

# Prednášky z Úvodu do deklaratívneho programovania

Ján Klúka

Letný semester 2013/2014

## 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 . . . . .	15
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 . . . . .	24
5.3. Substitúcia v parametri . . . . .	25
5.4. Verifikácia . . . . .	26
<b>III. Priradujúce diskriminácie</b>	
<b>Chvostová rekurzia</b>	<b>28</b>
5.5. Opakovanie . . . . .	28
<b>6. Priradujúce diskriminácie</b>	<b>29</b>
6.1. Priradenie . . . . .	29
6.2. Monadická diskriminácia . . . . .	30
<b>7. Chvostová rekurzia</b>	<b>32</b>
7.1. Cyklus while . . . . .	33
7.2. Fibonacci efektívne . . . . .	36
7.3. Cyklus for . . . . .	38
<b>IV. Binárna číselná sústava</b>	
<b>Párovanie</b>	<b>40</b>
7.4. Opakovanie . . . . .	40
<b>8. Binárna sústava</b>	<b>42</b>
8.1. Konštruktory . . . . .	43
8.2. Diskriminácia . . . . .	44
8.3. Rekurzia . . . . .	45
8.4. Aritmetika . . . . .	46
<b>9. Párovanie</b>	<b>49</b>
9.1. Kódovanie dátových štruktúr . . . . .	49

9.2. Párovacie funkcie . . . . .	50
9.2.1. Párovacia funkcia CL . . . . .	51
9.2.2. Programovanie s párovacou funkciou . . . . .	53
9.2.3. Tupling . . . . .	55

## V. Zoznamy.

### Chvostová rekurzia a tupling na zoznamoch **57**

## 10. Zoznamy **57**

10.1. Kódovanie konečných postupností . . . . .	58
10.2. Programovanie so zoznamami . . . . .	60
10.2.1. Zreťazenie . . . . .	60
10.2.2. Porovnanie s Pythonom . . . . .	62
10.2.3. Obrátenie . . . . .	63
10.3. Chvostová rekurzia na zoznamoch . . . . .	63
10.3.1. Súčet prvkov zoznamu . . . . .	63
10.3.2. Obrátenie zoznamu . . . . .	64
10.4. Tupling zoznamových operácií . . . . .	66
10.4.1. Split = Take, Drop . . . . .	66
10.4.2. Zip, Unzip . . . . .	67

## VI. Triedenie zoznamov

### Zoznamová reprezentácia množín **69**

## 11. Predikáty **69**

## 12. Triedenie zoznamov **71**

12.1. Špecifikácia triedenia . . . . .	71
12.2. Triedenie vsúvaním . . . . .	74
12.3. Zlúčenie utriedených zoznamov . . . . .	76
12.4. Triedenie zlučováním . . . . .	77

## 13. Zoznamová reprezentácia konečných množín **79**

<b>VII. Kombinatorické funkcie na zoznamoch</b>	<b>81</b>
<b>14. Kombinatorické funkcie na zoznamoch</b>	<b>82</b>
14.1. Súvislé úseky . . . . .	82
14.1.1. Sufixy . . . . .	82
14.1.2. Prefixy . . . . .	84
14.1.3. Segmenty . . . . .	87
14.2. Podpostupnosti . . . . .	88
14.3. Permutácie . . . . .	90
14.4. Ďalšie úlohy . . . . .	91
<b>VIII Binárne stromy</b>	<b>92</b>
<b>15. Binárne stromy</b>	<b>92</b>
15.1. Kódovanie . . . . .	92
15.2. Operácie . . . . .	95
15.3. Prechody . . . . .	96
15.4. Vyhľadávacie stromy . . . . .	99
<b>IX. Číslovanie binárnych stromov</b>	
<b>Všeobecné stromy</b>	<b>103</b>
<b>15. Binárne stromy</b>	<b>103</b>
15.5. Úplné binárne stromy . . . . .	103
15.6. Číslovanie vrcholov v binárnych stromoch . . . . .	105
<b>16. Všeobecné stromy</b>	<b>106</b>
16.1. Kódovanie . . . . .	106
16.2. Programovanie so všeobecnými stromami . . . . .	109
16.3. Prechody všeobecnými stromami . . . . .	110

<b>X. Aritmetické výrazy a výrokové formuly</b>	<b>112</b>
<b>17. Aritmetické výrazy a výrokové formuly</b>	<b>112</b>
17.1. Aritmetické výrazy . . . . .	112
17.2. Transformácia . . . . .	115
<b>XI. Výrokové formuly</b>	<b>119</b>
<b>17. Aritmetické výrazy a výrokové formuly (pokračovanie)</b>	<b>119</b>
17.3. Výrokové formuly . . . . .	119
17.4. Transformácia výrokových formúl . . . . .	121
<b>Organizácia skúšky a opravného testu</b>	<b>124</b>
<b>XII. Postfixový stroj. Kompilátor výrazov</b>	<b>126</b>
<b>18. Postfixový stroj. Kompilátor výrazov</b>	<b>126</b>
18.1. Postfixový stroj . . . . .	126
18.2. Kompilácia aritmetických výrazov . . . . .	129
18.3. Podmienené výrazy . . . . .	133
<b>XIII. Generovanie XML/XHTML</b>	<b>137</b>
<b>19. XML/XHTML</b>	<b>137</b>
19.1. Štruktúra . . . . .	137
19.2. Kódovanie . . . . .	139
19.3. Neformálne typovanie . . . . .	143
19.4. Generovanie . . . . .	144

