločný deliteľ:

$$x \neq 0 \vee y \neq 0 \rightarrow \text{gcd}(x, y) \mid x \wedge \text{gcd}(x, y) \mid y \wedge \\ \forall d(d \mid x \wedge d \mid y \rightarrow d \leq \text{gcd}(x, y))$$

- Využijeme

$$x \mid 0 \\ x \neq 0 \wedge d \mid x \rightarrow d \mid y \leftrightarrow d \mid y \bmod x \\ x \neq 0 \rightarrow y \bmod x < x$$

- Definícia všeobecnej rekúriou:

$$\text{gcd}(x, y) = y \quad \leftarrow x = 0 \\ \text{gcd}(x, y) = \text{gcd}(y \bmod x, x) \quad \leftarrow x \neq 0$$

- Ktorý argument gcd je rekurzívny (zaručene klesajúci)?
  - ▶ prvý ( $x$ ), lebo  $x \neq 0 \rightarrow y \bmod x < x$
- Mení sa aj nerekurzívny argument (*parameter*): course-of-values rekúzia *so substitúciou v parametri*



## 5.4. Verifikácia rekurzívne definovaných funkcií indukciou

### II.10 Verifikácia primitívne rekurzívnych funkcií

---

Funkcia

$$\sum_{i=0}^n i = 0 \quad \leftarrow n = 0$$

$$\sum_{i=0}^n i = n + \sum_{i=0}^{n-1} i \quad \leftarrow n \neq 0$$

má dobre známu vlastnosť:

$$\sum_{i=0}^n i = \frac{n \cdot (n+1)}{2} \quad (*)$$

Ako ju dokážeme? Skúsme analýzu prípadov podľa definície:

- Ak  $n = 0$ , podľa prvej klauzuly  $\sum_{i=0}^n i = 0$ . Zároveň

$$\frac{0 \cdot (0+1)}{2} = \frac{0}{2} = 0$$

Zjavne teda (\*) platí.

- Ak  $n > 0$ , podľa druhej klauzuly  $\sum_{i=0}^n i = n + \sum_{i=0}^{n-1} i$  a  $\frac{n \cdot (n+1)}{2} = \frac{((n-1)+1) \cdot (n+1)}{2} = \frac{(n-1) \cdot n + 2 \cdot n}{2} = n + \frac{(n-1) \cdot n}{2}$ .

Čo potrebujeme, aby sa obe strany v (\*) rovnali?

Potrebujeme, aby

$$\sum_{i=0}^{n-1} i = \frac{(n-1) \cdot n}{2},$$

teda aby (\*) platila pre  $n-1$ .

Inak povedané, potrebujeme *indukčný predpoklad*.

Začnime teda odznova:

### II.11 Primitívna rekurzia a matematická indukcia

---

$$\sum_{i=0}^n i = 0 \quad \leftarrow n = 0$$

$$\sum_{i=0}^n i = n + \sum_{i=0}^{n-1} i \quad \leftarrow n \neq 0$$

---

$$\sum_{i=0}^n i = \frac{n \cdot (n+1)}{2} \quad (*)$$

---

Dôkaz matematickou indukciou na  $n$ :

1. Dokážme (\*) pre 0:

$$\sum_{i=0}^0 i \stackrel{\text{def}}{=} 0 = \frac{0 \cdot (0+1)}{2}.$$

2. IP: Predpokladajme, že (\*) platí pre  $n$ .

Dokážme pre  $n + 1$ :

$$\begin{aligned} \sum_{i=0}^{n+1} i &\stackrel{\text{def}}{=} n + 1 + \sum_{i=0}^{n+1-1} i \stackrel{\text{IP}}{=} \\ &\stackrel{\text{IP}}{=} n + 1 + \frac{n \cdot (n+1)}{2} = \frac{2 \cdot n + 2 + n \cdot (n+1)}{2} = \frac{(n+1) \cdot (n+2)}{2}. \end{aligned}$$

## II.12 Všeobecná rekúzia a ...?

Funkcie definované primitívnou rekúziou spravidla verifikujeme matematickou indukciou.

Funkcie definované všeobecnou rekúziou spravidla verifikujeme ...?

# III. prednáška

## Priradujúce diskriminácie

### Chvostová rekurgia

25. februára 2013

## 5.5. Opakovanie

### III.1 Opakovanie: Rekurgia

---

#### Rekurzia

vyjadrenie hodnoty funkcie pre nejaký argument pomocou hodnôt *tej istej funkcie pre jednoduchšie argumenty*

#### Primitívna rekurgia

argument rekurzívneho volania je o 1 menší

$$\begin{aligned}x^y &= 1 && \leftarrow y = 0 \\x^y &= x \cdot (x^{y-1}) && \leftarrow y \neq 0\end{aligned}$$

#### Všeobecná rekurgia (course-of-values)

argument rekurzívneho volania je ľubovoľné číslo ostro menšie ako pôvodný argument

$$\begin{aligned}\gcd(x, y) &= y && \leftarrow x = 0 \\ \gcd(x, y) &= \gcd(y \bmod x, x) && \leftarrow x \neq 0 \quad (x \neq 0 \rightarrow y \bmod x < x)\end{aligned}$$

rekurzívnych volaní môže byť viac

$$\begin{aligned}\text{fib}_n &= 0 && \leftarrow n < 2 \wedge n = 0 \\ \text{fib}_n &= 1 && \leftarrow n < 2 \wedge n \neq 0 \\ \text{fib}_n &= \text{fib}_{n-1} + \text{fib}_{n-2} && \leftarrow n \geq 2 \quad (n \geq 2 \rightarrow n-1 < n \wedge n-2 < n)\end{aligned}$$

### III.2 Opakovanie: Známe diskriminácie

---

Zatiaľ poznáme iba dva druhy *diskriminácií* (dovolených kombinácií podmienok) v CL:

$$\begin{array}{ll} \dots \leftarrow s \leq \tau & \dots \leftarrow s = \tau \\ \dots \leftarrow s > \tau & \dots \leftarrow s \neq \tau \end{array}$$

Vzájomne výlučné, pokrývajúce všetky prípady

$$\begin{array}{ll} \neg(s \leq \tau \wedge s > \tau) & \neg(s = \tau \wedge s \neq \tau) \\ (s \leq \tau \vee s > \tau) & (s = \tau \vee s \neq \tau) \end{array}$$

Obe doterajšie diskriminácie testujú známe hodnoty.

## 6. Priradujúce diskriminácie

### 6.1. Priradenie

#### III.3 Priradenie

---

CL umožňuje *priradiť* hodnotu termu *novej* premennej

$$\dots \leftarrow \dots \wedge \tau = x$$

- nová premenná na *pravej strane* rovnosti (ako v query)
- $\tau$  je ľubovoľný term so známou hodnotou (zľava doprava)

Diskriminácia s jediným prípadom (vždy pravdivým). Vždy platí:

$$\exists x(\tau = x)$$

Príklady:

$$n! = 1 \quad \leftarrow n = 0$$

$$n! = n \cdot m! \quad \leftarrow n \neq 0 \wedge n \div 1 = m$$

$$\gcd(x, y) = y \quad \leftarrow x = 0$$

$$\gcd(x, y) = \gcd(x_1, y_1) \quad \leftarrow x \neq 0 \wedge y \bmod x = x_1 \wedge x = y_1$$

## 6.2. Monadická diskriminácia

### III.4 Monadická diskriminácia – porovnanie so vzorom

Monadická diskriminácia kombinuje test a priradenie

píšeme

$$\dots \leftarrow \tau = 0$$

$$\dots \leftarrow \tau = x + 1$$

namiesto

$$\dots \leftarrow \tau = 0$$

$$\dots \leftarrow \tau \neq 0 \wedge \tau \div 1 = x$$

$x$  je nová premenná,  $\tau$  je ľubovoľný term

Podmienky sú vzájomne vylučné a pokrývajú všetky prípady:

$$\neg(\tau = 0 \wedge \exists x(\tau = x + 1)) \quad (\tau = 0 \vee \exists x(\tau = x + 1))$$

Rovnosti  $\tau = x + 1$  a  $\tau = 0$  – porovnanie so vzorom (pattern matching)

Príklad použitia:

$$n! = 1 \quad \leftarrow n = 0$$

$$n! = n \cdot m! \quad \leftarrow n = m + 1$$

$$x^y = 1 \quad \leftarrow y = 0$$

$$x^y = x \cdot x^z \quad \leftarrow y = z + 1$$

### III.5 Monadická diskriminácia – dosadenie

Vzory monadickej diskriminácie

$$n! = 1 \quad \leftarrow n = 0$$

$$n! = n \cdot m! \quad \leftarrow n = m + 1$$

môžeme dosadiť namiesto premennej na ľavej strane rovnosti:

$$0! = 1$$

$$(m + 1)! = (m + 1) \cdot m!$$

$$x^0 = 1$$

$$x^{z+1} = x \cdot x^z$$



Monadickú diskrimináciu dosádzajte, iba ak je prvá

Dosaďte buď oba prípady alebo žiadny

### III.6 Zovšeobecnená monadická diskriminácia

Zovšeobecnená monadická diskriminácia — skratka kaskády:

$$\begin{array}{ll}
 \dots \leftarrow \tau = 0 & \dots \leftarrow \tau = 0 \\
 \dots \leftarrow \tau = 1 & \dots \leftarrow \tau = x_1 + 1 \wedge x_1 = 0 \\
 \dots \leftarrow \tau = 2 & \dots \leftarrow \tau = x_1 + 1 \wedge x_1 = x_2 + 1 \wedge x_2 = 0 \\
 \vdots & \vdots \\
 \dots \leftarrow \tau = \underline{k-1} & \dots \leftarrow \tau = x_1 + 1 \wedge \dots \wedge x_{k-1} = 0 \\
 \dots \leftarrow \tau = x + \underline{k} & \dots \leftarrow \tau = x_1 + 1 \wedge \dots \wedge x_{k-1} = x + 1
 \end{array}$$

$x$  je *nová* premenná,  $k$  je *konštanta*,  $\tau$  je ľubovoľný term

Príklad bez dosadenia

a s dosadením:

$$\begin{array}{lll}
 \text{fib}_n = 0 & \leftarrow n = 0 & \text{fib}_0 = 0 \\
 \text{fib}_n = 1 & \leftarrow n = 1 & \text{fib}_1 = 1 \\
 \text{fib}_n = \text{fib}_{m+1} + \text{fib}_m & \leftarrow n = m + 2 & \text{fib}_{n+2} = \text{fib}_{n+1} + \text{fib}_n
 \end{array}$$

## 7. Chvostová rekurzia

### III.7 Iterácia

---

- Ako by ste programovali výpočet  $n!$  v Pascale?

- Pravdepodobne cyklom `while`:

```
function Fact(n: Integer): Integer;
var f: Integer;
begin
  f := 1;
  while n <> 0 do begin
    f := f * n;
    n := n - 1
  end;
  Result := f
end;
```

- Cyklus opakuje — *iteruje* výpočet vo svojom tele  
Cyklus = *iterácia*

- Ako by ste simulovali tento výpočet deklaratívne v CL?
  - Musíme použiť rekurziu
  - Nemôžeme meniť hodnoty premenných
  - Hodnota funkcie závisí iba od jej argumentov

### 7.1. Rekurzia pre cyklus `while`

#### III.8 Rekurzia pre cyklus `while` — rozdelenie programu

---

1. Pascalovský program rozdelíme na 2 časti:

a) inicializácia

```
f := 1;
```

b) cyklus a vrátenie výsledku

```

while  $n <> 0$  do begin
     $f := f * n;$ 
     $n := n - 1$ 
end;
Result :=  $f$ 

```

Aký je *matematický* význam tejto časti programu?

- ▶ Funkcia, ktorá každej hodnote  $n$  a  $f$  priradí Result
- ▶ Nazveme ju While\_fact

### III.9 Rekúzia pre cyklus while — cyklus ako funkcia

2. Naprogramujeme funkciu While\_fact tak, aby počítala to, čo cyklus:

- Dva argumenty — hodnoty pascalovských premenných  $n$  a  $f$
- Aká bude hodnota Result, keď hodnota  $n = 0$ ?  
Rovnaká ako hodnota  $f$
- Aká bude hodnota Result, keď hodnota  $n \neq 0$ ?
  - a) Vykoná sa telo cyklu  $\Rightarrow$  nové hodnoty premenných  $n$  a  $f$
  - b) Cyklus sa zopakuje s novými hodnotami

While\_fact(nová hodnota  $n$ , nová hodnota  $f$ )

$$\begin{aligned}
 \text{While\_fact}(n, f) &= \overbrace{\text{While\_fact}(n_1, f_1)}^{\text{opakovanie}} \leftarrow \overbrace{n \neq 0}^{\text{podmienka}} \wedge \underbrace{f \cdot n = f_1 \wedge n - 1 = n_1}_{\text{telo cyklu}} \\
 \text{While\_fact}(n, f) &= \underbrace{f}_{\text{výsledok}} \leftarrow \underbrace{n = 0}_{\text{negácia podmienky}}
 \end{aligned}$$

Invariant:

$$\forall f (\text{While\_fact}(n, f) = f \cdot n!)$$



### III.10 Rekurzia pre cyklus while – inicializácia

#### 3. Inicializácia a spustenie cyklu

- Hlavná funkcia Fact spustí simuláciu cyklu s počiatocnými hodnotami premenných  $n$  a  $f$ .

$\text{Fact}(n) = \text{While\_fact}(n, 1)$

### III.11 Rekurzia pre cyklus while – rekapitulácia

```
function Fact(n: Integer): Integer;
var f: Integer;
begin
```

```
  f := 1;
  while n <> 0 do begin
    f := f * n;
    n := n - 1
  end;
```

```
  Result := f
```

```
end;
```

$\text{While\_fact}(n, f) = \text{While\_fact}(n_1, f_1) \leftarrow n \neq 0 \wedge f \cdot n = f_1 \wedge n - 1 = n_1$   
 $\text{While\_fact}(n, f) = f \leftarrow n = 0$

$\text{Fact}(n) = \text{While\_fact}(n, 1)$

Stručnejšie a elegantnejšie s monadickou diskrimináciou:

$\text{While\_fact}(n + 1, f) = \text{While\_fact}(n, f \cdot (n + 1))$

$\text{While\_fact}(0, f) = f$

### III.12 Chvostová rekúzia pre cyklus while — výpočet

Priebeh výpočtu obyčajnej definície  $n!$ :

$$\begin{array}{lcl}
 5! & = & 5 \cdot 4! & & = & 5 \cdot 24 = 120 \\
 & \downarrow & & & \uparrow & \\
 & = & 5 \cdot (4 \cdot 3!) & & = & 5 \cdot (4 \cdot 6) \\
 & \downarrow & & & \uparrow & \\
 & \vdots & & & \vdots & \\
 & \downarrow & & & \uparrow & \\
 & = & 5 \cdot (4 \cdot (3 \cdot (2 \cdot 1!))) & & = & 5 \cdot (4 \cdot (3 \cdot (2 \cdot 1))) \\
 & \downarrow & & & \uparrow & \\
 & = & 5 \cdot (4 \cdot (3 \cdot (2 \cdot (1 \cdot 0!)))) & & = & 5 \cdot (4 \cdot (3 \cdot (2 \cdot (1 \cdot 1))))
 \end{array}$$

Priebeh výpočtu Fact:

$$\begin{aligned}
 \text{Fact}(5) &= \text{While\_fact}(5, 1) = \text{While\_fact}(4, 5) = \\
 &= \text{While\_fact}(3, 20) = \dots = \text{While\_fact}(0, 120) = \\
 &= 120
 \end{aligned}$$

While\_fact: *chvostová* rekúzia

- Žiadne výpočty po návrate z rekurzívneho volania
- Nepotrebuje zásobník

### III.13 Chvostová rekúzia — príklad z minulého cvičenia

Chvostovú rekúziu ste použili už na minulom cvičení pri programovaní funkcie

$$\text{Sqrt}_0(x) = y \leftrightarrow x = y^2 \vee \forall z \neg (x = z^2) \wedge y = 0$$

- Aký cyklus ste simulovali?
- Ako ste funkciu  $\text{Sqrt}_0$  naprogramovali?
- Akú vlastnosť (invariant) má pomocná funkcia simulujúca cyklus?

## 7.2. Efektívny výpočet Fibonacciho postupnosti

### III.14 Fibonacciho postupnosť *neefektívne*

---

Na pripomenutie:  $n$ -tý prvok Fibonacciho postupnosti

$$\text{fib}_0 = 0$$

$$\text{fib}_1 = 1$$

$$\text{fib}_{n+2} = \text{fib}_{n+1} + \text{fib}_n$$

Výpočet obyčajnou rekurziou je veľmi neefektívny:

$$\begin{array}{ccccccc} & & & & \text{fib}_5 & & \\ & & & & + & & \text{fib}_3 \\ & & \text{fib}_4 & & & & \text{fib}_2 + \text{fib}_1 \\ & \text{fib}_3 & + & \text{fib}_2 & & & \\ \text{fib}_2 & + & \text{fib}_1 & \text{fib}_1 + \text{fib}_0 & \text{fib}_1 + \text{fib}_0 & & \\ \text{fib}_1 + \text{fib}_0 & & & & & & \end{array}$$

Zhruba  $\text{fib}_{n+2}$  rekurzívnych volaní

$\text{fib}_n$  rastie ako  $\phi^n$ ,  $\phi = (\sqrt{5} - 1)/2 \doteq 1.618$  — exponenciálne

### III.15 Fibonacciho postupnosť a králiky

---

Efektívny výpočet  $\text{fib}_n$ :

- pamätáme si dva prvky postupnosti
- počítame nasledujúci prvok

$\Rightarrow$  stačí  $n \div 1$  krokov

Analógia — množenie králikov:

- Dostali sme 1 pár králičích mláďat.
- Za 1 rok:
  - každý pár dospelých králikov vychová 1 pár mláďat;
  - každé mláďa narodené predchádzajúci rok dospeje.

- Koľko párov dospelých králikov máme po  $n$  rokoch?

rok	mláďatá	dospelí	rok	mláďatá	dospelí
0	1	$0 = \text{fib}_0$	$i+1$	$m = \text{fib}_i$	$d = \text{fib}_{i+1}$
1	$0 = \text{fib}_0$	$1 = \text{fib}_1$	$i+2$	$d = \text{fib}_{i+1}$	$m + d = \text{fib}_{i+2}$
2	$1 = \text{fib}_1$	$1 = \text{fib}_2$			
3	$1 = \text{fib}_2$	$2 = \text{fib}_3$			
4	$2 = \text{fib}_3$	$3 = \text{fib}_4$			
5	$3 = \text{fib}_4$	$5 = \text{fib}_5$			

### III.16 Fibonacciho postupnosť efektívne imperatívne

---

```

function Fib( $n$ : Integer): Integer;
var  $m, d, m1, d1$ : Integer;
begin
  if  $n = 0$  then
    Result := 0
  else begin
     $n := n - 1$ ;
     $m := 0$ ; { fib(0) }
     $d := 1$ ; { fib(1) }
    while  $n <> 0$  do begin
      {  $m = \text{fib}(i)$  &  $d = \text{fib}(i+1)$  }
       $m1 := d$ ;
       $d1 := m + d$ ;
       $m := m1$ ;  $d := d1$ ;
      {  $m = \text{fib}(i+1)$  &  $d = \text{fib}(i+2)$  }
       $n := n - 1$ 
    end;
    Result :=  $d$ 
  end
end;

```

### III.17 Fibonacciho postupnosť efektívne deklaratívne

---

- Rekurzia pre cyklus while:

$$\begin{aligned} \text{While\_fib}(n, m, d) &= \text{While\_fib}(n \div 1, m_1, d_1) \\ &\quad \leftarrow n \neq 0 \wedge d = m_1 \wedge d + m = d_1 \\ \text{While\_fib}(n, m, d) &= d \quad \leftarrow n = 0 \end{aligned}$$

Stručnejšie

$$\begin{aligned} \text{While\_fib}(n + 1, m, d) &= \text{While\_fib}(n, d, d + m) \\ \text{While\_fib}(0, m, d) &= d \end{aligned}$$

Invariant:

$$\forall i (\text{While\_fib}(n, \text{fib}_i, \text{fib}_{i+1}) = \text{fib}_{n+i+1})$$

- Úvodné testy, inicializácia a spustenie cyklu:

$$\begin{array}{l|l} \text{Fib}(n) = 0 & \leftarrow n = 0 \\ \text{Fib}(n) = \text{While\_fib}(n \div 1, 0, 1) & \leftarrow n \neq 0 \end{array} \quad \left| \begin{array}{l} \text{Fib}(0) = 0 \\ \text{Fib}(n + 1) = \text{While\_fib}(n, 0, 1) \end{array} \right.$$

## 7.3. Rekurzia pre cyklus for

### III.18 Spätná chvostová rekurzia pre cyklus for

Výpočet faktoriálu cyklom for:

```
function Fact(n: Integer): Integer;
var f: Integer;
begin
  f := 1;
  for i := 1 to n do
    f := f * i;
  Result := f
end;
```

Deklaratívne:

- Cyklu for zodpovedá *spätná rekurzia* — rekurzívna premenná  $i$  rastie, kým neprekročí vopred danú hranicu ( $n$ )

$$\begin{aligned} \text{For\_fact}(i, n, f) &= \text{For\_fact}(i + 1, n, f \cdot i) \quad \leftarrow i \leq n \\ \text{For\_fact}(i, n, f) &= f \quad \leftarrow i > n \end{aligned}$$

- Inicializácia premenných a štart cyklu:

$$\text{Fact}(n) = \text{For\_fact}(1, n, 1)$$

### III.19 Spätná vs. primitívna rekúzia

---

Spätná rekúzia, napríklad

$$\begin{aligned} \text{For\_fact}(i, n, f) &= \text{For\_fact}(i + 1, n, f \cdot i) \leftarrow i \leq n \\ \text{For\_fact}(i, n, f) &= f \quad \leftarrow i > n, \end{aligned}$$

sa dá ľahko nahradiť primitívnou rekúziou (rekurzívny argument klesá o 1).  
Ako?