## I. prednáška

# Organizácia kurzu

# Deklaratívne programovanie v CL

# Explicitné definície

17. februára 2014

## 1. Organizácia kurzu

### 1.1. Rozvrh a kontakty

#### 1.1 Organizácia kurzu — rozvrh a kontakty

---

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

**Cvičenia** utorok 18:10 H6  $\text{1ai}\{3\}$   
streda 15:40 H6  $\text{1ai}\{1,2,4\}$

**Druháci a tretiaci: Prihláste sa** na termín podľa vlastného výberu  
(**povinne do 28. 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.

*bINF* → Ján Komara (I-16)

**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

### I.2 Organizácia kurzu – hodnotenie

---

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

neúčast'  $\wedge$  *ospravedlnenie* (do 3 dní [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) v 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

### I.3 Deklaratívne programovanie

---

- Spôsob (*paradigma*) programovania
- Dôraz na *popis problému* (čo sa má počítať), nie (len) na postup riešenia (*ako sa to počíta*)
- Často nejaký druh *pravidiel*
- Program je zapísaný v jazyku vhodnej logiky

- Význam (*sémantika*) programu: matematický objekt  
Program *definuje* funkciu/reláciu (alebo ich systém)
- 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* (podmiených 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 podmiených 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 Kľuka

## 3.1. Základy syntaxe

### I.9 Základy syntaxe — premenné

---

**Definícia 1.** *Premenná* je neprázdna postupnosť malých písmen anglickej abecedy nasledovaná najviac dvoma číslicami.

*Príklad 2.*  $x$ ,  $y$ ,  $a3$ ,  $zoznam99$ ;

~~$X$~~ ,  ~~$velkePismo$~~ ,  ~~$podciar\_kovnik$~~ ,  ~~$y2k$~~ ,  ~~$p123$~~ ,  ~~$kEdel$~~

### I.10 Základy syntaxe — funkčné symboly a ich arita

---

**Definícia 3.** *Funkčný symbol* je postupnosť znakov začínajúca veľkým písmenom a pokračujúca ľubovoľnou postupnosťou malých písmen, číslic a podčiarkovníkov.

Každý funkčný symbol má priradený počet argumentov — *aritu*.

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

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

$n = 1$  *unárny f. s.*,  $n = 3$  *ternárny f. s.*

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

*Príklad 4.*  $F$ ,  $Perms$ ,  $Tree2string$ ,  $F\_n1\_m1$ ,  $Q54321$ ;

~~$malepismo$~~ ,  ~~$VelkePismo$~~ ,  ~~$podciarkovnik$~~ ,  ~~$Zvyšok$~~

### I.11 Základy syntaxe — konštanty

---

**Definícia 5.** *Číselná konštanta* (číselný literál) je postupnosť desiatkových číslic.

*Príklad 6.*  $0$ ,  $8$ ,  $2020$ ,  $0123456789876543210$

Ďalšie špeciálne druhy konštánt neskôr.

### I.12 Termy

---

**Definícia 7.** *Term* je postupnosť symbolov, ktorá je buď

- *premenná*,

- konštanta (ľubovoľného druhu),
- aplikácia niektorej zabudovanej binárnej funkcie  
 $(\tau + \sigma) (\tau \div \sigma) (\tau \cdot \sigma) (\tau \div \sigma) (\tau \bmod \sigma) (\tau, \sigma) (\tau \oplus \sigma)$   
 ak  $\tau$  a  $\sigma$  sú termy,

- aplikácia  $n$ -árneho funkčného symbolu  $F$  pre  $n \geq 1$

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

ak  $\tau_1, \tau_2, \dots, \tau_n$  sú termy.

Nič iné nie je termom.

Dohoda:  $\tau, \sigma$  označujú termy.

### I.13 Termy — príklady a zjednodušenia

Príklady termov:

Nemusíme písať všetky zátvorky:

$(a + b)$	$a + b$
$((a \cdot a) + ((2 \cdot a) \cdot b)) + (b \cdot b)$	$a \cdot a + 2 \cdot a \cdot b + b \cdot b$
$Sq(x)$	$Sq(x)$
$Sq((x + (y \bmod Zaklad)))$	$Sq(x + y \bmod Zaklad)$
$Min(Sq(Max(a, b)),$ $Min((x + 1), (y + 1)))$	$Min(Sq Max(a, b),$ $Min(x + 1, y + 1))$

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

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.14 Niektoré funkcie zabudované v CL

---

Binárne aritmetické funkcie *na prirodzených číslach*:

+ sčítanie (v zdrojovom kóde „+“)

• násobenie (v zdrojovom kóde „\*“)

÷ *modifikované* odčítanie (v zdrojovom kóde „-“)

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

÷ delenie (v zdrojovom kóde „/“)

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.15 Moduly a definície funkcií

---

**Definícia 8.** CL *modul* je postupnosť definícií funkcií

**Definícia 9.** *Definícia funkcie* je množina *klauzúl* (clauses) — podmienených rovností:

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

...