# IV. prednáška

## Dyadická číselná sústava

### Dyadické reťazce

4. marca 2013

#### 7.4. Opakovanie

##### IV.1 Opakovanie: Priradujúce diskriminácie

##### Priradenie

Premenná ako skratka za hodnotu termu

$$\begin{aligned} F(n) = 1 & \leftarrow n = 0 \\ F(n) = n \cdot F(m) & \leftarrow n \neq 0 \wedge n - 1 = m \end{aligned}$$

##### Monadická diskriminácia

Kombinácia testu na 0 a priradenia s odčítaním 1

Zapísaná formou porovnania so vzorom

$$\begin{array}{l|l} F(n) = 1 & \leftarrow n = 0 \\ F(n) = n \cdot F(m) & \leftarrow n = m + 1 \end{array} \quad \left| \quad \begin{array}{l} F(0) = 1 \\ F(m + 1) = (m + 1) \cdot F(m) \end{array} \right.$$

##### Zovšeobecnená monadická diskriminácia

Skratka kaskády monadických diskriminácií

$$\begin{array}{l|l} F(n) = 0 & \leftarrow n = 0 \\ F(n) = 1 & \leftarrow n = 1 \\ F(n) = F(m + 1) + F(m) & \leftarrow n = m + 2 \end{array} \quad \left| \quad \begin{array}{l} F(0) = 0 \\ F(1) = 1 \\ F(n + 2) = F(n + 1) + F(n) \end{array} \right.$$

### Chvostová rekúzia

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

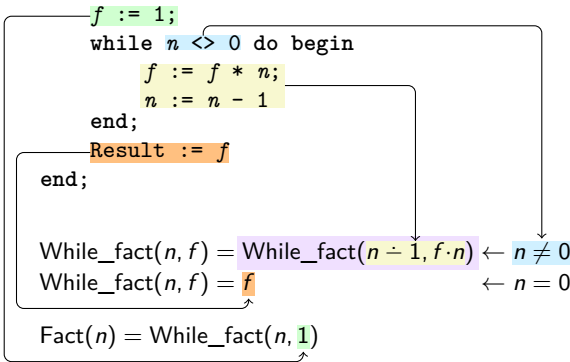
$$F(x, y, z) = {}_0F(\dots, \dots, \dots)_0 \leftarrow \dots$$

- Hodnota funkcie v rekúziívnom prípade = hodnote rekúziívneho volania b
- Výpočty iba v argumentoch rekúziívneho volania
- Pri výpočte nie je potrebný zásobník
- Do strojového kódu sa kompiluje ako cyklus

### Cyklus while ~ chvostová rekúzia riadená podmienkou cyklu

- premenné v cykle ~ argumenty funkcie
- inicializácia ~ volanie simulačnej funkcie hlavnou funkciou

```
function Fact(n: Integer): Integer;
var f: Integer;
begin
```

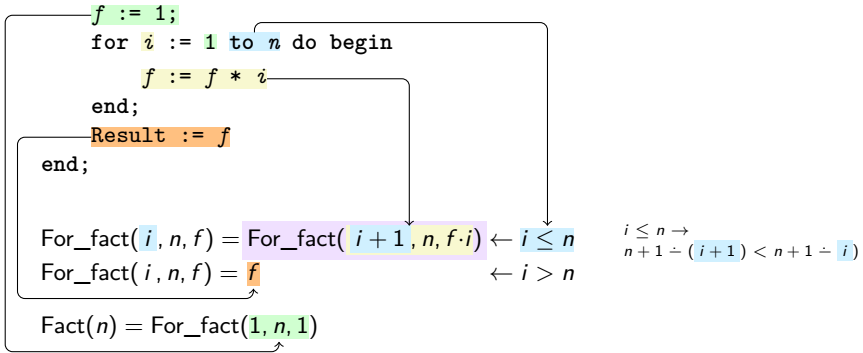




**Cyklus for** ~ spätná chvostová rekúzia

- premenné v cykle ~ argumenty funkcie
- inicializácia ~ volanie simulačnej funkcie hlavnou funkciou

```
function Fact(n: Integer): Integer;
var f: Integer;
begin
```



## 8. Dyadická číselná sústava

IV.5 Binárna číselná sústava

- Každé prirodzené číslo sa dá zapísať číslicami 0 a 1 v *binárnej* (dvojkovej) sústave (podobne ako číslicami 0 až 9 v desiatkovej sústave)
- Napríklad

$$17 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (10001)_b$$

- Vo všeobecnosti

$$\sum_{i=0}^k b_i \cdot 2^i = (b_k b_{k-1} \dots b_1 b_0)_b, \quad 0 \leq b_i \leq 1$$

- Zápis nie je jednoznačný — môžeme pridať ľubovoľne veľa 0 pred číslicu najvyššieho rádu:

$$\begin{aligned} & (b_k b_{k-1} \dots b_1 b_0)_b = \\ & = (0b_k b_{k-1} \dots b_1 b_0)_b = \\ & = (00b_k b_{k-1} \dots b_1 b_0)_b = \\ & \quad \quad \quad = \dots \end{aligned}$$

#### IV.6 Dyadická číselná sústava

---

- Každé prirodzené číslo sa dá *jednoznačne* zapísať v *dyadickej* sústave ako postupnosť začínajúca 0 a pokračujúca konečným počtom číslic **1** a **2**
- Napríklad

$$17 = \mathbf{1} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{2} \cdot 2^1 + \mathbf{1} \cdot 2^0 = \mathbf{01121}$$

- Vo všeobecnosti

$$\sum_{i=1}^k d_i \cdot 2^{i-1} = 0d_k \dots d_2 d_1, \quad 1 \leq d_i \leq 2$$

- V Hornerovej schéme:

$$\begin{aligned} 17 &= (((0 \cdot 2 + \mathbf{1}) \cdot 2 + \mathbf{1}) \cdot 2 + \mathbf{2}) \cdot 2 + \mathbf{1} \\ \sum_{i=1}^k d_i \cdot 2^{i-1} &= ((\dots (0 \cdot 2 + d_k) \dots) \cdot 2 + d_2) \cdot 2 + d_1 \end{aligned}$$

### 8.1. Dyadické konštruktory

#### IV.7 Dyadické konštruktory

---

- Zadefinujme

$$S_1(n) = 2 \cdot n + 1$$

$$S_2(n) = 2 \cdot n + 2$$

- Potom môžeme zapísať

$$17 = 01121$$

$$= (((0 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 2) \cdot 2 + 1$$

$$= S_1(((0 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 2)$$

$$= S_1 S_2((0 \cdot 2 + 1) \cdot 2 + 1)$$

$$= S_1 S_2 S_1(0 \cdot 2 + 1)$$

$$= S_1 S_2 S_1 S_1(0)$$

$$\sum_{1 \leq i \leq k} d_i \cdot 2^i = 0d_k \dots d_2 d_1$$

$$= (((\dots(0 \cdot 2 + d_k) \dots) \cdot 2 + d_2) \cdot 2 + d_1$$

$$= S_{d_1}(\dots(0 \cdot 2 + d_k) \dots) \cdot 2 + d_2$$

$$= S_{d_1} S_{d_2}(\dots(0 \cdot 2 + d_k) \dots)$$

$$= S_{d_1} S_{d_2} \dots S_{d_{k-1}}(0 \cdot 2 + d_k)$$

$$= S_{d_0} S_{d_1} \dots S_{d_k}(0)$$

- Takže:

$$S_1(n) = n1$$

$$S_2(n) = n2$$

- Funkcie  $S_1$  a  $S_2$  nazývame *dyadické konštruktory*

## 8.2. Dyadická diskriminácia

### IV.8 Testovanie dyadického zápisu

Dané je číslo  $n = 0d_k \dots d_2 d_1 = \sum_{i=1}^k d_i \cdot 2^{i-1}$ .

1. Kedy nemá  $n$  žiadnu dyadickú číslicu?

▶  $n = 0$

2. Kedy má  $n$  aspoň jednu dyadickú číslicu?

▶  $n \neq 0$

- a) Ako potom určíme najmenej významnú číslicu čísla  $n$ ?

▶  $\text{Last}_2(n) = d_1 = 2 \div (n \bmod 2)$

- b) Ako určíme číslo, ktoré vznikne z  $n$  odobratím najmenej významnej číslice?

▶  $\text{Init}_2(n) = 0d_k \dots d_2 = \sum_{i=1}^{k-1} d_{i+1} \cdot 2^{i-1} = (n \div 2) \div 2$

Naprogramujme zmenu poslednej dyadickej číslice:

▶  $\text{Flip\_last}_2(n) = 0 \quad \leftarrow n = 0$

$\text{Flip\_last}_2(n) = S_2(m) \quad \leftarrow n \neq 0 \wedge \text{Init}_2(n) = m \wedge \text{Last}_2(n) = 1$

$\text{Flip\_last}_2(n) = S_1(m) \quad \leftarrow n \neq 0 \wedge \text{Init}_2(n) = m \wedge \text{Last}_2(n) = 2$

V CL existuje elegantnejší spôsob...

#### IV.9 Dyadická diskriminácia

CL pozná *dyadickú diskrimináciu*:

- Každé číslo je buď 0, alebo sa dá rozdeliť na najmenej významnú dyadickú číslicu a významnejší zvyšok  $m$

$$\begin{array}{ll} \dots \leftarrow \tau = 0 & \dots \leftarrow \tau = 0 \\ \dots \leftarrow \tau = m\mathbf{1} & \dots \leftarrow \tau = S_1(m) \\ \dots \leftarrow \tau = m\mathbf{2} & \dots \leftarrow \tau = S_2(m) \end{array}$$

$\tau$  je term so známou hodnotou,  $m$  je nová premenná

Tieto prípady pokrývajú všetky možnosti

$$n = 0 \vee \exists m(n = m\mathbf{1}) \vee \exists m(n = m\mathbf{2})$$

a sú vzájomne vylučné:

$$\begin{aligned} &\neg(n = 0 \wedge \exists m(n = m\mathbf{1})) \wedge \neg(n = 0 \wedge \exists m(n = m\mathbf{2})) \wedge \\ &\wedge \neg(\exists m(n = m\mathbf{1}) \wedge \exists m(n = m\mathbf{2})) \end{aligned}$$

#### IV.10 Použitie dyadickej diskriminácie

v tele klauzuly

$$\begin{aligned} \text{Flip\_last}_2(n) = 0 &\leftarrow n = 0 \\ \text{Flip\_last}_2(n) = m\mathbf{2} &\leftarrow n = m\mathbf{1} \\ \text{Flip\_last}_2(n) = m\mathbf{1} &\leftarrow n = m\mathbf{2} \end{aligned}$$

v argumente funkcie

$$\begin{aligned} \text{Flip\_last}_2(0) &= 0 \\ \text{Flip\_last}_2(m\mathbf{1}) &= m\mathbf{2} \\ \text{Flip\_last}_2(m\mathbf{2}) &= m\mathbf{1} \end{aligned}$$

### 8.3. Dyadická rekurzia

#### IV.11 Dyadická rekurzia

*Dyadická rekurzia* — argument rekurzívneho volania je pôvodný argument bez poslednej (najmenej významnej) dyadickej číslice:

$$\begin{array}{ll} F(n, \dots) = \dots & \leftarrow n = 0 \\ F(n, \dots) = \dots F(m, \dots) \dots & \leftarrow n = m\mathbf{1} \\ F(n, \dots) = \dots F(m, \dots) \dots & \leftarrow n = m\mathbf{2} \end{array} \quad \left| \quad \begin{array}{l} F(0, \dots) = \dots \\ F(m\mathbf{1}, \dots) = \dots F(m, \dots) \dots \\ F(m\mathbf{2}, \dots) = \dots F(m, \dots) \dots \end{array}$$

Pretože platí

$$n < n1 = 2 \cdot n + 1, \quad n < n2 = 2 \cdot n + 2,$$

dyadická rekurgia je špeciálny prípad všeobecnej rekurgie

#### IV.12 Dyadická rekurgia — príklad a dĺžka rekurgie

Napríklad  $\text{Len}(n)$  — počet dyadických číslic čísla  $n$ :

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

$$\text{Len}(n1) = 1 + \text{Len}(n)$$

$$\text{Len}(n2) = 1 + \text{Len}(n)$$

Koľko krokov môže najviac dyadická rekurgia pre  $n$ ?  $\text{Len}(n)$

Ktorá analytická funkcia tomu približne zodpovedá?  $\log_2 n$

## 8.4. Aritmetické operácie na dyadickom zápise čísel

#### IV.13 Efektívna aritmetika s veľkými číslami

##### Neefektívna definícia aritmetickej operácie:

počet krokov = *hodnota* niektorého argumentu

Napríklad násobenie primitívnou rekurgiou

$$\text{Mul}(x, 0) = 0$$

$$\text{Mul}(x, y + 1) = x + \text{Mul}(x, y)$$

##### Efektívna definícia aritmetickej operácie:

počet krokov = *počet číslic* (logaritmus) niektorého argumentu

- Napríklad dyadickou rekurgiou
- Podobný princíp ako aritmetika v desiatkovom zápise (základná škola), ale menej číslic

#### IV.14 Nasledovník na dyadickom zápise

Začnime nasledovníkom (successor): funkcia  $\text{Succ}(x) = x + 1$

$x$	0 0	1 01	2 02	3 011	4 012	5 021	6 022	...
$x + 1$	1 01	2 02	3 011	4 012	5 021	6 022	7 0111	...

Príklady:

$$\begin{array}{r} 0 \\ + 01 \\ \hline 01 \end{array}$$

$$\begin{array}{r} 021 = 5 \\ + 01 \\ \hline 022 = 6 \end{array}$$

$$\begin{array}{r} 0212 = 12 \\ + 01 \\ \hline 021 \\ + 011 \\ \hline 0221 = 13 \end{array}$$

Zovšeobecnenie:

$$\begin{array}{r} 0 \\ + 01 \\ \hline 01 \end{array}$$

$$\begin{array}{r} x1 \\ + 01 \\ \hline x2 \end{array}$$

$$\begin{array}{r} x2 \\ + 01 \\ \hline x \\ + 011 \\ \hline (x + 1)1 \end{array}$$

#### IV.15 Nasledovník na dyadickom zápise

Algebraické odvodenie

- predpokladáme, že  $\text{Succ}(n)$  počíta  $n + 1$
- hľadáme vyjadrenie zamýšľaného výsledku iba použitím:
  - čísla 0
  - dyadických konštruktorov  $S_1(n) = n1$  a  $S_2(n) = n2$
  - rekurzie so zvyšnými číslicami dyadického zápisu

$$\text{Succ}(0) = 0 + 1 = 1 = 2 \cdot 0 + 1 = 01$$

$$\text{Succ}(x1) = x1 + 1 = (2 \cdot x + 1) + 1 = 2 \cdot x + 2 = x2$$

$$\begin{aligned} \text{Succ}(x2) &= x2 + 1 = (2 \cdot x + 2) + 1 = 2 \cdot (x + 1) + 1 = (x + 1)1 \\ &= (\text{Succ}(x))1 \end{aligned}$$

Klauzálna definícia použitím dyadickej rekurzie:

$$\begin{array}{l|l}
\text{Succ}(x) = 01 & \leftarrow x = 0 \\
\text{Succ}(x) = v2 & \leftarrow x = v1 \\
\text{Succ}(x) = (\text{Succ}(v))1 & \leftarrow x = v2
\end{array}
\quad \Bigg| \quad
\begin{array}{l}
\text{Succ}(0) = 01 \\
\text{Succ}(v1) = v2 \\
\text{Succ}(v2) = (\text{Succ}(v))1
\end{array}$$

#### IV.16 Sčítanie na dyadickom zápise

---

Príklady sčítania:

$$\begin{array}{r}
\phantom{+} 0 \\
+ 021 \\
\hline
021
\end{array}
\qquad
\begin{array}{r}
\phantom{+} 021 \\
+ 0111 \\
\hline
0212
\end{array}
\qquad
\begin{array}{r}
\phantom{+} 0212 \\
+ \phantom{0} 012 \\
\hline
01112
\end{array}$$

Zovšeobecnenie:

$$\begin{array}{r}
\phantom{+} 0 \\
+ y \\
\hline
y
\end{array}
\qquad
\begin{array}{r}
\phantom{+} x1 \\
+ 0 \\
\hline
x1
\end{array}
\qquad
\begin{array}{r}
\phantom{+} x1 \\
+ y1 \\
\hline
(x+y)2
\end{array}
\qquad
\begin{array}{r}
\phantom{+} x1 \\
+ y2 \\
\hline
?
\end{array}$$

$$\begin{array}{r}
\phantom{+} x2 \\
+ 0 \\
\hline
x2
\end{array}
\qquad
\begin{array}{r}
\phantom{+} x2 \\
+ y1 \\
\hline
?
\end{array}
\qquad
\begin{array}{r}
\phantom{+} x2 \\
+ y2 \\
\hline
?
\end{array}$$

#### IV.17 Sčítanie na dyadickom zápise

---

Klauzálna definícia dyadickou rekurziou s dvojitou dyadickou diskrimináciou

Odvodíme podľa príkladu z predchádzajúcej snímky alebo algebraicky použitím:

- čísla 0, dyadických konštruktorov  $S_1(n) = n1$  a  $S_2(n) = n2$
- nasledovníka
- rekurzie so zvyšnými číslicami dyadického zápisu

$$\text{Add}(0, y) = 0 + y = y$$

$$\text{Add}(x1, 0) = x1 + 0 = x1$$

$$\begin{aligned}\text{Add}(x1, y1) &= x1 + y1 = (2 \cdot x + 1) + (2 \cdot y + 1) = 2 \cdot (x + y) + 2 = \\ &= (x + y)2 = (\text{Add}(x, y))2\end{aligned}$$

$$\begin{aligned}\text{Add}(x1, y2) &= x1 + y2 = (2 \cdot x + 1) + (2 \cdot y + 2) = (2 \cdot x + 2 \cdot y + 2) + 1 = \\ &= 2 \cdot (x + y + 1) + 1 = ((x + y) + 1)1 = (\text{Succ}(\text{Add}(x, y)))1\end{aligned}$$

$$\text{Add}(x2, 0) = x2 + 0 = \dots$$

$$\text{Add}(x2, y1) = x2 + y1 = \dots \text{Add}(x, y) \dots$$

$$\text{Add}(x2, y2) = x2 + y2 = \dots \text{Add}(x, y) \dots$$

#### IV.18 Násobenie na dyadickom zápise

Potrebujeme pomocnú funkciu  $\text{Twice}(x) = 2 \cdot x$  (podobne ako sme pri sčítaní potrebovali nasledovníka):

$$\text{Twice}(0) = 2 \cdot 0 = 0$$

$$\text{Twice}(x1) = 2 \cdot (x1) = 2 \cdot (2 \cdot x + 1) = \dots \text{Twice}(x) \dots$$

$$\text{Twice}(x2) = 2 \cdot (x2) = \dots \quad (\text{bez rekurzie})$$

Algebraické odvodenie násobenia:

$$\text{Mul}(0, y) = 0 \cdot y = 0$$

$$\text{Mul}(x1, y) = x1 \cdot y = (2 \cdot x + 1) \cdot y = 2 \cdot x \cdot y + 1 \cdot y = \dots \text{Mul}(x, y) \dots$$

$$\text{Mul}(x2, y) = x2 \cdot y = (2 \cdot x + 2) \cdot y = \dots \text{Mul}(x, y) \dots$$



## V. prednáška

# Symbolické operácie na dyadickom zápise. Predikáty. Párovanie.

11. marca 2013

## Opakovanie

### V.1 Opakovanie – dyadický zápis čísel

---

- Dyadický zápis čísel konštruujeme z konštanty 0 konštruktormi

$$S_1(x) = x1 = 2 \cdot x + 1 \quad S_2(x) = x2 = 2 \cdot x + 2$$

- Dyadický zápis čísel testujeme dyadickou diskrimináciou:

$$\dots \leftarrow \tau = 0$$

$$\dots \leftarrow \tau = 0$$

$$\dots \leftarrow \tau = m1$$

$$\dots \leftarrow \tau = S_1(m)$$

$$\dots \leftarrow \tau = m2$$

$$\dots \leftarrow \tau = S_2(m)$$

### V.2 Opakovanie – dyadická aritmetika

---

Predpis pre implementáciu aritmetickej operácie v dyadickej aritmetike najľahšie odvodíme algebraickými úpravami:

$$0 + y = y$$

$$x1 + 0 = x1$$

$$x1 + y1 = (2 \cdot x + 1) + (2 \cdot y + 1) = 2 \cdot (x + y) + 2 = (x + y)2$$

$$x1 + y2 = (2 \cdot x + 1) + (2 \cdot y + 2) = (2 \cdot x + 2 \cdot y + 2) + 1 =$$

$$= 2 \cdot (x + y + 1) + 1 = ((x + y) + 1)1$$

...

Hľadáme rekurziu s menším počtom dyadických číslic,  
použitie už zadaných funkcií a konštruktorov  $S_1, S_2$

## 8.5. Symbolické operácie na dyadickom zápise čísel

### V.3 Symbolický pohľad na dyadický zápis

---

- Dyadický zápis čísla môžeme chápať *symbolicky* ako *reťazec* znakov (postupnosť symbolov) 1 a 2:
  - 0** prázdny reťazec
  - n1** neprázdny reťazec končiaci symbolom 1
  - n2** neprázdny reťazec končiaci symbolom 2
- Na dyadických reťazcoch môžeme vykonávať symbolické operácie:
  - Zistenie dĺžky reťazca (počtu znakov)
  - Zreťazenie dvoch reťazcov
  - Obrátenie reťazca
  - Vyhľadanie podreťazca
  - ...

### V.4 Dĺžka dyadického reťazca

---

Dĺžku reťazca vypočítame dyadickou rekurziou veľmi jednoducho:

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

$$\text{Len}(x1) = \text{Len}(x) + 1$$

$$\text{Len}(x2) = \text{Len}(x) + 1$$

$$\begin{aligned}\text{Len}(01221) &= \text{Len}(0122) + 1 = \\ &= \text{Len}(012) + 1 + 1 = \\ &= \text{Len}(01) + 1 + 1 + 1 = \\ &= \text{Len}(0) + 1 + 1 + 1 + 1 = \\ &= 0 + 1 + 1 + 1 + 1 = 4\end{aligned}$$

## V.5 Zreťazenie dyadických reťazcov

---

Zreťazenie spojí dva reťazce do nového reťazca, pričom vzájomné poradie znakov ostane zachované:

$$02112 \star 0221 = (02112)221 = 02112221$$

$$0d_{n+k} \dots d_{2+k}d_{1+k} \star 0d_k \dots d_2d_1 = 0d_{n+k} \dots d_{2+k}d_{1+k}d_k \dots d_2d_1$$

Ako vybrať rekurzívny argument?

- Porovnajme očakávané výsledky po odobratí posledného znaku:

– z prvého argumentu:

$$02112 \star 0221 = 02112221$$

$$0211 \star 0221 = 0211221$$

– z druhého argumentu:

$$02112 \star 0221 = 02112221$$

$$02112 \star 022 = 0211222$$

V ktorom prípade ľahko skonštruujeme úplný výsledok z výsledku rekurzívneho volania?

## V.6 Zreťazenie dyadických reťazcov – riešenie

---

$$x \star 0 = x$$

$$x \star (y1) = (x \star y)1$$

$$x \star (y2) = (x \star y)2$$

$$02112 \star 0221 = (02112 \star 022)1$$

$$= (02112 \star 02)21$$

$$= (02112 \star 0)221$$

$$= (02112)221$$

$$= 02112221$$

## V.7 Obrátenie dyadického reťazca

---

$\text{Rev}(x)$  je dyadický reťazec, ktorý obsahuje všetky znaky reťazca  $x$  v obrátenom poradí

$$\text{Rev}(01121) = 12110$$

$$\text{Rev}(0d_k \dots d_2d_1) = 0d_1d_2 \dots d_k$$

Ako nájsť rekurzívnu definíciu  $\text{Rev}$ ?

- Porovnajme očakávané výsledky po odobratí posledného znaku:

$$\text{Rev}(01121) = 12110$$

$$\text{Rev}(0112) = 2110$$

Ako dostaneme posledný znak na začiatok výsledku rekurzívneho volania?

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

$$\text{Rev}(x1) = \dots \text{Rev}(x) \dots$$

$$\text{Rev}(x2) = \dots \text{Rev}(x) \dots$$

## 9. Predikáty

### V.8 Predikáty

---

- *Predikát* je vlastnosť objektu alebo  $n$ -tice objektov

- číslo  $x$  je nepárne

$$\text{Odd}(x) \leftrightarrow \exists q(x = 2 \cdot q + 1)$$

- číslo  $d$  delí číslo  $x$

$$\text{Divides}(d, x) \leftrightarrow \exists q(x = q \cdot d)$$

- $x$  je prázdny dyadický reťazec

$$\text{Empty}(x) \leftrightarrow x = 0$$

- dyadický reťazec  $x$  je palindróm

$$\text{Palindrome}(x) \leftrightarrow x = \text{Rev}(x)$$

- $z$  je sufixom (koncovým úsekom) dyadického reťazca  $x$

$$\text{Suffix}(x, z) \leftrightarrow \exists y(x = y * z)$$

...

## V.9 Charakteristické funkcie

---

Namiesto predikátu  $P$  môžeme naprogramovať jeho *charakteristickú funkciu*  $P_*$ :

$$P_*(x) = \begin{cases} 1 & \text{ak } P(x) \text{ platí,} \\ 0 & \text{ak } P(x) \text{ neplatí} \end{cases}$$

Napríklad

$$\text{Palindrome}_*(x) = 1 \leftarrow x = \text{Rev}(x)$$

$$\text{Palindrome}_*(x) = 0 \leftarrow x \neq \text{Rev}(x)$$

$$\text{Suffix}_*(x, 0) = 1$$

$$\text{Suffix}_*(x, z1) = 0 \quad \leftarrow x = 0$$

$$\text{Suffix}_*(x, z1) = \text{Suffix}_*(y, z) \leftarrow x = y1$$

$$\text{Suffix}_*(x, z1) = 0 \quad \leftarrow x = y2$$

$$\text{Suffix}_*(x, z2) = 0 \quad \leftarrow x = 0$$

$$\text{Suffix}_*(x, z2) = 0 \quad \leftarrow x = y1$$

$$\text{Suffix}_*(x, z2) = \text{Suffix}_*(y, z) \leftarrow x = y2$$

## V.10 Definovanie predikátov v CL

---

- V CL môžeme predikáty definovať priamo
  - podobne ako funkcie
  - *vynechávame* klauzuly pre prípady, v ktorých predikát *neplatí*

- Napríklad

$$\text{Palindrome}(x) \leftarrow x = \text{Rev}(x)$$
~~$$\neg \text{Palindrome}(x) \leftarrow x \neq \text{Rev}(x)$$~~

$$\text{Suffix}(x, 0)$$
~~$$\neg \text{Suffix}(x, z1) \leftarrow x = 0$$~~
~~$$\neg \text{Suffix}(x, z1) \leftarrow x = y1 \wedge \neg \text{Suffix}(y, z)$$~~
~~$$\neg \text{Suffix}(x, z1) \leftarrow x = y2$$~~

$$\text{Suffix}(x, z1) \leftarrow x = y1 \wedge \text{Suffix}(y, z)$$
~~$$\neg \text{Suffix}(x, z1) \leftarrow x = y2$$~~
~~$$\neg \text{Suffix}(x, z2) \leftarrow x = 0$$~~
~~$$\neg \text{Suffix}(x, z2) \leftarrow x = y1$$~~
~~$$\neg \text{Suffix}(x, z2) \leftarrow x = y2 \wedge \neg \text{Suffix}(y, z)$$~~

$$\text{Suffix}(x, z2) \leftarrow x = y2 \wedge \text{Suffix}(y, z)$$

## V.11 Diskriminácia na platnosť predikátu

---

- Diskriminácia na platnosť predikátu

$$\dots \leftarrow P(x)$$

$$\dots \leftarrow \neg P(x)$$

- $\neg P(x)$  zapisujeme  $\sim P(x)$

## 10. Párovanie

### 10.1. Kódovanie dátových štruktúr

#### V.12 Dátové štruktúry

---

- Programy zriedka operujú iba na číslach
- Zvyčajne používame dátové štruktúry, napríklad ...
- Funkcie v CL pracujú iba s prirodzenými číslami (s ľubovoľnou presnosťou)

- Pomocou dyadického zápisu sme *zakódovali* do prirodzených čísel dátovú štruktúru: reťazce symbolov 1 a 2
- Najjednoduchšia dátová štruktúra: *usporiadaná dvojica*
  - Umožňuje vybudovať väčšinu prakticky potrebných dátových štruktúr (uvidíme neskôr)
- V Pascale: ...?
- Ako zakódujeme usporiadané dvojice v CL?

## 10.2. Párovacie funkcie

### V.13 Kódovanie usporiadaných dvojíc — párov

---

Ako zakódujeme usporiadané dvojice (*páry*) v CL?

- Každému páru  $(x, y)$  *jednoznačne* priradíme číslo  $P(x, y)$ 
  - Potom môžeme z čísla dekodovať dvojicu
- $P(x, y)$  je teda *injektívna* funkcia z  $\mathbb{N} \times \mathbb{N}$  do  $\mathbb{N}$
- Výhodné je, ak niektorému číslu (napr. 0) nie je priradená dvojica
  - také číslo je *atóm*
  - môže predstavovať nil
 a všetky ostatné čísla predstavujú dvojice
- $P(x, y)$  je teda *surjektívna* funkcia z  $\mathbb{N} \times \mathbb{N}$  na  $\mathbb{N} \setminus \{0\}$
- Súhrnne  $P: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \setminus \{0\}$  je *bijektívna* funkcia

### V.14 Párovacie funkcie

---

**Definícia 1.** *Párovacou funkciou* nazývame každé zobrazenie  $P: \mathbb{N}^2 \rightarrow \mathbb{N} \setminus \{0\}$ , pre ktoré platí:

- injektívnosť:

$$P(x, y) = P(u, v) \rightarrow x = u \wedge y = v$$

- surjektívnosť:

$$x = 0 \vee \exists u \exists v (x = P(u, v))$$

- pár je väčší ako jeho zložky:

$$x < P(x, y) \wedge y < P(x, y)$$

### V.15 Párovacie funkcie — príklady

---

- Párovacích funkcií je nekonečne veľa

- Napríklad:

– modifikovaná Cantorova funkcia

$x \setminus y$	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	31	...
4	15	20	26	32	40	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

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

$\leftarrow x+y=d$

– modifikované „zipsovanie“ binárnych zápisov

$x \setminus y$	0	1	2	3	4	...
0	1	3	9	11	33	...
1	2	4	10	12	34	...
2	5	7	13	15	37	...
3	6	8	14	16	38	...
4	17	19	25	27	49	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$$P'_b(x, y) = P_b(x, y) + 1$$

$$P_b(x, y) = 0 \leftarrow x+y = 0$$

$$P_b(x, y) = 4 \cdot P_b(x \div 2, y \div 2) + 2 \cdot (y \bmod 2) + x \bmod 2$$

$\leftarrow x+y \neq 0$



## 10.2.1. Párovacia funkcia CL

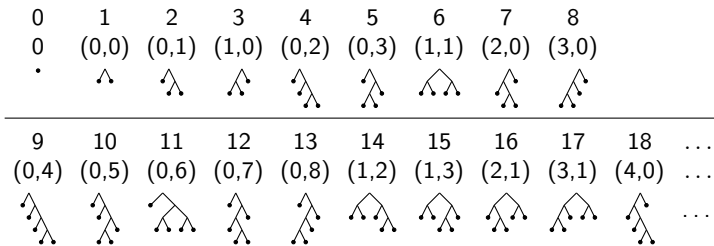
### V.16 Párovacia funkcia CL

Párovacia funkcia zabudovaná v CL:

- Zapisuje sa binárnym operátorom  $\wedge$ , (čiarka)
- Založená na očíslovaní binárnych stromov
- Na aplikáciu párovacej funkcie sa pozeráme ako na vytvorenie stromu z dvoch podstromov
- Požadujeme, aby  $t < s$  práve vtedy, keď
  - $t$  má menej uzlov ako  $s$ , alebo
  - $t = x, y$  má rovnaký počet uzlov ako  $s = u, v$  a
    - \*  $x < u$  alebo
    - \*  $x = u$  a  $y < v$ .

### V.17 Párovacia funkcia CL

- Kombináciou predchádzajúcej vlastnosti so základnými párovacími vlastnosťami dostaneme nasledovné číslovanie stromov:



- Fragment tabuľky párovacej funkcie ( $\wedge$ ):

$x \setminus y$	0	1	2	3	4	...
0	1	2	7	8	18	...
1	3	6	16	17	46	...
2	4	14	42	44	131	...
3	5	15	43	45	132	...
4	9	37	121	126	399	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

## V.18 Párovacia funkcia CL

---

- Syntax operátora ( $()$ ) v CL:
  - najnižšia priorita:  $1 + 2, 3 + 4 \equiv ((1 + 2), (3 + 4))$
  - implicitne sa zátvorkuje doprava:  $1, 2, 3, 4 \equiv 1, (2, (3, 4))$
- Výpočet založený na Catalanových číslach, náročný
- Párovacia funkcia sa počíta „lenivo“
  - iba keď je naozaj požadovaná číselná hodnota
  - inak si CL pamätá pôvodnú dvojicu čísel
- Projekčné funkcie:
  - $H(x)$  vráti prvú zložku (hlavu, head) páru  $x$  (alebo 0 ak  $x = 0$ )
  - $T(x)$  vráti druhú zložku (chvost, tail) páru  $x$  (alebo 0 ak  $x = 0$ )

Platí teda:

$$\begin{array}{ll} H(u, v) = u & H(0) = 0 \\ T(u, v) = v & T(0) = 0 \end{array}$$

## V.19 Usporiadané $n$ -tice

---

- Párovacia funkcia kóduje usporiadané dvojice čísel
- Usporiadané trojice kódujeme vnorením dvojíc doprava

$$x_1, (x_2, x_3) \equiv x_1, x_2, x_3$$

Všimnite si implicitné zátvorkovanie doprava

- Usporiadané  $n$ -tice:

$$x_1, (x_2, (x_3, \dots (x_{n-1}, x_n) \dots)) \equiv x_1, x_2, x_3, \dots, x_{n-1}, x_n$$

## 10.2.2. Programovanie s párovacou funkciou

### V.20 Programovanie s párovacou funkciou

---

- Pár vytvoríme termom „ $\sigma, \tau$ “
- Párová diskriminácia:

$$\dots \leftarrow \tau = 0$$

$$\dots \leftarrow \tau = u, v$$

Druhý prípad priradí zložky páru do nových premenných  $u$  a  $v$

- Párová diskriminácia je skratkou za:

$$\dots \leftarrow \tau = 0$$

$$\dots \leftarrow \tau \neq 0 \wedge H(\tau) = u \wedge T(\tau) = v$$

- Projekcie  $H$  a  $T$  *nebudeme* používať

### V.21 Programovanie s párovacou funkciou – príklad

---

- Príklad: funkcia počítajúca  $n$ -tý a  $(n+1)$ -ý prvok Fibonacciho postupnosti:  $\text{Twofib}(n) = (\text{fib}_n, \text{fib}_{n+1})$

$$\text{Twofib}(0) = 0, 1$$

$$\text{Twofib}(n+1) = 0 \quad \leftarrow \text{Twofib}(n) = 0$$

$$\text{Twofib}(n+1) = b, a + b \quad \leftarrow \text{Twofib}(n) = a, b$$

- Prípad v druhej klauzule nemôže nastať

Môžeme ju vynechať, CL si výsledok 0 doplní *defaultom*

$$\text{Twofib}(0) = 0, 1$$

$$\text{Twofib}(n+1) = b, a + b \quad \leftarrow \text{Twofib}(n) = a, b$$

- Defaulty slúžia pre prípady, keď výpočet *nemá zmysel*

*Nevynechávajú* klauzuly, keď funkcia vracia 0 ako riadny, očakávaný výsledok!

$$\text{Twofib}(0) = 0, 1$$

$$\text{Twofib}(n + 1) = b, a + b \leftarrow \text{Twofib}(n) = a, b$$

- Výpočet:

$$\begin{array}{rcl}
 \text{Twofib}(5) & = & 5, 8 \\
 & \downarrow & \uparrow \\
 \text{Twofib}(4) & = & 3, 5 \\
 & \downarrow & \uparrow \\
 \text{Twofib}(3) & = & 2, 3 \\
 & \downarrow & \uparrow \\
 \text{Twofib}(2) & = & 1, 2 \\
 & \downarrow & \uparrow \\
 \text{Twofib}(1) & = & 1, 1 \\
 & \downarrow \nearrow & \\
 \text{Twofib}(0) & = & 0, 1
 \end{array}$$

- CL nepočíta číselnú hodnotu párovacej funkcie, pokiaľ to nie je nevyhnutné

V pamäti vytvorí pascalovský record

### 10.2.3. Tupling

- Pomocou párovania môžeme súčasne počítať funkcie,
  - ktoré majú podobnú štruktúru (diskriminácie, rekurzívne argumenty)
  - a ich výsledky potrebujeme súčasne

Technika sa nazýva *tupling*

- Napríklad celočíselné delenie a zvyšok:

$$\begin{aligned}
x \div y &= 0 && \leftarrow y = 0 \\
x \div y &= 0 && \leftarrow y \neq 0 \wedge x < y \\
x \div y &= ((x \div y) \div y) + 1 && \leftarrow y \neq 0 \wedge x \geq y \\
x \bmod y &= 0 && \leftarrow y = 0 \\
x \bmod y &= x && \leftarrow y \neq 0 \wedge x < y \\
x \bmod y &= (x \div y) \bmod y && \leftarrow y \neq 0 \wedge x \geq y
\end{aligned}$$

spojíme do jednej funkcie

$$\text{Divmod}(x, y) = x \div y, x \bmod y$$

## V.24 Tupling

---

- Spojená funkcia:

$$\begin{aligned}
\text{Divmod}(x, y) &= 0, 0 \leftarrow y = 0 \\
\text{Divmod}(x, y) &= 0, x \leftarrow y \neq 0 \wedge x < y \\
\text{Divmod}(x, y) &= q + 1, r \leftarrow y \neq 0 \wedge x \geq y \wedge \text{Divmod}(x \div y, y) = q, r
\end{aligned}$$

## VI. prednáška

# Celočíselná aritmetika. Opakovanie

18. marca 2013

### 10.3. Kódovanie celých čísel párovaním

#### VI.1 Celé čísla a usporiadané dvojice

---

- Čo majú spoločné dvojice prirodzených čísel v týchto množinách:

$$K = \{(5341, 5339), (59, 57), (6, 4), (2, 0), (3, 1), \dots\}$$

$$L = \{(5, 6), (2, 3), (87, 88), (0, 1), (1563, 1564), \dots\}$$

$$M = \{(32, 32), (5, 5), (0, 0), (2, 2), (735, 735), \dots\}$$

- Zhodný rozdiel zložiek:

$$(a, b) \in K \leftrightarrow a - b = 2 \leftrightarrow a = b + 2$$

$$(a, b) \in L \leftrightarrow a - b = -1 \leftrightarrow a + 1 = b$$

$$(a, b) \in M \leftrightarrow a - b = 0 \leftrightarrow a = b$$

(Symbol „-“ označuje odčítanie a opačný prvok na  $\mathbb{Z}$ )

#### VI.2 Celé čísla v CL

---

Ako by sme mohli pracovať s celými číslami v CL?

- Každá dvojica prirodzených čísel  $(a, b) \in \mathbb{N}^2$  jednoznačne reprezentuje celé číslo  $a - b$
- Každé celé číslo  $x \in \mathbb{Z}$  je reprezentované nekonečne veľa dvojicami  $\{(a, b) \in \mathbb{N}^2; a - b = x\}$
- Dvojice zakódujeme párovaním

~~$\mathbb{Z}(0)$~~

$$\mathbb{Z}(a, b) \leftarrow \mathbb{N}(a) \wedge \mathbb{N}(b)$$

- Kódovanie *nie je jednoznačné* — jedno celé číslo reprezentuje nekonečne veľa dvojíc
  - ▶ Ktoré dvojice reprezentujú celé čísla 0, 7,  $-3$ ?
- Dohoda: *Kanonickým* reprezentantom je taká dvojica, ktorej aspoň jedna zložka je 0
- Každé celé číslo má *jedného* kanonického reprezentanta
  - ▶ Ktoré dvojice sú kanonickými reprezentantmi 0, 7,  $-3$ ?

### VI.3 Ekvivalencia reprezentantov celých čísel

---

- Dvojice (2, 4) a (53, 55) sú rôzne,  $(2, 4) \neq (53, 55)$ , ale reprezentujú rovnaké celé číslo  $2 - 4 = 53 - 55 = -2$
- Zdefinujme predikát  $\text{Eq}_z(x, y) \equiv (x =_z y)$ , ktorý platí práve vtedy, keď sú  $x$  a  $y$  dvojice reprezentujúce to isté celé číslo

~~$$\neg(x =_z y) \leftarrow x = 0$$~~

~~$$\neg(x =_z y) \leftarrow x = (a, b) \wedge y = 0$$~~

~~$$x =_z y \leftarrow x = (a, b) \wedge y = (c, d) \wedge a + d = c + b$$~~

~~$$\neg(x =_z y) \leftarrow x = (a, b) \wedge y = (c, d) \wedge a + d \neq c + b$$~~

Stručnejšie

$$(a, b) =_z (c, d) \leftarrow a + d = c + b$$

- Zdefinujme funkciu  $C_z(x)$ , ktorá pre ľubovoľného reprezentanta  $x$  vráti kanonického reprezentanta

~~$$C_z(x) = 0 \leftarrow x = 0$$~~

~~$$C_z(x) = (a \div b, 0) \leftarrow x = (a, b) \wedge a \geq b$$~~

~~$$C_z(x) = (0, b \div a) \leftarrow x = (a, b) \wedge a < b$$~~

### 10.3.1. Operácie na celých číslach

#### VI.4 Operácie na reprezentantoch celých čísel

---

- Na reprezentantoch (ľubovoľných, nie len kanonických) môžeme zadefinovať aritmetické operácie
- Zadefinujeme operáciu  $\text{Add}_Z(x, y) \equiv x +_Z y$
- Hodnotou je reprezentant súčtu celých čísel, ktoré sú reprezentované argumentami sčítania:

$$(a, b) +_Z (c, d) = (e, f) \leftrightarrow (a - b) + (c - d) = e - f$$

- Ako určíme hodnoty  $e$  a  $f$ ? Zoskupením kladných a záporných čísel na ľavej strane špecifikačnej rovnosti:

$$\begin{aligned}(a - b) + (c - d) &= a + c - b - d = \\ &= (a + c) - (b + d) = e - f\end{aligned}$$

- Sčítanie teda má explicitnú klauzálnu definíciu

$$(a, b) +_Z (c, d) = (a + c, b + d)$$

#### VI.5 Operácie na reprezentantoch celých čísel

---

- Podobne ako sčítanie môžeme špecifikovať a zadefinovať ďalšie operácie:

$$0_Z \quad -_Z x \quad x -_Z y \quad 1_Z \quad a \cdot_Z x \quad x \times_Z y$$

- Napríklad

$$0_Z = (e, f) \leftrightarrow e - f = 0$$

$$1_Z = (e, f) \leftrightarrow e - f = 1$$

$$-_Z(a, b) = (e, f) \leftrightarrow e - f = -(a - b)$$

$$a \cdot_Z(b, c) = (e, f) \leftrightarrow e - f = a \cdot (b - c)$$

$$(a, b) \times_Z (c, d) = (e, f) \leftrightarrow e - f = (a - b) \times (c - d)$$



## 10.3.2. Rozšírený euklidovský algoritmus

### VI.6 Rozšírený euklidovský algoritmus – Problém

- *Rozšírený euklidovský algoritmus* pre prirodzené čísla  $a$  a  $b$  vypočíta také celé čísla  $x$  a  $y$ , pre ktoré

$$\gcd(a, b) = a \cdot x + b \cdot y$$

- Hľadáme funkciu

$$\begin{aligned} N(a) \wedge N(b) &\rightarrow \exists x \exists y (Z(x) \wedge Z(y) \wedge \overline{\gcd}(a, b) = (x, y)) \\ \overline{\gcd}(a, b) = (x, y) &\leftrightarrow a \cdot z_1 x + z_2 b \cdot z_2 y = \gcd(a, b) \cdot z_1 z_2 \end{aligned}$$

- Pripomenutie: euklidovský algoritmus

$$\begin{aligned} \gcd(a, b) &= b && \leftarrow a = 0 \\ \gcd(a, b) &= \gcd(b \bmod a, a) && \leftarrow a \neq 0 \end{aligned}$$

- Odvodíme klauzálnu definíciu pre  $\overline{\gcd}$

$$\begin{aligned} \overline{\gcd}(a, b) &= (?, ?) \leftarrow a = 0 \\ \overline{\gcd}(a, b) &= (?, ?) \leftarrow a \neq 0 \wedge \overline{\gcd}(b \bmod a, a) = (x', y') \end{aligned}$$

### VI.7 Rozšírený euklidovský algoritmus – Riešenie

- Kontext:

$$\begin{aligned} \gcd(a, b) &= b && \leftarrow a = 0 \\ \gcd(a, b) &= \gcd(b \bmod a, a) && \leftarrow a \neq 0 \end{aligned}$$

$$\begin{aligned} \overline{\gcd}(a, b) &= (?, ?) \leftarrow a = 0 \\ \overline{\gcd}(a, b) &= (?, ?) \leftarrow a \neq 0 \wedge \overline{\gcd}(b \bmod a, a) = (x', y') \end{aligned}$$

- $a = 0 \implies \gcd(a, b) = b$

Pre aké  $x$ ,  $y$  platí (v neformálnej celočíselnej aritmetike)

$$\gcd(a, b) = b = a \cdot x + b \cdot y?$$

- $a \neq 0 \implies \gcd(a, b) = \gcd(b \bmod a, a) \wedge b \bmod a < a$   
Predpokladajme, že  $\overline{\gcd(b \bmod a, a)} = (x', y')$  a platí

$$\gcd(a, b) = \gcd(b \bmod a, a) = (b \bmod a) \cdot x' + a \cdot y'$$

Pre aké  $x, y$  platí (v neformálnej celočíselnej aritmetike)

$$\gcd(a, b) = (b \bmod a) \cdot x' + a \cdot y' = a \cdot x + b \cdot y?$$

- Ako zapíšeme tieto  $x$  a  $y$  v našej celočíselnej aritmetike na dvojiciach?

## VI.8 Rozšírený euklidovský algoritmus — domáca úloha

Prémiová domáca úloha du06:

- Odovzdajte súbor `ex06.c1` s naprogramovaným  $\overline{\gcd}$
- E-mailom na `udp@lists.dai.fmph.uniba.sk`
- Predmet správy: `du06`
- Termín: nedeľa 24. marca 2013 o 23:59:59 SEČ
- Hodnota: 3 body
- Zadanie `ex06.c1` zverejníme v utorok 19. marca 2013 o 12:00

# 11. Opakovanie

## 11.1. Chvostová rekúzia

### VI.9 Opakovanie — chvostová rekúzia

- Funkcia je definovaná *chvostovou rekúziou*, ak sa jej hodnota v rekúziívnom prípade rovná hodnote rekúziívneho volania (bez ďalších úprav)

- Napríklad definícia

$$G(n + 1, a) = G(n, (n + 1) \cdot a)$$
$$G(0, a) = a$$

je chvostovorekurzívna

- Definícia

$$G(n + 1, a) = (n + 1) \cdot G(n, a)$$
$$G(0, a) = a$$

nie je chvostovorekurzívna!

- Chvostovou rekurziou simulujeme cykly (while, for)
- Dôležitá je *inicializácia*

$$F(n) = G(n, 1)$$

## VI.10 Opakovanie – chvostová rekurzia

---

Príklad: efektívny výpočet rekurentného vzťahu

- Zadefinujte efektívnu verziu  $f'(n)$  funkcie  $f(n)$ :

$$f(0) = 0$$

$$f(1) = 1$$

$$f(2) = 2$$

$$f(n + 3) = f(n + 2) + f(n + 1) + f(n)$$

- Využite pomocnú funkciu  $g(n, a, b, c)$  definovanú chvostovou rekurziou s vlastnosťou:

$$g(n, f(i), f(i + 1), f(i + 2)) = f(n + i + 2)$$

## VI.11 Opakovanie — chvostová rekúzia

---

Príklad: inverzná funkcia

- Zadefinujte efektívnu verziu  $f(n)$ , ktorá pre kladné  $n$  nájde najväčšie  $k$ , pre ktoré  $k! \leq n$ , teda

$$n \neq 0 \rightarrow k! \leq n \wedge n < (k + 1)!$$

Pre  $n = 0$  definujeme  $f(0) = 0$ .

- Využite pomocnú funkciu  $g(i, n, f)$  definovanú spätnou chvostovou rekúziou:

$$\begin{aligned} i \leq k \wedge k \leq n \wedge k! \leq n \wedge n < (k + 1)! &\rightarrow \\ g(i, n, f) &= k \\ \neg \exists k (i \leq k \wedge k \leq n \wedge k! \leq n \wedge n < (k + 1)!) &\rightarrow \\ g(i, n, f) &= 0 \end{aligned}$$

## 11.2. Dyadická aritmetika

### VI.12 Opakovanie — dyadický zápis čísel

---

- Dyadický zápis čísel konštruujeme z konštanty 0 konštruktormi

$$S_1(x) = x\mathbf{1} = 2 \cdot x + 1 \quad S_2(x) = x\mathbf{2} = 2 \cdot x + 2$$

- Dyadický zápis čísel testujeme dyadickou diskrimináciou:

$$\begin{array}{ll} \dots \leftarrow \tau = 0 & \dots \leftarrow \tau = 0 \\ \dots \leftarrow \tau = m\mathbf{1} & \dots \leftarrow \tau = S_1(m) \\ \dots \leftarrow \tau = m\mathbf{2} & \dots \leftarrow \tau = S_2(m) \end{array}$$

### VI.13 Opakovanie — dyadická aritmetika

---

Predpis pre implementáciu aritmetickej operácie v dyadickej aritmetike najľahšie odvodíme algebraickými úpravami:

$$0 + y = y$$

$$x1 + 0 = x1$$

$$x1 + y1 = (2 \cdot x + 1) + (2 \cdot y + 1) = 2 \cdot (x + y) + 2 = (x + y)2$$

$$x1 + y2 = (2 \cdot x + 1) + (2 \cdot y + 2) = (2 \cdot x + 2 \cdot y + 2) + 1 =$$

$$= 2 \cdot (x + y + 1) + 1 = ((x + y) + 1)1$$

...

Hľadáme rekurziu s menším počtom dyadických číslíc, použitie už zadaných funkcií a konštruktorov  $S_1$ ,  $S_2$

## 11.3. Symbolické operácie na dyadickom zápise

### VI.14 Opakovanie — dyadický zápis ako reťazec

---

- Dyadický zápis čísla môžeme chápať *symbolicky* ako *reťazec* znakov (postupnosť symbolov) 1 a 2:

**0** prázdny reťazec

**n1** neprázdny reťazec končiaci symbolom 1

**n2** neprázdny reťazec končiaci symbolom 2

- Príklad — naprogramujme symbolické operácie:
  - $\text{Del}_2(x)$
  - $\text{First}(x)$

## VII. prednáška

### Zoznamy.

# Chvostová rekurzia a tupling na zoznamoch

25. marca 2013

## 12. Zoznamy

### VII.1 Rekapitulácia

---

Doteraz sme videli dve kódovania dátových štruktúr:

- Dyadická reprezentácia čísel kóduje konečné postupnosti (reťazce) symbolov 1, 2
- Párovacia funkcia kóduje usporiadané dvojice čísel

Teraz si ukážeme kódovanie konečných postupností

### 12.1. Kódovanie konečných postupností

#### VII.2 Kódovanie konečných postupností

---

- Často potrebujeme spracúvať postupnosti údajov
  - Napríklad spočítať priemerný počet bodov z testu
- Ako zakódujeme postupnosti *ľubovoľných* čísel *ľubovoľnej dĺžky* (vo-pred neznámej)?
  - Napríklad postupnosť 8, 2, 0, 4, 6 alebo postupnosť 1, 3, 5, 7

### VII.3 Kódovanie konečných postupností

---

- Skúsme využiť vnorené párovanie:

$$\begin{aligned}x &= 8, 2, 0, 4, 6 = 8, (2, (0, (4, 6))) = 7798222 \\ y &= 1, 3, 5, 7 = 1, (3, (5, 7)) = 148217\end{aligned}$$

- Dekódujme teraz postupnosť zakódovanú číslom  $x$   
Dĺžku nepoznáme vopred

$$\begin{aligned}x &= a_1, x_1 \rightsquigarrow a_1 = 8 & x_1 &= 2, 0, 4, 6 = 45576 \\ x_1 &= a_2, x_2 \rightsquigarrow a_2 = 2 & x_2 &= 0, 4, 6 = 830 \\ x_2 &= a_3, x_3 \rightsquigarrow a_3 = 0 & x_3 &= 4, 6 = 401 \\ x_3 &= a_4, x_4 \rightsquigarrow a_4 = 4 & x_4 &= 6 \\ x_4 &= a_5, x_5 \rightsquigarrow a_5 = 1 & x_5 &= 1\end{aligned}$$

- Kódovanie je *nejednoznačné*, lebo každé kladné číslo kóduje dvojicu!

$$\begin{aligned}x &= 8, 45576 = 8, 2, 830 = 8, 2, 0, 401 = \\ &= 8, 2, 0, 4, 6 = 8, 2, 0, 4, 1, 1 = 8, 2, 0, 4, 1, 0, 0\end{aligned}$$

### VII.4 Kódovanie konečných postupností

---

- Nie každé číslo kóduje dvojicu:

0

- Nie každé číslo kódujúce dvojicu kóduje trojicu:

$x_1, 0$

$\vdots$

- Nie každé číslo kódujúce  $n$ -ticu kóduje  $(n + 1)$ -ticu:

$x_1, x_2, \dots, x_{n-1}, 0$

## VII.5 Kódovanie konečných postupností

---

- $n$ -prvkovú postupnosť čísel  $x_1, x_2, \dots, x_n$  zakódujeme párovaním ako usporiadanú  $(n + 1)$ -ticu, ktorej posledná zložka je 0:

$$x_1, x_2, \dots, x_n, 0$$

Ukončenie 0 zaručí jednoznačné dekódovanie

$$xs = 8, 2, 0, 4, 6, 0 = 8, (2, (0, (4, (6, 0))))$$

$$ys = 1, 3, 5, 7, 0 = 1, (3, (5, (7, 0)))$$

- Takémuto kódu postupnosti hovoríme **zoznam**
- 0 je *prázdny zoznam*, kóduje prázdnu postupnosť
- Konvencia:

Premenenné obsahujúce zoznamy majú príponu -s

$$xs \quad ys \quad zs \quad \dots$$

(anglické množné číslo, čítame „ixy“, „ypsilon“, „zety“, ...)

## 12.2. Programovanie so zoznamami

### 12.2.1. Zreťazenie

#### VII.6 Programovanie so zoznamami

---

Naprogramujme funkciu  $\text{Conc}(xs, ys)$ , ktorá *zreťazí* zoznamy  $xs$  a  $ys$

- ▶  $\text{Conc}$  vytvorí *nový* zoznam, ktorý obsahuje najprv všetky prvky  $xs$  a potom všetky prvky  $ys$

Ako hľadať riešenie?



## VII.7 Programovanie so zoznamami: Zreťazenie

---

Ako hľadať riešenie?

1. Načrtne si príklad

$$\blacktriangleright \text{Conc}((1, 2, 3, 0), (4, 5, 6, 0)) = 1, 2, 3, 4, 5, 6, 0$$

2. Uvedomíme si, že vstup je zoznam:

- je buď *prázdny*:  $xs = 0$
- alebo má *prvý prvok a zvyšok*:  $xs = x, zs$

⇒ párová diskriminácia

3. Určíme výsledok pre prázdny zoznam

$$\blacktriangleright \text{Conc}(\underbrace{0}_{xs}, \underbrace{(4, 5, 6, 0)}_{ys}) = \underbrace{4, 5, 6, 0}_{ys}$$

4. *Výsledok pre zvyšok zs zoznamu xs*

$$\blacktriangleright \text{Conc}(\underbrace{(2, 3, 0)}_{zs}, \underbrace{(4, 5, 6, 0)}_{ys}) = 2, 3, 4, 5, 6, 0$$

upravíme na výsledok pre *celý zoznam xs*

$$\blacktriangleright \text{Conc}(\underbrace{(\underbrace{1}_x, \underbrace{2, 3, 0}_{zs})}_{xs}, \underbrace{(4, 5, 6, 0)}_{ys}) = \underbrace{1}_x, \underbrace{2, 3, 4, 5, 6, 0}_{\text{Conc}(zs, ys)}$$

## VII.8 Programovanie so zoznamami: Zreťazenie

---

• Výsledná funkcia

$$\text{Conc}(xs, ys) = ys \quad \leftarrow xs = 0$$

$$\text{Conc}(xs, ys) = x, \text{Conc}(zs, ys) \quad \leftarrow xs = x, zs$$

• Párovú diskrimináciu môžeme dosadiť do argumentov a premenovať premennú  $zs$  na  $xs$ :

$$\text{Conc}(0, ys) = ys$$

$$\text{Conc}((x, xs), ys) = x, \text{Conc}(xs, ys)$$

- Výpočet:

$$\begin{aligned}
 & \text{Conc}((1, 2, 3, 0), (4, 5, 6, 0)) \\
 &= 1, \text{Conc}((2, 3, 0), (4, 5, 6, 0)) \\
 &= 1, 2, \text{Conc}((3, 0), (4, 5, 6, 0)) \\
 &= 1, 2, 3, \text{Conc}(0, (4, 5, 6, 0)) \\
 &= 1, 2, 3, 4, 5, 6, 0
 \end{aligned}$$

## VII.9 Zabudované zreťazenie

---

- Zreťazenie je zabudované do CL
- Operátor  $xs ++ ys \equiv xs \oplus ys$

$$xs \oplus ys = \text{Conc}(xs, ys)$$

- Vyššia priorita ako párovanie:

$$\begin{aligned}
 xs \oplus a, ys &\equiv (xs \oplus a), ys && (\dagger\dagger\dagger) \\
 xs \oplus (a, ys) &&& (\text{OK})
 \end{aligned}$$

Chyba v  $(\dagger\dagger\dagger)$ : zreťazujeme zoznam  $xs$  s prvkom  $a$

Zreťazovať môžeme *iba zoznamy!*

## 12.2.2. Porovnanie s Pascalom

### VII.10 Zoznamy v CL a v Pascale

---

- Ako by sme naprogramovali zreťazenie

$$\begin{aligned}
 \text{Conc}(xs, ys) &= ys && \leftarrow xs = 0 \\
 \text{Conc}(xs, ys) &= x, \text{Conc}(zs, ys) && \leftarrow xs = x, zs
 \end{aligned}$$

v Pascale?

- Napríklad takto:

```

function Conc(xs, ys: TVrchol): TVrchol;
var x: Integer; zs: TVrchol;
begin
  if xs <> nil then
    begin
      x := xs.Info;
      zs := xs.Next;
      Result := TVrchol.Create(x, Conc(zs, ys))
    end
  else
    Result := ys;
end;

```

### 12.2.3. Obrátenie

#### VII.11 Programovanie so zoznamami: Obrátenie

Naprogramujme funkciu  $\text{Rev}(xs)$ , ktorá vytvorí nový zoznam s obráteným poradím prvkov ako v  $xs$

Riešenie hľadáme rovnakým postupom ako pri zretžazení:

1. Príklad:

$$\blacktriangleright \text{Rev}(3, 1, 0, 2, 4, 0) = 4, 2, 0, 1, 3, 0$$

2. Uvedomíme si, že vstup je zoznam:

- je buď *prázdny*:  $xs = 0$
- alebo má *prvý prvok a zvyšok*:  $xs = x, zs$

⇒ párová diskriminácia

3. Určíme výsledok pre prázdny zoznam

$$\blacktriangleright \text{Rev}(\underbrace{0}_{xs}) = 0$$

4. Výsledok pre zvyšok  $zs$  zoznamu  $xs$

$$\blacktriangleright \text{Rev}(\underbrace{1, 0, 2, 4, 0}_{zs}) = 4, 2, 0, 1, 0$$

upravíme na výsledok pre *celý zoznam xs*

$$\blacktriangleright \text{Rev}(\underbrace{3}_x, \underbrace{(1, 0, 2, 4, 0)}_{zs}) = \underbrace{(4, 2, 0, 1, 0)}_{\text{Rev}(zs)} \underbrace{?? 3}_x$$

## 12.3. Chvostová rekúzia na zoznamoch

### 12.3.1. Súčet prvkov zoznamu

#### VII.12 Cykly na zoznamoch

---

- Veľa zoznamových operácií sa dá naprogramovať cyklom
- Napríklad súčet prvkov:

```
function Suml(xs: TVrchol): Integer;  
var s, x: Integer; zs: TVrchol;  
begin  
    s := 0;  
    while xs <> nil do begin  
        x := xs.Info; zs := xs.Next;  
        s := s + x;  
        xs := zs  
    end;  
    Result := s  
end;
```

- V deklaratívnom programovaní úlohu cyklov plní chvostová rekúzia

#### VII.13 Chvostová rekúzia na zoznamoch

---

- Chvostová rekúzia simulujúca while cyklus pre súčet prvkov zoznamu:

```
While_suml(xs, s) = s                                ← xs = 0  
While_suml(xs, s) = While_suml(zs, s + x)         ← xs = x, zs
```

- Inicializácia:

$$\text{Suml}(xs) = \text{While\_suml}(xs, 0)$$

- S dosadením do argumentov:

$$\text{While\_suml}(0, s) = s$$

$$\text{While\_suml}((x, xs), s) = \text{While\_suml}(xs, s + x)$$

$$\text{Suml}(xs) = \text{While\_suml}(xs, 0)$$

### 12.3.2. Obrátenie zoznamu

#### VII.14 Neefektivita obrátenia zoznamu zreťazovaním

---

- Pred chvíľou sme obrátenie zoznamu naprogramovali:

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

$$\text{Rev}(x, xs) = \text{Rev}(xs) \oplus (x, 0)$$

- Na zreťazenie  $xs \oplus ys$  treba  $L(xs)$  krokov

- Výpočet Rev:

$$\begin{array}{rcl}
 \text{Rev}(1, 2, 3, 4, 0) & = \text{Rev}(2, 3, 4, 0) \oplus (1, 0) & = (4, 3, 2, 0) \oplus (1, 0) & 3 \\
 & \swarrow & \uparrow & \\
 \text{Rev}(2, 3, 4, 0) & = \text{Rev}(3, 4, 0) \oplus (2, 0) & = (4, 3, 0) \oplus (2, 0) & 2 \\
 & \swarrow & \uparrow & \\
 \text{Rev}(3, 4, 0) & = \text{Rev}(4, 0) \oplus (3, 0) & = (4, 0) \oplus (3, 0) & 1 \\
 & \swarrow & \uparrow & \\
 \text{Rev}(4, 0) & = \text{Rev}(0) \oplus (4, 0) & = (0) \oplus (4, 0) & 0 \\
 & & \uparrow & \\
 & & \text{Rev}(0) = 0 & \\
 \hline
 & & & 3+2+1+0 = 6
 \end{array}$$

- Vo všeobecnosti  $(n^2 + n)/2$  krokov, kde  $n = L(xs) \div 1$

## VII.15 Efektívne obrátenie zoznamu

---

- Zoznam  $xs$  možno obrátiť na  $L(xs)$  krokov:

$xs$	$rs$
1, 2, 3, 4, 0	0
2, 3, 4, 0	1, 0
3, 4, 0	2, 1, 0
4, 0	3, 2, 1, 0
0	4, 3, 2, 1, 0

- Aký program realizuje načrtnutý postup?

## VII.16 Efektívne obrátenie zoznamu v CL

---

- Efektívne obrátenie zoznamu `while` cyklom:

```

rs := nil;
while xs <> nil do begin
  x := xs.Info; zs := xs.Next;
  rs := TVrchol.Create(x, rs);
  xs := zs
end;
Result := rs

```

- Efektívne obrátenie zoznamu chvostovou rekurziou simulujúcou `while` cyklus:

```

While_rev( 0, rs) = rs
While_rev((x, xs), rs) = While_rev(xs, (x, rs))

```

```

Rev(xs) = While_rev(xs, 0)

```

## 12.4. Tupling zoznamových operácií

### 12.4.1. Split = Take, Drop

#### VII.17 Príklad — L, Take, Drop

---

1. Naprogramujme funkciu  $L(xs)$ , ktorej hodnotou je dĺžka (počet prvkov) zoznamu  $xs$
2. Naprogramujme funkciu  $Take(i, xs)$ , ktorej hodnotou je zoznam prvých  $i$  prvkov zoznamu  $xs$

$$\begin{aligned}i \leq L(xs) &\rightarrow \\Take(i, xs) = ys &\leftrightarrow L(ys) = i \wedge \exists zs(xs = ys \oplus zs) \\i > L(xs) &\rightarrow Take(i, xs) = xs\end{aligned}$$

3. Naprogramujme funkciu  $Drop(i, xs)$ , ktorej hodnotou je zvyšok zoznamu  $xs$  po vynechaní prvých  $i$  prvkov

$$\begin{aligned}i \leq L(xs) &\rightarrow \\Drop(i, xs) = zs &\leftrightarrow \exists ys(L(ys) = i \wedge xs = ys \oplus zs) \\i > L(xs) &\rightarrow Drop(i, xs) = 0\end{aligned}$$

#### VII.18 Tupling zoznamových operácií — Split

---

- Operácie

$$\begin{aligned}Take(0, xs) &= 0 \\Take(i + 1, 0) &= 0 \\Take(i + 1, (x, xs)) &= x, Take(i, xs) \\Drop(0, xs) &= xs \\Drop(i + 1, 0) &= 0 \\Drop(i + 1, (x, xs)) &= Drop(i, xs)\end{aligned}$$

majú rovnakú štruktúru diskriminácií a rekurzie

- Ak potrebujeme výsledky oboch  $\implies$  dva prechody zoznamom
- Chceme spojenú jednoprechodovú funkciu — [tupling<sup>↑10.2.3</sup>](#)

$$\text{Split}(i, xs) = \text{Take}(i, xs), \text{Drop}(i, xs)$$

- Príklad s náčrtom rekúzie:

$$\text{Split}(2, (3, 5, 7, 11, 13, 0)) = (3, 5, 0), (7, 11, 13, 0)$$

$$\text{Split}(3, (2, 3, 5, 7, 11, 13, 0)) = (2, 3, 5, 0), (7, 11, 13, 0)$$

## 12.4.2. Zip, Unzip

### VII.19 Tupling zoznamových operácií — Unzip

---

- Zip — spojenie dvoch zoznamov rovnakej dĺžky do *zoznamu dvojíc*

$$\text{Zip}((0, 1, 2, 0), (7, 6, 5, 0)) = (0, 7), (1, 6), (2, 5), 0$$

- Unzip — rozdelenie zoznamu dvojíc na dva zoznamy

$$L(xs) = L(ys) \rightarrow \text{Unzip Zip}(xs, ys) = xs, ys$$

- V podstate tuplingová úloha: spájame funkcie

$$\text{Firsts}((0, 7), (1, 6), (2, 5), 0) = 0, 1, 2, 0$$

$$\text{Seconds}((0, 7), (1, 6), (2, 5), 0) = 7, 6, 5, 0$$

$$\text{Unzip}(xys) = \text{Firsts}(xys), \text{Seconds}(xys)$$

- Príklad s náčrtom rekúzie:

$$\text{Unzip}((1, 6), (2, 5), 0) = (1, 2, 0), (6, 5, 0)$$

$$\text{Unzip}((0, 7), (1, 6), (2, 5), 0) = (0, 1, 2, 0), (7, 6, 5, 0)$$



## IX. prednáška

# Triedenie zoznamov

## Zoznamová reprezentácia množín

8. apríla 2013

### 13. Triedenie zoznamov

#### 13.1. Špecifikácia triedenia

##### IX.1 Špecifikácia triedenia

---

Funkcia  $\text{Sort}(xs) = ys$  *utriedi* zoznam  $xs$ , ak súčasne platí:

1. zoznam  $ys$  obsahuje všetky prvky zoznamu  $xs$  vrátane prípadných opakovaní
  - $ys$  je *permutáciou*  $xs$
  - formálne: platí  $\text{Perm}(xs, ys) \equiv (xs \sim ys)$
2. zoznam  $ys$  je usporiadaný od najmenšieho prvku po najväčší
  - formálne: platí  $\text{Ord}(ys)$

Formalizácia špecifikácie:

$$\text{Sort}(xs) \sim xs$$

$$\text{Ord}(\text{Sort}(xs))$$

##### IX.2 Permutácia zoznamu

---

Ako zistíme, či je zoznam  $xs$  permutáciou zoznamu  $ys$ ?

- Každý prvok sa v  $xs$  nachádza rovnaký početkrát ako v  $ys$
- Potrebujeme počet výskytov prvku  $a$  v zozname  $xs$

► Funkcia  $\text{Count}(a, xs) \equiv \#_a(xs)$

- Potom máme

$$xs \sim ys \leftrightarrow \forall a (\#_a(xs) = \#_a(ys))$$

### IX.3 Permutácia zoznamu – rekurzívny predikát

---

Ako vypočítame  $xs \sim ys$  rekuziou?

1. Rekurzívny výpočet  $\#_a(xs)$

$$\#_a(0) = 0$$

$$\#_a(x, xs) = \#_a(xs) + 1 \leftarrow a = x$$

$$\#_a(x, xs) = \#_a(xs) \quad \leftarrow a \neq x$$

2. Pomocný predikát  $\text{Eq\_counts}$ :

$$\text{Eq\_counts}(zs, xs, ys) \leftrightarrow \forall a (a \in zs \rightarrow \#_a(xs) = \#_a(ys))$$

Naprogramujeme zoznamovou rekuziou na  $zs$

$$\text{Eq\_counts}(0, xs, ys)$$

$$\text{Eq\_counts}((z, zs), xs, ys) \leftarrow ?$$

3. Na otestovanie  $xs \sim ys$  stačí porovnať počty výskytov prvkov, ktoré sa nachádzajú v  $xs$  alebo v  $ys$

$$xs \sim ys \leftarrow \text{Eq\_counts}(xs \oplus ys, xs, ys)$$

### IX.4 Usporiadaný zoznam

---

Ako zistíme, či je zoznam  $xs$  usporiadaný?

- Každý prvok  $a$  v zozname  $xs$  je menší alebo rovnaký ako každý nasledujúci prvok  $b$  v zozname  $xs$

- Vyjadrenie indexovaním:

$$\text{Ord}(xs) \leftrightarrow \forall i \forall j (i < j \wedge j < L(xs) \rightarrow xs[i] \leq xs[j])$$

- Vyjadrenie zreťazením:

$$\text{Ord}(xs) \leftrightarrow \forall a \forall b \forall ws \forall ys \forall zs (xs = ws \oplus (a, ys) \oplus (b, zs) \rightarrow a \leq b)$$

- Stačí porovnávať susediace prvky:

$$\text{Ord}(xs) \leftrightarrow \forall a \forall b \forall ys \forall zs (xs = ys \oplus (a, b, zs) \rightarrow a \leq b)$$

### IX.5 Usporiadaný zoznam — rekurzívny predikát

---

Ako vypočítame  $\text{Ord}(xs)$  rekuziou bez indexovania?

- Prázdny zoznam a jednoprvkové zoznamy sú usporiadané
- V dvoj- a viacprvkovom zozname porovnáваме susedov
  - ▶ Pozor! Musíme porovnať *všetky dvojice* susedov

$$\begin{aligned} \text{Ord}(\quad) & \\ \text{Ord}(a, \quad) & \\ \text{Ord}(a, b, xs) & \leftarrow a \leq b \wedge \text{Ord}(b, xs) \end{aligned}$$

### IX.6 Špecifikácia triedenia — rekapitulácia

---

Funkcia  $\text{Sort}(xs)$  utriedi zoznam  $xs$ , ak spĺňa

$$\begin{aligned} \text{Sort}(xs) & \sim xs \\ \text{Ord}(\text{Sort}(xs)), & \end{aligned}$$

pričom

$$\begin{aligned} xs \sim ys & \leftrightarrow \forall a (\#_a(xs) = \#_a(ys)) \\ \text{Ord}(us) & \leftrightarrow \forall a \forall b \forall xs \forall ys \forall zs (us = xs \oplus (a, ys) \oplus (b, zs) \rightarrow a \leq b) \end{aligned}$$

- Všetky algoritmy triedenia zoznamov spĺňajú rovnakú špecifikáciu
- Líšia sa efektivitou — dĺžkou výpočtu triedenia
- Algoritmy na triedenie polí sa dajú prispôbiť na zoznamy
- Opakované indexovanie je neefektívne, používame postupné prechody zoznamami

## 13.2. Triedenie vsúvaním

- Jednoduchý, ale neefektívny algoritmus
- Zdefinujeme funkciu Isort zoznamovou rekurziou:
  - Aby sme dodržali  $\text{Isort}(0) \sim 0 \wedge \text{Ord}(\text{Isort}(0))$ , zrejme

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

- Predpokladáme, že Isort utriedi  $xs$

$$\text{Isort}(xs) \sim xs \wedge \text{Ord}(\text{Isort}(xs))$$

Hľadáme predpis pre  $\text{Isort}(x, xs)$  tak, aby

$$\text{Isort}(x, xs) \sim (x, xs) \wedge \text{Ord}(\text{Isort}(x, xs))$$

Napríklad

$$\begin{array}{l} \text{Isort}(\underbrace{7, 2, 0, 5, 0}_{xs}) = 0, 2, 5, 7, 0 \\ \qquad \downarrow? \\ \text{Isort}(\underbrace{3, 7, 2, 0, 5, 0}_{xs}) = \underbrace{0, 2, 3, 5, 7, 0}_{?(x, \text{Isort}(xs))} \end{array}$$

Funkcia ? vloží  $x$  do utriedeného zoznamu  $\text{Isort}(xs)$

$$\text{Isort}(x, xs) = \text{Ins}(x, \text{Isort}(xs))$$

## IX.9 Triedenie vsúvaním: vsunutie

---

- Funkcia  $\text{Ins}$  vsunie prvok do utriedeného zoznamu

$$\begin{aligned}\text{Ins}(a, xs) &\sim a, xs \\ \text{Ord}(xs) &\rightarrow \text{Ord } \text{Ins}(a, xs)\end{aligned}$$

- Opäť zoznamovou rekurziou:

- Vsunutím prvku  $a$  do prázdneho zoznamu vzniká jednoprvkový zoznam obsahujúci iba  $a$ :

$$\text{Ins}(a, 0) = a, 0$$

- Predpokladajme, že  $\text{Ins}(a, xs)$  vsunie  $a$  do utriedeného zoznamu  $xs$

Ako vložíme  $a$  do utriedeného zoznamu  $x, xs$ ?

Závisí od vzájomného vzťahu  $a$  a  $x$ :

$$\begin{aligned}\text{Ins}\left(\underbrace{3}_a, \left(\underbrace{5}_x, \underbrace{7, 9, 9, 0}_{xs}\right)\right) &= \underbrace{3}_a, \underbrace{5}_x, \underbrace{7, 9, 9, 0}_{xs} \\ \text{Ins}\left(\underbrace{3}_a, \left(\underbrace{0}_x, \underbrace{2, 5, 7, 0}_{xs}\right)\right) &= \underbrace{0}_x, \underbrace{2, 3, 5, 7, 0}_{\text{Ins}(a, xs)}\end{aligned}$$

## IX.10 Triedenie vsúvaním: výpočet

---

$$\begin{aligned}\text{Isort}(3, 7, 2, 0, 5, 0) &= \text{Ins}(3, \text{Isort}(7, 2, 0, 5, 0)) \\ &= \text{Ins}(3, \text{Ins}(7, \text{Isort}(2, 0, 5, 0))) \\ &\dots \\ &= \text{Ins}(3, \text{Ins}(7, \text{Ins}(2, \text{Ins}(0, \text{Ins}(5, \text{Isort}(0)))))) \\ &= \text{Ins}(3, \text{Ins}(7, \text{Ins}(2, \text{Ins}(0, \text{Ins}(5, 0)))))) \\ &= \text{Ins}(3, \text{Ins}(7, \text{Ins}(2, \text{Ins}(0, (5, 0))))) \\ &= \text{Ins}(3, \text{Ins}(7, \text{Ins}(2, (0, 5, 0)))) \\ &= \text{Ins}(3, \text{Ins}(7, (0, 2, 5, 0))) \\ &= \text{Ins}(3, (0, 2, 5, 7, 0)) \\ &= 0, 2, 3, 5, 7, 0\end{aligned}$$

Dĺžka výpočtu  $\text{Ins}(a, xs)$ : priemerne  $L(xs)/2$  krokov

Dĺžka výpočtu  $\text{Isort}(xs)$ : priemerne  $\sum_{i=0}^{L(xs)} i/2 \approx (L(xs))^2$

### 13.3. Zlúčenie utriedených zoznamov

#### IX.11 Zlúčenie utriedených zoznamov

---

- Ako zlúčime dva utriedené zoznamy do utriedeného zoznamu?

$$\text{Merge}(xs, ys) \sim xs \oplus ys$$

$$\text{Ord}(xs) \wedge \text{Ord}(ys) \rightarrow \text{Ord Merge}(xs, ys)$$

Napríklad

$$\text{Merge}((2, 3, 5, 0), (0, 1, 7, 0)) = 0, 1, 2, 3, 5, 7, 0$$

- Mohli by sme využiť `Ins` a postupne vložiť všetky prvky z prvého zoznamu do druhého:

$$\text{Merge}(0, ys) = ys$$

$$\text{Merge}((x, xs), ys) = \text{Ins}(x, \text{Merge}(xs, ys))$$

Dĺžka výpočtu takéhoto  $\text{Merge}(xs, ys)$ : priemerne  $\sum_{i=0}^{L(xs)} (i + L(ys)) / 2 \approx (L(xs) + L(ys))^2 / 2$  (ako `Isort`)

- Vôbec sme nevyužili, že  $xs$  je utriedený

#### IX.12 Zlúčenie utriedených zoznamov efektívne

---

- Využime, že oba zoznamy sú utriedené a porovnávajme ich prvé prvky:

$xs$	$ys$	$\text{Merge}(xs, ys)$
<b>2</b> , 3, 5, 0	<b>0</b> , 1, 7, 0	0,
<b>2</b> , 3, 5, 0	<b>1</b> , 7, 0	1,
<b>2</b> , 3, 5, 0	<b>7</b> , 0	2,
<b>3</b> , 5, 0	<b>7</b> , 0	3,
<b>5</b> , 0	<b>7</b> , 0	5,
0	7, 0	7, 0

- Dĺžka výpočtu  $\text{Merge}(xs, ys)$  je teraz  $L(xs) + L(ys)$
- Kostra definície:

$\text{Merge}(xs, ys) = ? \leftarrow xs = 0$

$\text{Merge}(xs, ys) = ? \leftarrow xs = x, us \wedge ys = 0$

$\text{Merge}(xs, ys) = ? \leftarrow xs = x, us \wedge ys = y, vs \wedge x \leq y$

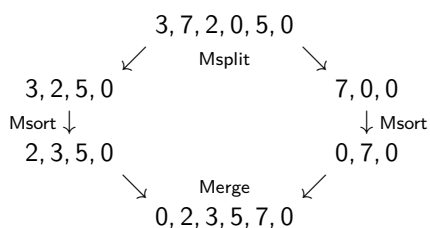
$\text{Merge}(xs, ys) = ? \leftarrow xs = x, us \wedge ys = y, vs \wedge x > y$

## 13.4. Triedenie zlučováním

### IX.13 Triedenie zlučováním (merge sort)

---

- Efektívny algoritmus, metóda rozdeľuj a panuj
- Zoznam rozdelíme na dve približne rovnako dlhé časti
  - ▶ Funkcia  $\text{Msplit}$
- Časti rekurzívne utriedíme
- Utriedené časti efektívne zlúčime
  - ▶ Funkcia  $\text{Merge}$
- Náčrt  $\text{Msort}(3, 7, 2, 0, 5, 0)$ :



- Kľúčová vlastnosť (pozor na triviálne prípady!):

$\text{Msplit}(xs) = ys, zs \rightarrow \text{Msort}(xs) = \text{Merge}(\text{Msort}(ys), \text{Msort}(zs))$

- Pomocnú funkciu Merge sme už rozoberali
- Potrebujeme rozdelenie zoznamu na približne rovnako dlhé časti

$$\exists ys \exists zs (\text{Msplit}(xs) = ys, zs)$$

$$\text{Msplit}(xs) = ys, zs \rightarrow xs \sim ys \oplus zs \wedge (L(ys) = L(zs) \vee L(ys) = L(zs) + 1)$$

- Dá sa urobiť rôzne, napríklad

$$\text{Msplit}(xs) = ys, zs \rightarrow \forall i (xs[2 \cdot i] = ys[i] \wedge xs[2 \cdot i + 1] = zs[i])$$

Čiže

$xs$	3, 7, 2, 0, 5, 0
$ys$	3, 2, 5, 0
$zs$	7, 0, 0

- Kostra definície:

$$\text{Msplit}(0) = ?, ?$$

$$\text{Msplit}(a, 0) = ?, ?$$

$$\text{Msplit}(a, b, xs) = ?, ? \leftarrow \text{Msplit}(xs) = ys, zs$$

## 14. Zoznamová reprezentácia konečných množín

- Konečná množina je kontajner — objekt, ktorý obsahuje iné objekty (prvky)
  - Záleží iba na príslušnosti prvku do množiny ( $\in$ )
  - Nezáleží na počte výskytov prvku
  - Nezáleží na poradí prvkov
- Implementácie rôznymi dátovými štruktúrami
  - Bitové polia, polia, zoznamy, rôzne druhy stromov, ...
  - Rôzne implementácie — rôzna zložitosť operácií



## IX.16 Množiny ako usporiadané zoznamy

---

- Jednoduchá implementácia množín: usporiadané zoznamy bez opakovania
- Vyjadrenie pomocou porovnania susediacich prvkov:

$$\text{Set}(xs) \leftrightarrow \forall a \forall b \forall ys \forall zs (xs = ys \oplus (a, b, zs) \rightarrow a < b)$$

Rekurzívna definícia — podobne ako Ord

- Na definovanie operácií budeme používať *trichotomickú* diskrimináciu

$$\dots \leftarrow s < t$$

$$\dots \leftarrow s = t$$

$$\dots \leftarrow s > t$$

## IX.17 Príslušnosť a extenzionalita

---

- Príslušnosť do množiny

$$\text{Set}(xs) \rightarrow a \in xs \leftrightarrow a \varepsilon xs$$

- Pre neprázdne množiny:

$$\text{Set}(x, xs) \wedge a < x \rightarrow a \notin (x, xs)$$

- Využijeme pri rekurzívnej definícii predikátu  $\text{Member}(a, xs) \equiv (a \in xs)$

~~$$a \notin 0$$~~

~~$$a \notin x, xs \leftarrow a < x$$~~

~~$$a \in x, xs \leftarrow a = x$$~~

~~$$a \in x, xs \leftarrow a > x \wedge a \in xs$$~~

~~$$a \notin x, xs \leftarrow a > x \wedge a \notin xs$$~~

- Platí extenzionalita

$$\text{Set}(xs) \wedge \text{Set}(ys) \rightarrow xs = ys \leftrightarrow \forall a (a \in xs \leftrightarrow a \in ys)$$

- Vlastnosti zoznamovej reprezentácie využijeme pri binárnych operáciách na množinách
- Napríklad zjednotenie  $\text{Union}(xs, ys) \equiv xs \cup ys$

$$\text{Set}(xs) \wedge \text{Set}(ys) \rightarrow \text{Set}(xs \cup ys)$$

$$\text{Set}(xs) \wedge \text{Set}(ys) \rightarrow a \in xs \cup ys \leftrightarrow a \in xs \vee a \in ys$$

- Princíp implementácie je rovnaký ako pri Merge

$$0 \cup ys = ?$$

$$(x, xs) \cup 0 = ?$$

$$(x, xs) \cup (y, ys) = ? \leftarrow x < y$$

$$(x, xs) \cup (y, ys) = ? \leftarrow x = y$$

$$(x, xs) \cup (y, ys) = ? \leftarrow x > y$$

- Ďalšie operácie ( $\subseteq$ ,  $\cap$ ,  $\setminus$ ,  $\Delta$ ) sa implementujú podobne

## X. prednáška

# Kombinatorické predikáty a funkcie na zoznamoch

15. apríla 2013

## 15. Kombinatorické predikáty a funkcie na zoznamoch

### X.1 Predikáty (opakovanie)

---

- *Predikát* je vlastnosť objektu alebo  $n$ -tice objektov
- Príklady predikátov na zoznamoch:

- $xs$  je prázdny zoznam

$$\text{Empty}(xs) \leftrightarrow xs = 0$$

- zoznam  $xs$  je palindróm

$$\text{Palindrome}(xs) \leftrightarrow xs = \text{Rev}(xs)$$

- $a$  je prvkom zoznamu  $xs$

$$a \in xs \leftrightarrow \exists ys \exists zs (xs = ys \oplus (a, zs))$$

- $ys$  je prefixom (začiatočným úsekom) zoznamu  $xs$

$$\text{Prefix}(ys, xs) \leftrightarrow \exists zs (xs = ys \oplus zs)$$

...

- V CL predikáty definujeme
  - podobne ako funkcie
  - *vynechávame* klauzuly pre prípady, že predikát *neplatí*

## 15.1. Súvislé úseky

### 15.1.1. Sufixy

#### X.2 Sufix

---

- Sufix — koncový úsek zoznamu

$$\text{Suffix}(xs, zs) \equiv (xs \sqsupseteq zs) \leftrightarrow \exists ys(xs = ys \oplus zs)$$

- Napríklad

$$(7, 11, 0) \sqsupseteq (7, 11, 0) \quad (1)$$

$$(2, 3, 5, 7, 11, 0) \sqsupseteq (7, 11, 0) \quad (2)$$

$$\neg(0 \sqsupseteq (7, 11, 0))$$

$$\neg((5, 7, 11, 13, 0) \sqsupseteq (7, 11, 0))$$

- Všimnite si:

$$\begin{array}{l} zs = \quad \quad \quad | 7, 11, 0 \\ xs = \underbrace{2, 3, 5,}_{\text{čokoľvek}} | \underbrace{7, 11, 0}_{zs} \end{array}$$

- Ako zadefinujeme rekurzívne?

#### X.3 Sufix — analýza prípadov

---

- $xs \sqsupseteq zs$  analýzou prípadov pre  $xs$ :

- Sufixom prázdneho zoznamu je iba prázdny zoznam:

$$0 \sqsupseteq zs \leftrightarrow zs = 0$$

- Sufixami neprázdneho zoznamu  $(x, xs)$  sú

- \* zoznam  $(x, xs)$  sám a
- \* všetky sufixy zoznamu  $xs$ ,

teda

$$(x, xs) \sqsupset zs \leftrightarrow zs = (x, xs) \vee xs \sqsupset zs$$

- Klauzálna definícia s analýzou prípadov namiesto disjunkcie:

$$0 \sqsupset zs \leftarrow zs = 0$$

$$(x, xs) \sqsupset zs \leftarrow zs = (x, xs)$$

$$(x, xs) \sqsupset zs \leftarrow zs \neq (x, xs) \wedge xs \sqsupset zs$$

#### X.4 Všetky sufixy

---

- Naprogramujme teraz funkciu, ktorej hodnotou pre  $xs$  je zoznam všetkých sufixov  $xs$

$$\text{Suffixes}(2, 3, 5, 7, 11, 0) = (2, 3, 5, 7, 11, 0), (3, 5, 7, 11, 0), \\ (5, 7, 11, 0), (7, 11, 0), (11, 0), 0, 0$$

- Špecifikácia:

$$zs \in \text{Suffixes}(xs) \leftrightarrow xs \sqsupset zs$$

- Všimnite si:

$$a \in (x, xs) \leftrightarrow a = x \vee a \in xs$$

- Využime analýzu prípadov pre  $xs \sqsupset zs$

#### X.5 Všetky sufixy — odvodenie riešenia

---

$$zs \in \text{Suffixes}(xs) \leftrightarrow xs \sqsupset zs$$

$$a \in (x, xs) \leftrightarrow a = x \vee a \in xs$$

- Využime analýzu prípadov pre  $xs \sqsupset zs$ :
  - Sufixom prázdneho zoznamu je iba prázdny zoznam:

$$zs \in \text{Suffixes}(0) \leftrightarrow 0 \sqsupset zs \leftrightarrow zs = 0$$

►  $\text{Suffixes}(0) = 0, 0$

– Sufixami neprázdného zoznamu  $(x, xs)$  sú

- \* zoznam  $(x, xs)$  sám, a
- \* všetky sufixy zoznamu  $xs$ ,

teda

$$\begin{aligned}zs \in \text{Suffixes}(x, xs) &\leftrightarrow (x, xs) \sqsupseteq zs \\&\leftrightarrow zs = (x, xs) \vee xs \sqsupseteq zs \leftrightarrow zs = (x, xs) \vee zs \in \text{Suffixes}(xs) \\&\leftrightarrow zs \in ((x, xs), \text{Suffixes}(xs))\end{aligned}$$

►  $\text{Suffixes}(x, xs) = (x, xs), \text{Suffixes}(xs)$

## 15.1.2. Prefixy

### X.6 Prefix

---

- Prefix — začiatočný úsek zoznamu

$$\text{Prefix}(ys, xs) \equiv (ys \sqsubseteq xs) \leftrightarrow \exists zs(xs = ys \oplus zs)$$

- Napríklad

$$\begin{aligned}0 &\sqsubseteq (2, 3, 5, 7, 11, 0) \\(2, 3, 0) &\sqsubseteq (2, 3, 5, 7, 11, 0) \\ \neg((2, 3, 0) &\sqsubseteq 0) \\ \neg((2, 3, 5, 0) &\sqsubseteq (2, 3, 0)) \\ \neg((1, 2, 3, 0) &\sqsubseteq (2, 3, 5, 7, 11, 0))\end{aligned}$$

- Všimnite si:

$$\begin{array}{l}ys = 2, 3, 5, | 0 \\xs = \underbrace{2, 3, 5, |}_{\text{prvky } ys} \underbrace{7, 11, 0}_{\text{čokoľvek}}\end{array}$$

- Ako zdefinujeme rekurzívne?

## X.7 Prefix – analýza prípadov

---

- $ys \sqsubset xs$  analýzou prípadov pre  $ys$ :

– Prázdny zoznam je prefixom každého zoznamu  $xs$

$$0 \sqsubset xs$$

– Zoznam  $(y, ys)$  je prefixom iba neprázdneho zoznamu  $(x, xs)$ , kde  $x = y$  a  $ys$  je prefixom  $xs$ :

$$(y, ys) \sqsubset xs \leftrightarrow \exists us(xs = (y, us) \wedge ys \sqsubset us)$$

- $ys \sqsubset xs$  analýzou prípadov pre  $xs$ :

– Prefixom prázdneho zoznamu je iba prázdny zoznam:

$$ys \sqsubset 0 \leftrightarrow ys = 0$$

– Prefixami neprázdneho zoznamu  $(x, xs)$  sú

\* prázdny zoznam  $0$ , a

\* každý neprázdny zoznam  $(x, vs)$ , ak  $vs$  je prefixom  $xs$ ,

teda

$$ys \sqsubset (x, xs) \leftrightarrow ys = 0 \vee \exists vs(ys = (x, vs) \wedge vs \sqsubset xs)$$

- Druhá verzia je užitočnejšia

## X.8 Všetky prefixy

---

- Naprogramujme teraz funkciu, ktorej hodnotou pre  $xs$  je zoznam všetkých prefixov  $xs$

$$\text{Prefixes}(2, 3, 5, 7, 11, 0) = 0, (2, 0), (2, 3, 0), (2, 3, 5, 0), \\ (2, 3, 5, 7, 0), (2, 3, 5, 7, 11, 0), 0$$

- Špecifikácia:

$$ys \in \text{Prefixes}(xs) \leftrightarrow ys \sqsubset xs$$

- Využime analýzu prípadov pre  $ys \sqsubset xs$

## X.9 Všetky prefixy — odvodenie riešenia

---

$$ys \in \text{Prefixes}(xs) \leftrightarrow ys \sqsubset xs$$

- Prefixom prázdneho zoznamu je iba prázdny zoznam:

$$ys \in \text{Prefixes}(0) \leftrightarrow ys \sqsubset 0 \leftrightarrow ys = 0$$

- Prefixami neprázdneho zoznamu  $(x, xs)$  sú
  - prázdny zoznam  $0$ , a
  - každý neprázdny zoznam  $(x, vs)$ , ak  $vs$  je prefixom  $xs$ , teda

$$\begin{aligned}ys \in \text{Prefixes}(x, xs) &\leftrightarrow ys \sqsubset (x, xs) \\&\leftrightarrow ys = 0 \vee \exists vs (ys = (x, vs) \wedge vs \sqsubset xs) \\&\leftrightarrow ys = 0 \vee \exists vs (ys = (x, vs) \wedge vs \in \text{Prefixes}(xs)) \\&\leftrightarrow ys = 0 \vee ys \in ???(x, \text{Prefixes}(xs)) \\&\leftrightarrow ys \in 0, ???(x, \text{Prefixes}(xs))\end{aligned}$$

- Potrebujeme pomocnú funkciu ???

## X.10 Všetky prefixy — pomocná funkcia

---

- Špecifikácia pomocnej funkcie pre Prefixes:

$$ys \in \text{Map\_pair}(x, vss) \leftrightarrow \exists vs (ys = (x, vs) \wedge vs \in vss)$$

- Pridá  $x$  na začiatok každého zoznamu z  $vss$

### 15.1.3. Segmenty

## X.11 Segment

---

- Segment — všeobecný súvislý úsek zoznamu

$$\text{Segment}(us, xs) \equiv (us \subset xs) \leftrightarrow \exists ys \exists zs (xs = ys \oplus us \oplus zs)$$



- Například

$$\begin{aligned} (5, 7, 0) &\subset (5, 7, 11, 13, 0) \\ (5, 7, 0) &\subset (2, 3, 5, 7, 11, 13, 0) \\ \neg((5, 11, 0) &\subset (2, 3, 5, 7, 11, 13, 0)) \end{aligned}$$

- Všimnite si:

$$\begin{array}{rcl} us & = & \underbrace{\quad\quad\quad} \mid \underbrace{5, 7, \quad} \mid 0 \\ xs & = & \underbrace{2, 3, \quad} \mid \underbrace{5, 7, \quad} \mid \underbrace{11, 13, 0} \\ & & \text{čokol'vek} \quad \text{prvky } us \quad \text{čokol'vek} \end{array}$$

- Ako zadefinujeme rekurzívne?
- Ktorý z predchádzajúcich predikátov nám pomôže?

## 15.2. Podpostupnosti

### X.12 Podpostupnosť

---

- Zoznam  $ys$  je *vybranou podpostupnosťou* zoznamu  $xs = x_1, x_2, \dots, x_n, 0$ , ak

$$ys = x_{i_1}, x_{i_2}, \dots, x_{i_k}, 0$$

pre nejakú rastúcu postupnosť  $0 \leq i_1 < i_2 < \dots < i_k \leq n$

- Formálnejšie na zoznamoch:

$$\begin{aligned} \text{Subseq}(ys, xs) &\equiv (ys \triangleleft xs) \leftrightarrow \\ &\exists is(\text{Set}(is) \wedge L(is) = L(ys) \wedge \\ &\forall i(i < L(is) \rightarrow is[i] < L(xs) \wedge ys[i] = xs[is[i]])) \end{aligned}$$

- Neformálne:
  - $ys$  vznikne vynechaním niektorých prvkov zo zoznamu  $xs$
  - vzájomné poradie zostávajúcich prvkov sa nezmení
- Napríklad:

$$\begin{array}{r} xs = 3, 5, 2, 11, 17, 7, 13, 0 \\ ys = 5, 2, \quad \quad 7, \quad 0 \end{array}$$

### X.13 Podpostupnosť — analýza prípadov

---

Podpostupnosť môžeme zdefinovať rekurzívne využitím:

- Podpostupnosťou prázdneho zoznamu je iba prázdny zoznam:

$$ys \triangleleft 0 \leftrightarrow ys = 0$$

- Podpostupnosťou zoznamu  $(x, xs)$  je
  - zoznam  $(x, zs)$ , kde  $zs$  je podpostupnosťou zoznamu  $xs$ 
    - ▶  $x$  je zahrnuté v podpostupnosti
  - alebo
  - podpostupnosť zoznamu  $xs$ ,
    - ▶  $x$  je vynechané z podpostupnosti

teda

$$ys \triangleleft (x, xs) \leftrightarrow \exists zs (ys = (x, zs) \wedge zs \triangleleft xs) \vee ys \triangleleft xs$$

### X.14 Všetky podpostupnosti

---

- Generovanie všetkých podpostupností zoznamu — funkcia  $\text{Subseqs}(xs)$

$$ys \in \text{Subseqs}(xs) \leftrightarrow ys \triangleleft xs$$

- Odvodíme opäť využitím analýzy prípadov pre  $ys \triangleleft xs$
- Potrebujeme tiež:

$$a \in xs \oplus ys \leftrightarrow a \in xs \vee a \in ys$$

1. Jedinou podpostupnosťou prázdneho zoznamu je prázdny zoznam

$$ys \in \text{Subseqs}(xs) \leftrightarrow ys \triangleleft 0 \leftrightarrow ys = 0$$

2. Podpostupnosti zoznamu  $(x, xs)$ :

a) všetky  $(x, zs)$ , kde  $zs$  je vybranou podpostupnosťou  $xs$

►  $x$  zahrnieme do podpostupnosti

b) všetky vybrané podpostupnosti  $xs$

►  $x$  vynecháme

$$ys \in \text{Subseqs}(x, xs) \leftrightarrow ys \triangleleft (x, xs)$$

$$\leftrightarrow \exists zs (ys = (x, zs) \wedge zs \triangleleft xs) \vee ys \triangleleft xs$$

$$\leftrightarrow \exists zs (ys = (x, zs) \wedge zs \in \text{Subseqs}(xs)) \vee ys \in \text{Subseqs}(xs)$$

Príklad:

$$\text{Subseqs}('ak') = 'ak', 'a', 'k', '', 0$$

$$\text{Subseqs}('tak') = ('tak', 'ta', 'tk', 't', 0) \quad \text{a)}$$

$$\oplus ('ak', 'a', 'k', '', 0) \quad \text{b)}$$

### 15.3. Permutácie

- Permutáciu zoznamu môžeme zdefinovať aj inak ako

$$xs \sim ys \leftrightarrow \forall a (\#_a(xs) = \#_a(ys))$$

- Potrebujeme pomocný pojem — vsunutie:

$$\text{Insertion}(xs, ys, a) \equiv (xs \approx ys[\downarrow a]) \leftrightarrow$$

$$\exists us \exists vs (xs = us \oplus (a, vs) \wedge ys = us \oplus vs)$$

- Analýza prípadov pre permutácie:

– Permutáciou prázdneho zoznamu je iba prázdny zoznam:

$$0 \sim ys \leftrightarrow ys = 0$$

– Permutáciou zoznamu  $(x, xs)$  je ľubovoľné vsunutie prvku  $x$  do nejakej permutácie zoznamu  $xs$ :

$$(x, xs) \sim ys \leftrightarrow \exists zs(ys \approx zs[\downarrow x] \wedge xs \sim zs)$$

## X.17 Vsunutie

---

• Analýza prípadov pre vsunutie:

–  $ys$  je vsunutím  $a$  do prázdneho zoznamu, ak  $ys$  obsahuje iba  $a$ :

$$ys \approx 0[\downarrow a] \leftrightarrow ys = (a, 0)$$

–  $ys$  je vsunutím  $a$  do zoznamu  $(x, xs)$ , ak  $ys$  vznikne pridaním  $a$  na začiatok  $(x, xs)$  alebo ak  $ys$  začína prvkom  $x$  a pokračuje nejakým vsunutím  $zs$  prvku  $a$  do zoznamu  $xs$ :

$$ys \approx (x, xs)[\downarrow a] \leftrightarrow \\ ys = (a, x, xs) \vee \exists zs(ys = (x, zs) \wedge zs \approx xs[\downarrow a])$$

## 15.4. Ďalšie úlohy

### X.18 Ďalšie kombinatorické úlohy

---

Premyslite si:

- $k$ -prvkové podpostupnosti,
- variácie (ako permutácie, ale niektoré prvky možno vynechať)
- $k$ -prvkové variácie,
- [segmenty](#)<sup>↑15.1.3</sup>

# XI. prednáška

## Binárne stromy

22. apríla 2013

### 16. Binárne stromy

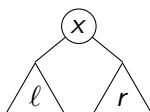
#### 16.1. Kódovanie stromov párovacími konštruktormi

##### XI.1 Binárne stromy

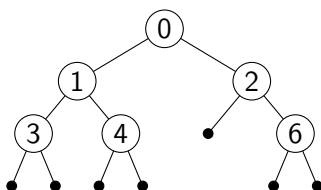
---

Binárny strom — dátová štruktúra definovaná rekurzívne:

- Prázdny strom
- Vrchol s hodnotou  $x$  a dvoma deťmi  $l$  a  $r$ , ktoré sú binárnymi stromami



Napríklad



##### XI.2 Párovacie konštruktory

---

- Binárne stromy by sme v CL mohli zakódovať párovaním:
  - Prázdny strom: 0

– Uzol s hodnotou  $x$  a deťmi  $\ell$  a  $r$ : trojica  $(x, \ell, r)$

- Použijeme elegantnejší spôsob — *párovacie konštruktory*

$$C(x_1, x_2, \dots, x_n) = \underline{k}, x_1, x_2, \dots, x_n$$

$\underline{k}$  je číselná konštanta — *tag*

– Z hodnoty konštruktora sa dajú jednoznačne dekodovať jeho argumenty

$$C(x_1, \dots, x_n) = C(y_1, \dots, y_n) \rightarrow x_1 = y_1 \wedge \dots \wedge x_n = y_n$$

– Žiadna hodnota nemôže byť súčasne výsledkom dvoch konštruktorov s rôznymi tagmi

$$m \neq n \rightarrow (m, x) \neq (n, y)$$

⇒ Konštruktory s rôznymi tagmi sú *diskriminovatelné*

### XI.3 Kódovanie binárnych stromov

---

Binárny strom *zakódujeme* konštruktormi takto:

- Prázdny strom:

$$E \equiv \bullet = 0, 0$$

- Uzol s hodnotou  $x$  a deťmi  $\ell$  a  $r$ :

$$Nd(x, \ell, r) \equiv \frac{x}{\ell|r} = 1, x, \ell, r$$

Binárny strom *dekódujeme* diskrimináciou:

$$\dots \leftarrow t = E$$

$$\dots \leftarrow t = \bullet$$

$$\dots \leftarrow t = Nd(x, \ell, r)$$

$$\dots \leftarrow t = \frac{x}{\ell|r}$$

Po dosadení do argumentov:

$$F(E) = \dots$$

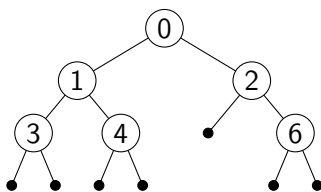
$$F(\bullet) = \dots$$

$$F(Nd(x, \ell, r)) = \dots$$

$$F\left(\frac{x}{\ell|r}\right) = \dots$$

#### XI.4 Kódovanie binárnych stromov — príklad

Napríklad strom



zakódujeme

$$\begin{aligned}
 & \text{Nd}(0, \\
 & \quad \text{Nd}(1, \\
 & \quad \quad \text{Nd}(3, E, E), \\
 & \quad \quad \text{Nd}(4, E, E)), \\
 & \quad \text{Nd}(2, \\
 & \quad \quad E, \\
 & \quad \quad \text{Nd}(6, E, E))) \equiv \begin{array}{c} \hline 0 \\ \hline \begin{array}{c|c} \hline 1 & 2 \\ \hline \begin{array}{c|c} \hline 3 & 4 \\ \hline \bullet & \bullet \end{array} & \begin{array}{c|c} \hline \bullet & 6 \\ \hline \bullet & \bullet \end{array} \\ \hline \end{array} \end{array}
 \end{aligned}$$

#### XI.5 Formát binárnych stromov

- Nie každé číslo kóduje binárny strom
- Číslo, ktoré kóduje binárny strom, sa dá zapísať pomocou konštruktorov E, Nd
- Predikát  $\text{Bt}(t)$  platí iba pre kódy binárnych stromov:

$$\begin{aligned}
 & \text{Bt}(E) \\
 & \text{Bt}(\text{Nd}(x, \ell, r)) \leftarrow N(x) \wedge \text{Bt}(\ell) \wedge \text{Bt}(r) \\
 & \equiv \\
 & \text{Bt}(\bullet) \\
 & \text{Bt}\left(\frac{x}{\ell|r}\right) \leftarrow N(x) \wedge \text{Bt}(\ell) \wedge \text{Bt}(r)
 \end{aligned}$$

- V CL takéto predikáty slúžia aj na formátovanie výsledkov query

## 16.2. Operácie na binárnych stromoch

### XI.6 Rekurgia na binárnych stromoch

---

Operácie na binárnych stromoch definujeme *rekurziou na binárnych stromoch*

$$F(E) = \dots$$

$$F(\text{Nd}(x, \ell, r)) = \dots F(\ell) \dots F(r) \dots$$

Špeciálny prípad všeobecnej (course-of-values) rekurzcie, kódy detí sú menšie čísla ako kód ich rodičovského uzla

$$\ell < \text{Nd}(x, \ell, r) \wedge r < \text{Nd}(x, \ell, r)$$

### XI.7 Dovoľené defaulty

---

- Všímajte si: Klauzálna definícia funkcie  $F(t)$

$$F(E) = \dots$$

$$F(\text{Nd}(x, \ell, r)) = \dots F(\ell) \dots F(r) \dots$$

*neurčuje* hodnotu  $F(t)$  pre všetky čísla  $t$

- $F(t)$  definujeme za predpokladu  $\text{Bt}(t)$ , ostatné prípady prenechávame na default
- Všetky funkcie v tejto prednáške definujeme za predpokladu  $\text{Bt}(t)$
- **Pravidlá pre použitie defaultov:**
  - Ak platia predpoklady špecifikácie, napr.  $\text{Bt}(t)$ , *musíme* hodnotu  $F(t)$  uviesť explicitne, *aj keď je ňou 0!*
  - Ak predpoklady špecifikácie neplatia, napr.  $\neg\text{Bt}(t)$ , môžeme hodnotu  $F(t)$  prenechať na default



### X1.8 Operácie na binárnych stromoch

Veľkosť (počet uzlov) binárneho stromu  $Sz(t) \equiv |t|_b$ :

$$Sz(E) = 0 \qquad \equiv |\bullet|_b = 0$$

$$Sz Nd(x, \ell, r) = 1 + Sz(\ell) + Sz(r) \equiv \left| \frac{x}{\ell | r} \right|_b = 1 + |\ell|_b + |r|_b$$

Predikát „x je prvkom t“  $Inbt(a, t) \equiv a \varepsilon_b t$ :

$$a \varepsilon_b \frac{x}{\ell | r} \leftarrow a = x$$

$$a \varepsilon_b \frac{x}{\ell | r} \leftarrow a \neq x \wedge a \varepsilon_b \ell$$

$$a \varepsilon_b \frac{x}{\ell | r} \leftarrow a \neq x \wedge \neg(a \varepsilon_b \ell) \wedge a \varepsilon_b r$$

## 16.3. Prechody binárnymi stromami

### X1.9 Prechody binárnymi stromami

Uzly binárneho stromu môžeme spracúvať v rôznom poradí

**Preorder:** rodič, ľavé dieťa, pravé dieťa

**Inorder:** ľavé dieťa, rodič, pravé dieťa

**Postorder:** ľavé dieťa, pravé dieťa, rodič

Prechod stromu a uloženie hodnôt z vrcholov do zoznamu v poradí inorder —  $Inorder(t)$  (za predpokladu  $Bt(t)$ ):

$$Inorder \left( \begin{array}{c|c} 0 & \\ \hline \frac{1}{\begin{array}{c|c} 3 & 4 \\ \bullet & \bullet \end{array}} & \frac{2}{\begin{array}{c|c} \bullet & 6 \\ \bullet & \bullet \end{array}} \end{array} \right) = 3, 1, 4, 0, 2, 6, 0$$

↗ ? ↘

$$Inorder \left( \frac{1}{\begin{array}{c|c} 3 & 4 \\ \bullet & \bullet \end{array}} \right) = 3, 1, 4, 0 \quad Inorder \left( \frac{2}{\begin{array}{c|c} \bullet & 6 \\ \bullet & \bullet \end{array}} \right) = 2, 6, 0$$

Ako skonštruujeme inorderový zoznam pre strom zo zoznamov pre jeho podstromy?

### XI.10 Prechod binárnym stromom — výpočet

$$\begin{aligned}
 & \text{Inorder} \left( \frac{0}{\frac{1}{\frac{3}{\bullet} | \frac{4}{\bullet}} | \frac{2}{\bullet | \frac{6}{\bullet}}} \right) \\
 &= \text{Inorder} \left( \frac{1}{\frac{3}{\bullet} | \frac{4}{\bullet}} \right) \oplus \left( 0, \text{Inorder} \left( \frac{2}{\bullet | \frac{6}{\bullet}} \right) \right) \\
 &= \left( \text{Inorder} \left( \frac{3}{\bullet} \right) \oplus \left( 1, \text{Inorder} \left( \frac{4}{\bullet} \right) \right) \right) \\
 &\quad \oplus \left( 0, \text{Inorder}(\bullet) \oplus \left( 2, \text{Inorder} \left( \frac{6}{\bullet} \right) \right) \right) \\
 &= \dots \\
 &= ((0 \oplus (3, 0)) \oplus (1, 0 \oplus (4, 0))) \oplus (0, 0 \oplus (2, 0 \oplus (6, 0))) \\
 &= ((3, 0) \oplus (1, 4, 0)) \oplus (0, 2, 6, 0) \\
 &= (3, 1, 4, 0) \oplus (0, 2, 6, 0) \\
 &= 3, 1, 4, 0, 2, 6, 0
 \end{aligned}$$

### XI.11 Efektívne prechody binárnymi stromami

- Výpočet Inorder je neefektívny — zretazenie prechádza znova zoznam  $\text{Inorder}(\ell)$
- Zefektívnenie — odstránenie zretazenia
- Trik: Pomocná premenná (akumulátor)  $as$  — zoznam, ktorý pripojíme za výpis (pod)stromu inorderom
- Prvky pridávame na začiatok akumulátora
- Požadovaná vlastnosť efektívnej (akumulátorovej) funkcie:

$$\text{Inordera}(t, as) = \text{Inorder}(t) \oplus as \quad (*)$$

- Funkciu  $\text{Inordera}$  naprogramujeme bez zretazenia  
Program odvodíme z požadovanej vlastnosti (\*)

XI.12 Efektívne prechody binárnymi stromami

$$\text{Inordera}(t, as) = \text{Inorder}(t) \oplus as \quad (*)$$

$$\begin{aligned} \text{Inordera}\left(\begin{array}{c|c} 0 & \\ \hline \frac{1}{\begin{array}{c|c} 3 & 4 \\ \bullet & \bullet \end{array}} & \frac{2}{\begin{array}{c|c} 6 & \\ \bullet & \bullet \end{array}} \\ \hline & \frac{98, 99, 0}{as} \end{array}\right) = \frac{3, 1, 4, 0, 2, 6, 98, 99, 0}{as} \\ \uparrow ? \\ \text{Inordera}\left(\begin{array}{c|c} 1 & \\ \hline \frac{3}{\bullet} & \frac{4}{\bullet} \\ \hline & \frac{77, 78, 0}{as_1} \end{array}\right) = \frac{3, 1, 4, 77, 78, 0}{as_1} \\ \text{Inordera}\left(\begin{array}{c|c} 2 & \\ \hline \bullet & \frac{6}{\bullet} \\ \hline & \frac{88, 89, 0}{as_2} \end{array}\right) = \frac{2, 6, 88, 89, 0}{as_2} \end{aligned}$$

XI.13 Efektívne prechody binárnymi stromami

$$\text{Inordera}(t, as) = \text{Inorder}(t) \oplus as \quad (*)$$

$$\begin{aligned} \text{Inordera}(\bullet, as) &\stackrel{(*)}{=} \text{Inorder}(\bullet) \oplus as = 0 \oplus as = as \\ \text{Inordera}\left(\frac{x}{\ell | r}, as\right) &\stackrel{(*)}{=} \text{Inorder}\left(\frac{x}{\ell | r}\right) \oplus as \\ &= (\text{Inorder}(\ell) \oplus (x, \text{Inorder}(r))) \oplus as \\ &= \text{Inorder}(\ell) \oplus ((x, \text{Inorder}(r)) \oplus as) \\ &= \text{Inorder}(\ell) \oplus (x, \underline{\text{Inorder}(r) \oplus as}) \\ &\stackrel{(*)}{=} \text{Inorder}(\ell) \oplus (x, \underline{\text{Inordera}(r, as)}) \\ &\stackrel{(*)}{=} \underline{\text{Inordera}(\ell, (x, \text{Inordera}(r, as)))} \end{aligned}$$

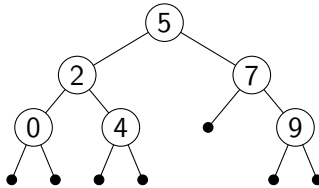
## 16.4. Binárne vyhľadávacie stromy

### XI.14 Binárne vyhľadávacie stromy

---

Binárny vyhľadávací strom

- Binárny strom
- V každom uzle s hodnotou  $x$  platí súčasne:
  - $x$  je väčšie ako všetky hodnoty v ľavom dieťati
  - $x$  je menšie ako všetky hodnoty v pravom dieťati
  - obe deti sú binárne vyhľadávacie stromy



### XI.15 Binárny vyhľadávací strom — príklad

---

Vyjadrenie rekurzívnym predikátom:

$\text{Bst}(\bullet)$

$$\text{Bst}\left(\frac{x}{\ell|r}\right) \leftarrow x \succ \ell \wedge x \prec r \wedge \text{Bst}(\ell) \wedge \text{Bst}(r)$$

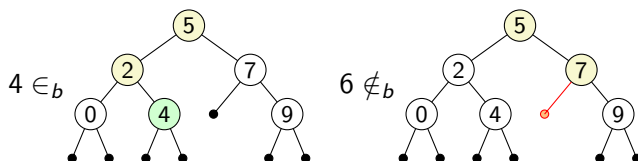
Cvičenie: rekurzívne definície predikátov

$$x \succ t \leftrightarrow \forall y (y \in_b t \rightarrow x > y)$$

$$x \prec t \leftrightarrow \forall y (y \in_b t \rightarrow x < y)$$

### XI.16 Využitie vyhľadávacej vlastnosti

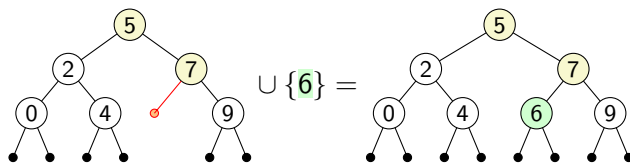
- Binárne vyhľadávacie stromy reprezentujú množiny
  - ▶ Podobne ako ostro usporiadané zoznamy v 8. prednáške
- Využitie vyhľadávacej vlastnosti pri predikáte „byť prvkom“  $a \in_b t$  pre  $t = \frac{x}{\ell|r}$ :
  - Ak  $a < x$ , prehľadávame iba  $\ell$ , lebo  $a$  nemôže byť v  $r$
  - Ak  $a > x$ , prehľadávame iba  $r$ , lebo  $a$  nemôže byť v  $\ell$ $\Rightarrow$  Prehľadávame *iba jednu cestu* stromom



### XI.17 Vkladanie do vyhľadávacieho stromu

Princíp vkladania hodnoty do vyhľadávacieho stromu:

- Pokúsime sa nájsť vkladajúcu hodnotu
  - a) Hodnota sa v strome nenachádza (na jej mieste je **prázdny strom**)  
 $\Rightarrow$  vyrobíme **nový uzol**

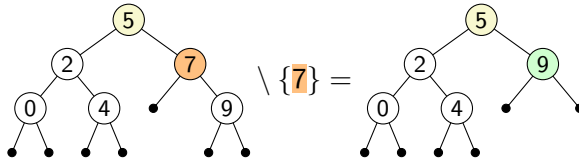


- b) Hodnota sa v strome nachádza  $\Rightarrow$  strom sa nemení
- Cestou naspäť strom **zrekonštruujeme**

### XI.18 Zmazanie z vyhľadávacieho stromu I

Zmazanie hodnoty je komplikovanejšie ako vloženie

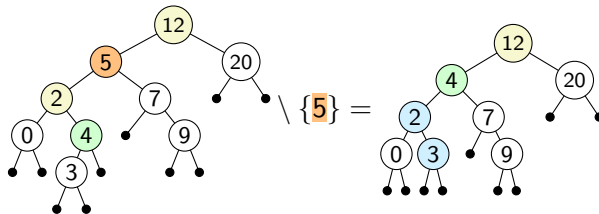
- Pokúsime sa nájsť zmazávanú hodnotu:
  - a) Hodnotu nenájde  $\Rightarrow$  strom sa nezmení
  - b) Hodnotu **nájde** a niektoré dieťa je prázdne  $\Rightarrow$  nahradíme uzol **druhým dieťaťom**



### XI.19 Zmazanie z vyhľadávacieho stromu II

- Pokúsime sa nájsť zmazávanú hodnotu:
  - ...
  - c) Hodnotu **nájde** a obe deti sú neprázdne:

1. Nájde náhradu za zmazávanú hodnotu:  
napríklad **maximum ľavého podstromu**
2. Odstránime náhradu **z pôvodného miesta**
3. Použijeme náhradu namiesto zmazávanej hodnoty



Kroky 1 a 2 — pomocná tuplingová funkcia (rozoberieme podrobnejšie)

- Cestou naspäť strom **zrekonštruujeme**

## XI.20 Maximum vyhľadávacieho stromu

---

- Nájdenie a zmazanie maxima vyhľadávacieho stromu:

$$\text{Bst}(t) \wedge t \neq \bullet \rightarrow \text{Maxt}(t) \varepsilon_b t$$

$$\text{Bst}(t) \wedge t \neq \bullet \wedge x \varepsilon_b t \rightarrow x \leq \text{Maxt}(t)$$

$$\text{Bst}(t) \wedge t \neq \bullet \rightarrow \text{Bst Delmaxt}(t)$$

$$\text{Bst}(t) \wedge t \neq \bullet \rightarrow x \varepsilon_b \text{Delmaxt}(t) \leftrightarrow x \varepsilon_b t \wedge x \neq \text{Maxt}(t)$$

- Programy hľadajú maximum v najpravejšom uzle:

$$\text{Maxt}\left(\frac{x}{\ell|r}\right) = x \quad \leftarrow r = \bullet$$

$$\text{Maxt}\left(\frac{x}{\ell|r}\right) = \text{Maxt}(r) \quad \leftarrow r \neq \bullet$$

$$\text{Delmaxt}\left(\frac{x}{\ell|r}\right) = \ell \quad \leftarrow r = \bullet$$

$$\text{Delmaxt}\left(\frac{x}{\ell|r}\right) = \frac{x}{\ell|\text{Delmaxt}(r)} \quad \leftarrow r \neq \bullet$$

- Rovnaké diskriminácie a argumenty rekurzie  $\Rightarrow$  kandidáti na *tupling*

## XI.21 Maximum vyhľadávacieho stromu – tupling

---

- Spojená funkcia — extrakcia maxima:

$$\text{Bst}(t) \wedge t \neq \bullet \rightarrow \text{Extract\_max}(t) = \text{Maxt}(t), \text{Delmaxt}(t)$$

$$\text{Bst}(t) \wedge t \neq \bullet \rightarrow$$

$$\exists m \exists t_1 (\text{Extract\_max}(t) = m, t_1 \wedge m \varepsilon_b t \wedge \text{Bst}(t_1)$$

$$\wedge \forall x (x \varepsilon_b t \rightarrow x \leq m)$$

$$\wedge \forall x (x \varepsilon_b t_1 \leftrightarrow x \varepsilon_b t \wedge x \neq m))$$

- Definícia:

$$\text{Extract\_max}\left(\frac{x}{\ell|r}\right) = ?, ? \leftarrow r = \bullet$$

$$\text{Extract\_max}\left(\frac{x}{\ell|r}\right) = ?, ? \leftarrow r \neq \bullet \wedge \text{Extract\_max}(?) = ?, ?$$

## XII. prednáška

# Číslovanie binárnych stromov

## Všeobecné stromy

29. apríla 2013

### 16. Binárne stromy

#### 16.5. Číslovanie vrcholov v binárnych stromoch

##### XII.1 Úplné binárne stromy

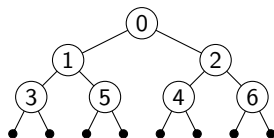
---

- Hĺbka stromu — dĺžka najdlhšej cesty koreň–list:

$$d(\bullet) = 0$$

$$d\left(\frac{x}{\ell|r}\right) = 1 + \max(d(\ell), d(r))$$

- Úplný binárny strom:
  - na každej úrovni okrem poslednej: všetky uzly neprázdné
  - na poslednej úrovni sú iba prázdne stromy



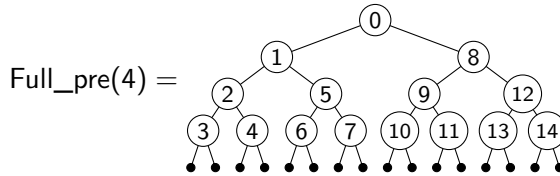
- Vzťah veľkosti a hĺbky úplného stromu:

$$|t|_b + 1 = 2^{d(t)}$$



## XII.2 Číslovanie vrcholov v úplných stromoch I

- Číslovanie uzlov úplného stromu číslami  $0, 1, 2, \dots$  v poradí preorder:



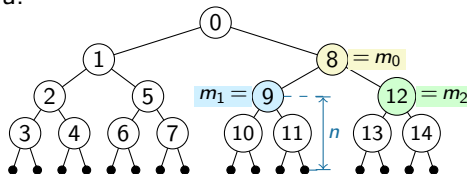
- Na nájdenie rekurzívneho riešenia musíme problém *zovšeobecniť*
- Zovšeobecnenie umožní číslovať podstromy

## XII.3 Číslovanie vrcholov v úplných stromoch II

- Zovšeobecnenie — Full\_pre<sub>1</sub>( $n, m$ ):

Číslovanie uzlov úplného stromu hĺbky  $n$  číslami  $m, m + 1, m + 2, \dots$  v poradí preorder

- Schéma riešenia:

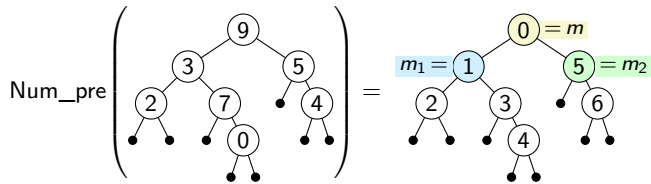


Číslovanie v poradí *preorder*:

1. očísľujeme koreň podstromu  $m_0 = m$
  2. očísľujeme ľavý podstrom od  $m_1 = m_0 + 1$
  3. očísľujeme pravý podstrom od  $m_2 = m_1 + s$
- Aké je  $s$  pre úplný strom hĺbky  $n$ ?
  - Ako sa číslovanie zmení pre inorder, postorder?

## XII.4 Číslovanie vrcholov v ľubovoľných stromoch

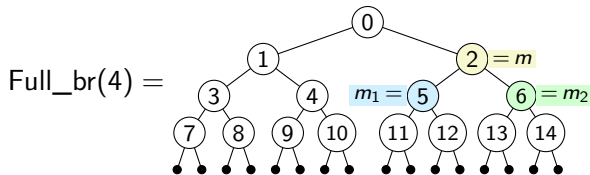
- Číslovanie uzlov ľubovoľného stromu číslami  $0, 1, 2, \dots$  v poradí pre-order:



- Rovnaký postup ako pri úplných stromoch:
  - Zovšeobecníme na číslovanie od  $m$  —  $\text{Num\_pre}_1(t, m)$
  - Aký je rozdiel medzi číslom ľavého a pravého dieťaťa?
  - Rozdiel vieme počítať aj bez pomocnej funkcie — tupling

## XII.5 Číslovanie úplných stromov do šírky

- Číslovanie uzlov úplného stromu číslami  $0, 1, 2, \dots$  v poradí prechodu do šírky:



- Problém zovšeobecníme pre podstrom s koreňom s číslom  $m$
- Aký je vzťah medzi číslom uzla  $m$  a číslami jeho detí  $m_1$  a  $m_2$ ?

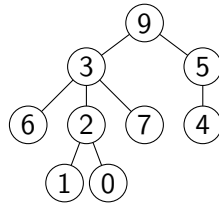
# 17. Všeobecné stromy

## 17.1. Kódovanie

### XII.6 Všeobecné stromy — kódovanie

---

Všeobecný strom — ľubovoľne veľa detí každého uzla:



Ako zakódujeme takýto strom v CL?

- Počet detí je premenlivý a neobmedzený
- Uložíme ich do zoznamu
- Stačí nám jeden párovací konštruktor:

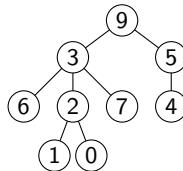
$$\text{Gnd}(x, ts) \equiv \frac{x}{ts} = 1, x, ts$$

$x$  — hodnota uložená v uzle stromu,  $ts$  — zoznam detí

- Ako zakódujeme strom na obrázku?

### XII.7 Všeobecné stromy — kódovanie

---



- Ako zakódujeme strom na obrázku?

$$\begin{aligned}
T_1 &= \text{Gnd}(9, (\text{Gnd}(3, (\text{Gnd}(6, 0), \\
&\quad \text{Gnd}(2, 0), \\
&\quad \text{Gnd}(7, (\text{Gnd}(1, 0), \text{Gnd}(0, 0), 0)), \\
&\quad 0)), \\
&\quad \text{Gnd}(5, (\text{Gnd}(4, 0), 0), \\
&\quad 0)) \\
&\quad 9 \\
&\equiv \frac{3}{\frac{6}{0}, \frac{2}{0}, \frac{7}{\frac{1}{0}, \frac{0}{0}}, 0}, \frac{5}{\frac{4}{0}, 0}, 0
\end{aligned}$$

## XII.8 Všeobecné stromy — dekódovanie

---

- Všeobecný strom dekódujeme diskrimináciou

$$\dots \leftarrow t = \frac{x}{ts} \quad \equiv \quad \dots \leftarrow t = \text{Gnd}(x, ts)$$

- Zoznam všeobecných stromov dekódujeme kombináciou zoznamovej diskriminácie a diskriminácie na všeobecných stromoch:

$$\begin{aligned}
\dots \leftarrow ts = 0 &\quad \equiv \quad \dots \leftarrow ts = 0 \\
\dots \leftarrow ts = \frac{x}{ts_1}, ts_2 &\quad \equiv \quad \dots \leftarrow ts = \text{Gnd}(x, ts_1), ts_2
\end{aligned}$$

- Obe diskriminácie môžeme dosadiť do argumentov:

$$\begin{aligned}
F\left(\frac{x}{ts}\right) = \dots &\quad \equiv \quad F \text{Gnd}(x, ts) = \dots \\
F(0) = \dots &\quad \equiv \quad F(0) = \dots \\
F\left(\frac{x}{ts_1}, ts_2\right) = \dots &\quad \equiv \quad F(\text{Gnd}(x, ts_1), ts_2) = \dots
\end{aligned}$$

## XII.9 Všeobecné stromy

---

Aké čísla kódujú všeobecné stromy?

- Rozpoznáme ich predikátom Gt:

$$\text{Gt}\left(\frac{x}{ts}\right) \leftarrow N(x) \wedge ???(ts)$$

- $ts$  je zoznam všeobecných stromov (*les*)

$$\text{Lgt}(0)$$

$$\text{Lgt}(t, ts) \leftarrow \text{Gt}(t) \wedge \text{Lgt}(ts)$$

$$\text{Gt}\left(\frac{x}{ts}\right) \leftarrow N(x) \wedge \text{Lgt}(ts)$$

- Predikáty  $\text{Gt}$  a  $\text{Lgt}$  sú *vzájomne rekurzívne*
- Problém: CL vzájomnú rekurziu nepodporuje
- Riešenie: dosadíme definíciu  $\text{Gt}$  do definície  $\text{Lgt}$

$$\text{Lgt}(0)$$

$$\text{Lgt}\left(\frac{x}{ts_1}, ts_2\right) \leftarrow N(x) \wedge \text{Lgt}(ts_1) \wedge \text{Lgt}(ts_2)$$

$$\text{Gt}\left(\frac{x}{ts}\right) \leftarrow N(x) \wedge \text{Lgt}(ts)$$

- Tento postup — rekurziu na zoznamoch všeobecných stromov využijeme aj v ďalších úlohách

## XII.10 Odbočka o zovšeobecňovaní

Všimnite si:

- Rozpoznať kód zoznamu stromov je *všeobecnejší problém* ako rozpoznať kód stromu
- Zovšeobecnenie často vedie k (efektívnemu) riešeniu
- Už sme videli príklady, keď zovšeobecnenie viedlo k efektívnemu riešeniu:
  - funkcia  $\text{While\_fib}(n, a, b)$  je všeobecnejšia ako  $\text{Fib}(n + 1) = \text{While\_fib}(n, 0, 1)$
  - $\text{While\_rev}(xs, rs) = \text{Rev}(xs) \oplus rs$  je všeobecnejšia  $\text{Rev}(xs) = \text{While\_rev}(xs, 0)$
  - $\text{Preordera}(t, as) = \text{Preorder}(t) \oplus as$  je všeobecnejšia ako  $\text{Preorder}(t) = \text{Preordera}(t, 0)$

alebo k vôbec nejakému riešeniu:

- $\text{Full\_pre}_1(n, m)$  je všeobecnejšia, ako  $\text{Full\_pre}(n) = \text{Full\_pre}_1(n, 0)$
- $\text{Num\_pre}_1(t, m)$  je všeobecnejšia, ako  $\text{Num\_pre}(t) = \text{Num\_pre}_1(t, 0)$

## 17.2. Programovanie so všeobecnými stromami

### XII.11 Programovanie so všeobecnými stromami

- Pri programovaní funkcií na všeobecných stromoch využívame pomocné funkcie na zoznamoch všeobecných stromov (podobne ako pri  $\text{Gt}(t)$ ):

$$f_L(0) = \dots$$

$$f_L\left(\frac{x}{ts_1}, ts_2\right) = \dots f_L(ts_1) \dots f_L(ts_2) \dots$$

$$F\left(\frac{x}{ts}\right) = \dots f_L(ts) \dots$$

- Príklad: Zdefinujme funkciu  $|t|_g$  počítajúcu veľkosť všeobecného stromu  $t$

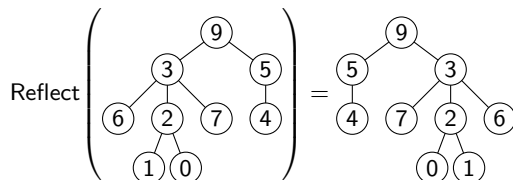
$$|0|_L = 0$$

$$\left|\frac{x}{ts_1}, ts_2\right|_L = 1 + |ts_1|_L + |ts_2|_L$$

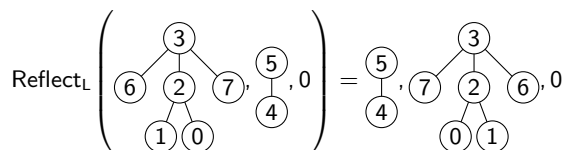
$$\left|\frac{x}{ts}\right|_g = 1 + |ts|_L$$

### XII.12 Zrkadlový obraz

- Zdefinujme zrkadlový obraz stromu  $t$  — funkciu  $\text{Reflect}(t)$



- Zovšeobecníme na zoznamy stromov  $\text{Reflect}_L(ts)$



## XII.13 Zrkadlový obraz

---

- Skombinujeme obrátenie zoznamu a zrkadlový obraz binárneho stromu

$$\begin{aligned} \text{Rev}(0) &= 0 \\ \text{Rev}(x, xs) &= \text{Rev}(xs) \oplus (x, 0) \quad \times \quad \text{Reflect}(\bullet) = \bullet \\ &= \text{Reflect} \left( \frac{x}{\ell \mid r} \right) = \frac{x}{\text{Reflect}(r) \mid \text{Reflect}(\ell)} \end{aligned}$$

- Výsledná definícia

$$\begin{aligned} \text{Reflect}_L(0) &= 0 \\ \text{Reflect}_L \left( \frac{x}{ts_1}, ts_2 \right) &= \text{Reflect}_L(ts_2) \oplus \left( \frac{x}{\text{Reflect}_L(ts_1)}, 0 \right) \end{aligned}$$

- Podobne ako  $\text{Rev}$ , aj  $\text{Reflect}_L$  je neefektívna
- Efektívna verzia s akumulátorom a vlastnosťou

$$\text{Reflecta}_L(ts, rts) = \text{Reflect}_L(ts) \oplus rts$$

Príklad:

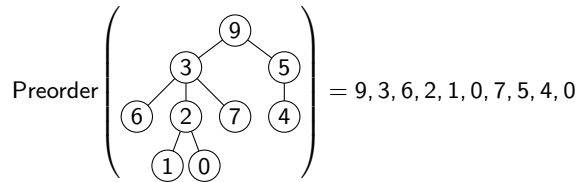
$$\text{Reflecta}_L \left( \left( \begin{array}{c} \textcircled{2} \\ \textcircled{1} \quad \textcircled{0} \end{array}, \textcircled{7}, 0 \right), \left( \textcircled{6}, 0 \right) \right) = \textcircled{7}, \begin{array}{c} \textcircled{2} \\ \textcircled{0} \quad \textcircled{1} \end{array}, \textcircled{6}, 0$$

## 17.3. Prechody všeobecnými stromami

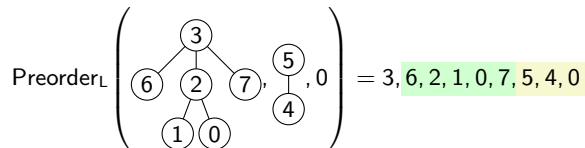
### XII.14 Prechody všeobecnými stromami

---

- Podobne ako pri binárnych stromoch
- Význam má preorder a postorder, ale nie inorder
- Napríklad:



- Opäť zovšeobecníme na zoznamy stromov — funkcia  $\text{Preorder}_L(ts)$



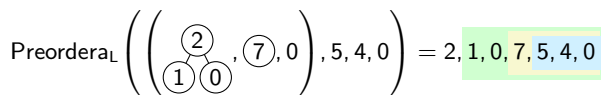
### XII.15 Prechody všeobecnými stromami

---

- Podobne ako na binárnych stromoch, Preorder na všeobecných stromoch používajúci zretazenie je neefektívny
- Efektívna verzia s akumulátorom a vlastnosťou

$$\text{Preorder}_L(ts, as) = \text{Preorder}_L(ts) \oplus as$$

- Príklad/návod:





## XIII. prednáška

# Postfixový stroj a numerické výrazy

6. mája 2013

## 18. Postfixový stroj a numerické výrazy

### 18.1. Postfixový stroj

#### XIII.1 Postfixový stroj

---

- Jednoduchý výpočtový model, virtuálny stroj
- Pamäť: zásobník čísel

5
3
7
2

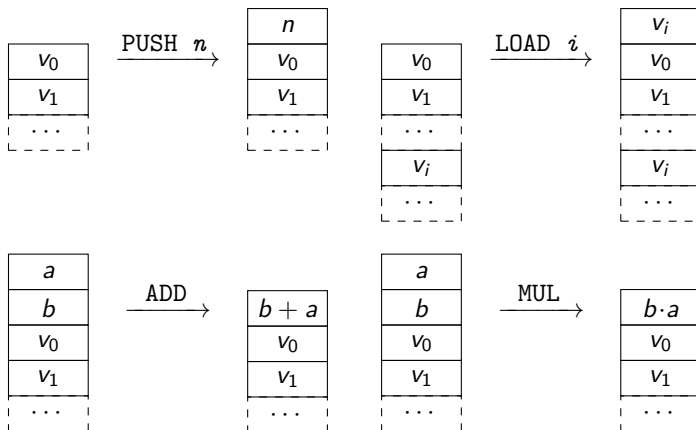
- Program: postupnosť inštrukcií

```
LOAD 2  
PUSH 4  
MUL  
LOAD 2  
LOAD 2  
MUL  
ADD
```

Inštrukcie modifikujú zásobník

Pridávajú/odoberajú čísla na vrch/z vrchu zásobníka

### XIII.2 Inštrukcie postfixového stroja

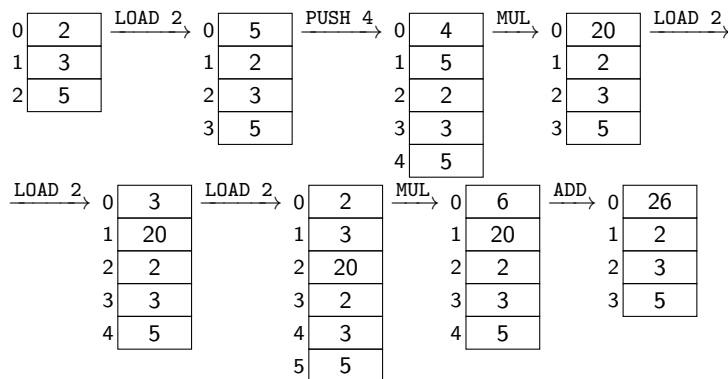


### XIII.3 Príklad výpočtu programu

Počítajme program

LOAD 2, PUSH 4, MUL, LOAD 2, LOAD 2, MUL, ADD

so zásobníkom, v ktorom sú čísla 2, 3, 5:

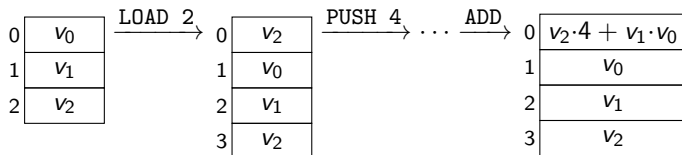


### XIII.4 Príklad výpočtu programu — všeobecnejšie

Počítajme program

LOAD 2, PUSH 4, MUL, LOAD 2, LOAD 2, MUL, ADD

so zásobníkom, v ktorom sú čísla  $v_0, v_1, v_2$ :



### XIII.5 Kódovanie postfixových strojov

Postfixové stroje ľahko zakódujeme v CL:

- Zásobník: zoznam čísel; vrch zásobníka: prvý prvok
- Inštrukcie: párové konštruktory (podobne ako E a Nd)

$$\begin{aligned} \text{PUSH}(n) &\equiv C_i(n) = 0, n & \text{ADD} &\equiv A_i = 2, 0 \\ \text{LOAD}(i) &\equiv V_i(i) = 1, i & \text{MUL} &\equiv M_i = 3, 0 \end{aligned}$$

Predikát  $\text{Instr}(x)$  platí, ak  $x$  kóduje inštrukciu:

$$\begin{aligned} \text{Instr PUSH}(n) &\leftarrow N(n) \\ \text{Instr LOAD}(i) &\leftarrow N(i) \\ \text{Instr(ADD)} & \\ \text{Instr(MUL)} & \end{aligned}$$

- Program: zoznam inštrukcií

$$\begin{aligned} \text{Program}(0) & \\ \text{Program}(i, is) &\leftarrow \text{Instr}(i) \wedge \text{Program}(is) \end{aligned}$$

### XIII.6 Simulácia postfixových strojov

Zakódovaný postfixový stroj môžeme *simulovať*:

- Funkcia  $\text{Run}(is, vs)$
- Simuluje beh postfixového stroja s programom  $is$  a zásobníkom  $vs$

- Vrátí vrch zásobníka na konci programu
- Klauzálna definícia:

$$\begin{aligned} \text{Run}(0, (v, vs)) &= v \\ \text{Run}(\text{PUSH}(n), is, vs) &= \text{Run}(is, (n, vs)) \\ \text{Run}(\text{LOAD}(i), is, vs) &= \dots \\ \text{Run}(\text{ADD}, is, (a, b, vs)) &= \dots \\ \text{Run}(\text{MUL}, is, (a, b, vs)) &= \dots \end{aligned}$$

## 18.2. Numerické výrazy s premennými

### XIII.7 Numerické výrazy s premennými

---

Numerický výraz s premennými je

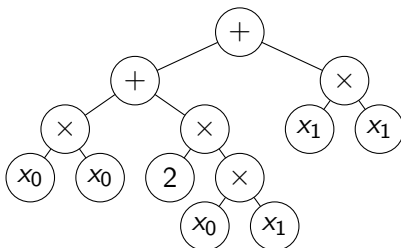
- číselná konštanta  $k \in \mathbb{N}$ 
  - ▶ 3, 0, 777, ...
- premenná  $x_i$  s indexom  $i \in \mathbb{N}$ 
  - ▶  $x_2, x_{17}, x_0, \dots$
- súčtový výraz  $(t_1 + t_2)$ , kde  $t_1$  a  $t_2$  sú numerické výrazy s premennými
  - ▶  $(x_1 + 5), (x_5 + x_0), (x_3 + (x_0 + x_2)), \dots$
- súčinnový výraz  $(t_1 \times t_2)$ , kde  $t_1$  a  $t_2$  sú numerické výrazy s premennými
  - ▶  $(2 \times x_0), (x_4 \times x_1), (((x_1 \times x_1) + (2 \times x_1)) + 1), \dots$

### XIII.8 Numerické výrazy ako stromy

---

- Numerické výrazy majú stromovú štruktúru

- Napríklad  $((x_0 \times x_0) + (2 \times (x_0 \times x_1))) + (x_1 \times x_1)$



### XIII.9 Kódovanie numerických výrazov

Numerické výrazy môžeme v CL zakódovať podobne ako binárne stromy:

- Párovacie konštruktory (podobne ako E a Nd)

$$n^\bullet \equiv \text{Ct}(n) = 0, n$$

$$x_i^\bullet \equiv \text{Vt}(i) = 1, i$$

$$t_1 +^\bullet t_2 \equiv \text{At}(t_1, t_2) = 2, t_1, t_2$$

$$t_1 \times^\bullet t_2 \equiv \text{Mt}(t_1, t_2) = 3, t_1, t_2$$

- Predikát  $\text{Term}(t)$  platí, ak  $t$  kóduje numerický výraz:

$$\text{Term}(n^\bullet) \leftarrow \text{N}(n)$$

$$\text{Term}(x_i^\bullet) \leftarrow \text{N}(i)$$

$$\text{Term}(t_1 +^\bullet t_2) \leftarrow \text{Term}(t_1) \wedge \text{Term}(t_2)$$

$$\text{Term}(t_1 \times^\bullet t_2) \leftarrow \text{Term}(t_1) \wedge \text{Term}(t_2)$$

### XIII.10 Kódovanie numerických výrazov – príklad

- Výraz  $((x_0 \times x_0) + (2 \times (x_0 \times x_1))) + (x_1 \times x_1)$  zakódujeme ako

$$((x_0^\bullet \times^\bullet x_0^\bullet) +^\bullet (2^\bullet \times^\bullet (x_0^\bullet \times^\bullet x_1^\bullet))) +^\bullet (x_1^\bullet \times^\bullet x_1^\bullet)$$

$$\equiv \text{At}(\text{At}(\text{Mt}(\text{Vt}(0), \text{Vt}(0)),$$

$$\text{Mt}(\text{Ct}(2),$$

$$\text{Mt}(\text{Vt}(0), \text{Vt}(1))),$$

$$\text{Mt}(\text{Vt}(1), \text{Vt}(1)))$$

### XIII.11 Vyhodnocovanie numerických výrazov

Ohodnotenie premenných:

- zoznam  $vs = v_0, v_1, v_2, \dots$
- priraduje premennej  $x_i$  hodnotu  $v_i$

Hodnota (*denotácia*) výrazu pre dané ohodnotenie premenných — funkcia  $\text{Den}(t, vs) \equiv \llbracket t \rrbracket^{vs}$ :

- Napríklad:

$$t = ((x_0 \times x_0) + (2 \times (x_0 \times x_1))) + (x_1 \times x_1)$$

$$\llbracket t \rrbracket^{3,5,7,0} = ((3 \cdot 3) + (2 \cdot (3 \cdot 5))) + (5 \cdot 5)$$

- Všeobecne:

$$\llbracket n \rrbracket^{vs} = n$$

$$\llbracket x_i \rrbracket^{vs} = \dots$$

$$\llbracket t_1 + t_2 \rrbracket^{vs} = \dots$$

$$\llbracket t_1 \times t_2 \rrbracket^{vs} = \dots$$

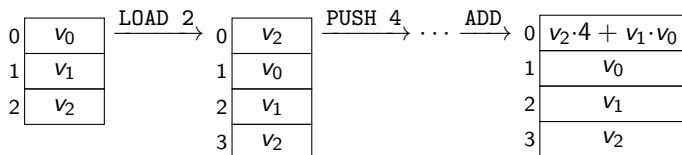
## 18.3. Kompilácia

### XIII.12 Programy a výrazy — príklad

Program

LOAD 2, PUSH 4, MUL, LOAD 2, LOAD 2, MUL, ADD

so zásobníkom, v ktorom sú čísla  $v_0, v_1, v_2$ :



vypočíta na vrch zásobníka hodnotu numerického výrazu

$$(x_2 \times 4) + (x_1 \times x_0)$$

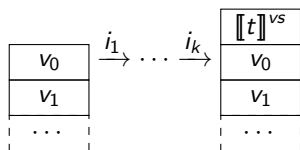
pri ohodnotení premenných  $vs = v_0, v_1, v_2, 0$

$$\begin{aligned} & \text{Run}(\text{LOAD}(2), \text{PUSH}(4), \text{MUL}, \text{LOAD}(2), \text{LOAD}(2), \text{MUL}, \text{ADD}, 0), vs) \\ &= \llbracket (x_2 \times 4) + (x_1 \times x_0) \rrbracket^{vs} \end{aligned}$$

### XIII.13 Programy a výrazy — kompilácia

---

- Numerické výrazy — jazyk vyššej úrovne
- Programy postfixového stroja — jazyk nižšej úrovne
- Preklad výrazov do programov — *kompilácia*
- Kompilátor — funkcia  $\text{Comp}(t)$
- Vytvorí program  $i_1, i_2, \dots, i_k, 0$ , ktorý na vrch zásobníka s hodnotami premenných vypočíta hodnotu výrazu  $t$



$$\text{Term}(t) \rightarrow \text{Program } \text{Comp}(t)$$

$$\text{Term}(t) \rightarrow \text{Run}(\text{Comp}(t), vs) = \llbracket t \rrbracket^{vs}$$

### XIII.14 Kompilácia — príklad

---

Videli sme, že

$$\begin{aligned} & \text{Run}(\text{LOAD}(2), \text{PUSH}(4), \text{MUL}, \text{LOAD}(2), \text{LOAD}(2), \text{MUL}, \text{ADD}, 0), vs) \\ &= \llbracket (x_2 \times 4) + (x_1 \times x_0) \rrbracket^{vs} \end{aligned}$$

Chceme, aby kompilátor skompiloval výraz

$$(x_2 \times 4) + (x_1 \times x_0)$$

do programu

LOAD(2), PUSH(4), MUL, LOAD(2), LOAD(2), MUL, ADD, 0

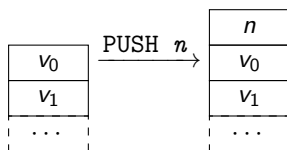
teda

$$\begin{aligned} & \text{Comp}((x_2 \times 4) + (x_1 \times x_0)) \\ &= \text{LOAD}(2), \text{PUSH}(4), \text{MUL}, \text{LOAD}(2), \text{LOAD}(2), \text{MUL}, \text{ADD}, 0 \end{aligned}$$

### XIII.15 Kompilácia — konštanty

---

- Hodnotou výrazu  $n$  je číslo  $n$
- Aký program vypočíta na vrch zásobníka hodnotu  $n$ ?

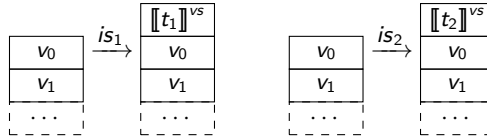


- Klauzula kompilátora:

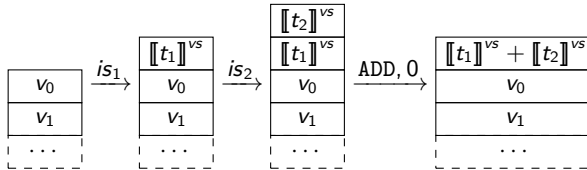
$$\text{Comp}(n) = \text{PUSH}(n), 0$$



- Hodnotou výrazu  $t_1 + \bullet t_2$  je číslo  $[[t_1]]^{vs} + [[t_2]]^{vs}$
- Akým programom ju vypočítame?
  1. Skompilujeme výrazy  $t_1$  a  $t_2$  Dostaneme programy  $\text{Comp}(t_1) = is_1$  a  $\text{Comp}(t_2) = is_2$



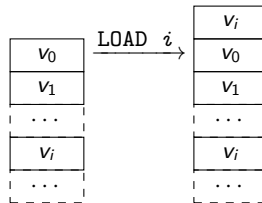
2. Zreťazíme ich a pridáme inštrukciu pre sčítanie



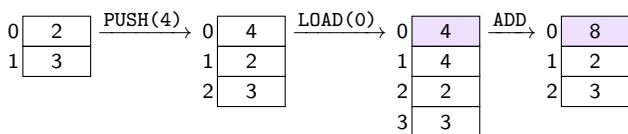
- Klauzula kompilátora:

$$\text{Comp}(t_1 + \bullet t_2) = \text{Comp}(t_1) \oplus \text{Comp}(t_2) \oplus (\text{ADD}, 0)$$

- Hodnotou výrazu  $x_i^\bullet$  je číslo  $vs[i]$
- Aký program dá na vrch zásobníka hodnotu  $vs[i]$ ?



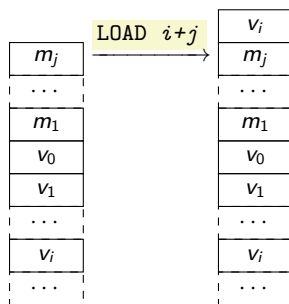
- Skompilujeme  $\text{Comp}(4 \bullet + \bullet x_0^\bullet) = \text{PUSH}(4), \text{LOAD}(0), \text{ADD}, 0$
- Otestujeme s ohodnotením 2, 3, 0:



ZLE!  $\text{LOAD}(0)$  malo načítať 2

### XIII.18 Kompilácia – medzivýsledky

- Kompilátor si musí pamätať počet  $j$  medzivýsledkov v zásobníku:  $\text{Comp}_1(t, j)$
- $\text{Comp}(t) = \text{Comp}_1(t, 0)$
- Aký program dá na vrch zásobníka hodnotu  $vs[j]$ ?  $\text{Comp}_1(x_i^\bullet, j) = ?$



- Počet medzivýsledkov vzrastie po vypočítaní  $t_1$  vo výrazoch  $t_1 \bullet t_2$  a  $t_1 \times^\bullet t_2$ :

$$\text{Comp}_1(t_1 \bullet t_2, j) = \text{Comp}_1(t_1, j) \oplus \text{Comp}_1(t_2, j+1) \oplus (\text{ADD}, 0)$$

## 19. Organizácia: Opravný test a skúška

### XIII.19 Náhradný/opravný test

---

**Termín:** piatok 24. mája o 9:30 v H6

**Účel:** nahradiť skóre z jedného semestrálneho testu

**Materiál:** ako nahrádzaný test

- Podmienky:**
- *záväzné prihlásenie* do 23. mája 23:59:59 (AIS)
    - účasť bez prihlásenia nie je možná
  - výber nahrádzaného testu priamo na termíne
    - neprítomní prihlásení: 2. test
  - nové skóre nahradí pôvodné skóre *bez ohľadu na ich vzájomný rozdiel*
    - ⇒ môžete si pohoršiť
    - skóre neprítomných prihlásených je 0

### XIII.20 Skúška

---

Termíny:	Dátum	Čas	Typ (max. počet študentov)
	piatok 24. mája	9:30	riadny (32)
	štvrtok 30. mája	14:00	riadny (32)
	piatok 7. júna	9:30	riadny (16), 1. opravný (16)
	štvrtok 20. júna	14:00	1. opravný (16), 2. opravný (16)
	piatok 28. júna	9:30	2. opravný (16)

**Podmienka:** skóre z testov  $\geq 30$

**Materiál:** úlohy najmä podľa ex11–ex14

- Konzultácie:**
- piatok 17. mája o 11:30–14:00 v H6
  - 29. mája, 6., 19. a 27. júna 13:00–15:00  
zraz na I-33b, presun do voľnej počítačovej haly

## XIV. prednáška

# Generovanie XML/XHTML

13. mája 2013

## 20. XML/XHTML

### 20.1. Štruktúra

#### XIV.1 Čo je XML/XHTML

---

XML/XHTML je formát na zápis štruktúrovaného textu

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Príklad</title>
  </head>
  <body>
    <p>Príklad
      <abbr title="Extensible Hypertext
        Markup Language">XHTML</abbr><br/>
      dokumentu.</p>
  </body>
</html>
```

#### XIV.2 Štruktúra XML/XHTML dokumentu

---

Štruktúra v XML dokumente — *elementy*

- ▶ úsek textu medzi otváracím tagom (<p>) a príslušným zatváracím tagom (</p>)

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Príklad</title>
  </head>
  <body>
```

```

<p>Príklad
  <abbr title="Extensible Hypertext
    Markup Language">XHTML</abbr><br/>
  dokumentu.</p>
</body>
</html>

```

#### XIV.3 Strom elementov

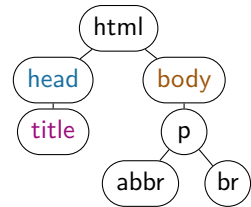
---

Elementy vytvárajú stromovú štruktúru:

```

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Príklad</title>
  </head>
  <body>
    ...
  </body>
</html>

```



Čo v strome chýba?

#### XIV.4 Neštruktúrovaný text

---

```

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Príklad</title>
  </head>
  <body>
    <p>Príklad
    <abbr title="Extensible Hypertext
      Markup Language">XHTML</abbr><br/>
    dokumentu.</p>
  </body>
</html>

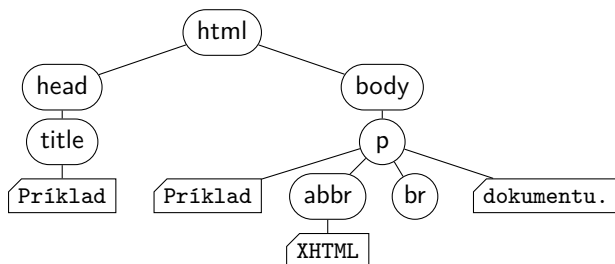
```

V strome elementov chýba neštruktúrovaný text

#### XIV.5 Strom elementových a textových uzlov

---

Pridáme uzly pre neštruktúrovaný text:



Všeobecný strom s dvoma druhmi uzlov:

- **elementové uzly** — ľubovoľne veľa detí (aj 0, napr. `br`)
- **textové uzly** — listy, žiadne deti

## 20.2. Kódovanie

#### XIV.6 Kódovanie stromu dokumentu v CL

---

- Strom XML dokumentu má 2 druhy uzlov
- Každý druh zakódujeme osobitným konštruktorom (podobne ako  $E$  a  $N_d$ )
- Textové uzly:  $X_s(t)$ 
  - $t$  — text v uzle — reťazec
- Elementové uzly:  $X_e(n, as, cs)$ 
  - $n$  — meno elementu (`html`, `body`, `p`, ...) — reťazec
  - $as$  — atribúty elementu
  - $cs$  — deti elementu

#### XIV.7 Kódovanie atribútov elementu

$Xe(n, as, cs)$

```
<a href="http://example.com/" class="external">link</a>
```

- Atribút je dvojica *meno=hodnota*
- Kódujeme dvojicou reťazcov:

$$\text{Attr}(n, v) \leftarrow \text{Str}(n) \wedge \text{Str}(v)$$

- Všetky atribúty elementu zakódujeme zoznamom:

$$\text{Lattr}(0)$$
$$\text{Lattr}(a, as) \leftarrow \text{Attr}(a) \wedge \text{Lattr}(as)$$

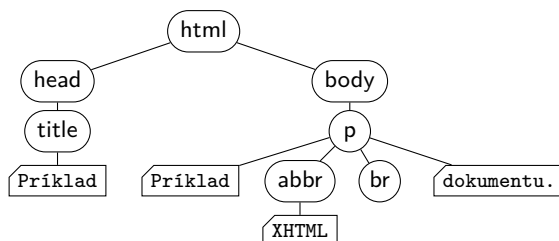
- Napríklad:

$$('href', 'http://example.com/'), ('class', 'external'), 0$$

#### XIV.8 Kódovanie detí elementu

$Xe(n, as, cs)$

- Elementové uzly majú rôzny počet detí:



- Deti zakódujeme do zoznamu uzlov XML stromu

#### XIV.9 Kódovanie XML dokumentu — príklad

---

```
Xe('html', (('xmlns', 'http://www.w3.org/1999/xhtml'), 0),
  Xe('head', 0,
    Xe('title', 0, Xs('Príklad'), 0),
    0),
  Xe('body', 0,
    Xe('p', 0,
      Xs('Príklad '),
      Xe('abbr', (('title', 'Extensible . . . Language'), 0),
      Xs('XHTML'),
      0),
      Xe('br', 0, 0),
      Xs(' dokumentu.'),
      0),
    0),
  0)
```

#### XIV.10 Kódovanie XML — predikát

---

- Štruktúru binárneho stromu sme popísali predikátom:

$Bt(\bullet)$

$$Bt\left(\frac{x}{\ell|r}\right) \leftarrow N(x) \wedge Bt(\ell) \wedge Bt(r)$$

- Štruktúru všeobecného stromu sme popísali predikátmi:

$Lgt(0)$

$$Lgt\left(\frac{x}{ts_1}, ts_2\right) \leftarrow N(x) \wedge Lgt(ts_1) \wedge Lgt(ts_2)$$

$$Gt\left(\frac{x}{ts}\right) \leftarrow N(x) \wedge Lgt(ts)$$

- Ako popíšeme štruktúru XML stromu?
  - Predikát  $X(x)$  —  $x$  kóduje uzol XML stromu
  - Pomocný predikát  $Lx(xs)$  —  $xs$  je zoznam XML uzlov



- Prirodzená definícia *vzájomnou rekurziou*:

$$\begin{aligned} X X_e(n, as, cs) &\leftarrow \text{Str}(n) \wedge \text{Lattr}(as) \wedge L_x(cs) \\ X X_s(t) &\leftarrow \text{Str}(t) \end{aligned}$$

$$\begin{aligned} L_x(0) \\ L_x(x, xs) &\leftarrow X(x) \wedge L_x(xs) \end{aligned}$$

- CL neumožňuje vzájomnú rekurziu:

$$\begin{aligned} L_x(0) \\ L_x(X_e(n, as, cs), xs) &\leftarrow \text{Str}(n) \wedge \text{Lattr}(as) \wedge L_x(cs) \wedge L_x(xs) \\ L_x(X_s(t), xs) &\leftarrow \text{Str}(t) \wedge L_x(xs) \end{aligned}$$

$$\begin{aligned} X X_e(n, as, cs) &\leftarrow \text{Str}(n) \wedge \text{Lattr}(as) \wedge L_x(cs) \\ X X_s(t) &\leftarrow \text{Str}(t) \end{aligned}$$

- Ak v CL zakódujeme strom XML dokumentu, zobrazíme ho ako skutočné XML formátom Xml:

Query:  $X_e('pokus', 0, X_s('text'), 0), 0 = xs : \text{Xml}$

Results:  $xs = \boxed{\langle pokus \rangle \text{text} \langle /pokus \rangle}$

- Formátom Xml môžeme zobrazovať *iba* hodnoty  $xs$ :
  - jednoprvkové zoznamy uzlov:  $L_x(xs) \wedge L(xs) = 1$
  - jediným prvkom je  $X_e$
  - ▶ XML dokument musí mať jeden koreňový element
- Skratková funkcia  $Xhtml(t, cs)$  — kostra XHTML dokumentu
  - $t$  — titulok — reťazec
  - $cs$  — obsah tela (body) —  $L_x$
 Formát Xml zobrazí výsledok ako skutočné XHTML

## 20.3. Neformálne typovanie

### XIV.13 Neformálne typovanie

---

- CL je beztypový jazyk — všetky dáta sú čísla
- Môžeme typovať neformálne predikátmi
- $F :: T_1 \rightarrow T_2$  znamená:
  - $T_1, T_2$  sú predikáty
  - argument funkcie  $F$  je typu  $T_1$
  - výsledok je typu  $T_2$
  - skratka za:

$$T_1(x) \rightarrow T_2(F(x))$$

- Napríklad funkcie na binárnych stromoch:

$$\text{Preorder} :: \text{Bt} \rightarrow \text{Ln}$$

$$\text{Insert} :: (\text{Bt}, \text{N}) \rightarrow \text{Bt}$$

$$\text{Extract\_max} :: \text{Bt} \rightarrow (\text{N}, \text{Bt})$$

- Ako otypujeme  $Xe, Xs, Xhtml$ ?

## 20.4. Generovanie

### XIV.14 Generovanie XML — príklad

---

Zadefinujme funkciu  $\text{Mk\_ul}(xs)$ , ktorá zo zoznamu reťazcov  $xs$  vytvorí nečíslovaný XHTML zoznam ( $ul$ ), ktorého položky ( $li$ ) obsahujú reťazce zo zoznamu  $xs$

- Otypujme túto funkciu — vytvorí jeden uzol XML stromu:

$$\text{Mk\_ul} :: \text{Lstr} \rightarrow \text{X}$$

- Naprogramujme ju:

- Potrebujeme pomocnú funkciu  $Mk\_lis(xs)$ , ktorá vytvorí zoznam položiek (uzlov XML stromu):

$$Mk\_lis :: Lstr \rightarrow Lx$$

$$Mk\_lis(0) = 0$$

$$Mk\_lis(x, xs) = Xe('li', 0, Xs(x), 0), Mk\_lis(xs)$$

- Vytvorené položky použijeme ako deti elementu ul:

$$Mk\_ul(xs) = Xe('ul', 0, Mk\_lis(xs))$$