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

skonštruovaná určitým spôsobom (postupne sa ho naučíme).

- Určuje *spôsob výpočtu* hodnoty funkcie pre ľubovoľné hodnoty jej argumentov
- Jednoznačne určuje *význam* (sémantiku) funkčného symbolu

## I.16 Explicitné definície

---

**Definícia 10.** Definícia funkcie je *explicitná*, ak klauzuly obsahujú iba premenné a zabudované alebo *predtým* definované konštanty a funkcie.

*Príklad 11.* Najjednoduchšie explicitné definície majú iba jednu rovnosť:

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

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

$$Z(x) = 0 \quad \text{Twice}(x) = 2 \cdot x \quad \text{Square}(x) = x \cdot x$$

$$\text{Binomial1}(a, b) = a \cdot a + 2 \cdot a \cdot b + b \cdot b$$

$$\text{Binomial2}(a, b) = \text{Square}(a + b)$$

## 4.1. Explicitné definície s jednou podmienkou

### I.17 Explicitné definície s podmienkami

---

- Zložitejšie definície majú viacero klauzúl

$$\text{Sgn}(x) = 0 \leftarrow x = 0 \quad \text{Min}(x, y) = y \leftarrow x > y$$

$$\text{Sgn}(x) = 1 \leftarrow x \neq 0 \quad \text{Min}(x, y) = x \leftarrow x \leq y$$

- Argumenty funkcie musia byť označené *rovnakými premennými*:

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

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

- *Podmienky* v klauzulách musia:
  - byť *výlučné* a
  - pokryť *všetky možné prípady*

Potom sa pri výpočte vždy dá použiť *práve jedna* klauzula

- CL **nerozpozná výlučnosť hocijakých podmienok**

Postupne sa naučíme, ktoré sady podmienok rozpoznáva

### I.18 Diskriminácie

---

Rozpoznateľnú 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

- môžeme napísať napríklad:

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

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

$$G(x, y) = x \cdot y \leftarrow \text{Square}(y) > x$$

$$G(x, y) = x+y \leftarrow \text{Square}(y) \leq x$$

- nemôžeme napísať napríklad:

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

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

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

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

## 4.2. Explicitné definície s viacerými podmienkami

### I.19 Kombinované podmienky

---

- Podmienky v jednej klauzule *nemôžeme* kombinovať ľubovoľne
- Môžeme ich spájať **iba** logickou spojku  $\wedge$
- CL musí rozpoznať, že podmienky sú vzájomne výlučné a pokrývajú všetky prípady
- **Nemôžeme** napríklad napísať:

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

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

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

## I.20 Ako kombinovať podmienky

---

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

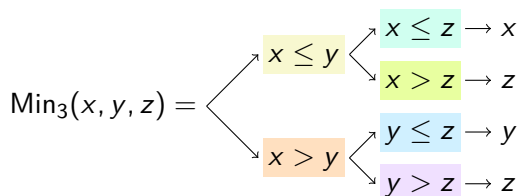
$$\text{Min}_3(x, y, z) = x \leftarrow \begin{array}{l} x \leq y \\ x \leq z \end{array}$$

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

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

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

vzniknú vnorením diskriminácií:



## I.21 Ako kombinovať podmienky – recept

---

Zaručene správna konštrukcia podmienok:

Kým nevieme určiť výsledok, vnárame dovolené diskriminácie



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

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

$$\text{Min}_3(x, y, z) = ? \leftarrow x > y \quad \text{nezabudneme na ostatné časti jej diskriminácie}$$

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

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

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

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

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

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

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

### 4.3. Verifikácia explicitne definovaných funkcií

#### I.22 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*

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

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

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

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

Ako na to?

$$\underbrace{\text{Min}_3(x, y, z) \leq x}_{(a)} \wedge \underbrace{\text{Min}_3(x, y, z) \leq y}_{(b)} \wedge \underbrace{\text{Min}_3(x, y, z) \leq z}_{(c)} \quad (*)$$

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  $\text{Min}_3(x, y, z) = x$ . Lahko overíme, že platia všetky konjunkty dokazovanej vlastnosti (\*): zrejme platí  $x \leq x$  (a) a podľa aktuálnych predpokladov  $x \leq y$  (b) a  $x \leq z$  (c).
  - Ak  $x > z$ , podľa (2) je  $\text{Min}_3(x, y, z) = z$ . Opäť overíme platnosť všetkých konjunktov (\*): Podľa aktuálnych predpokladov platí  $z < x$  (a),  $z < x \leq y$  (b) a zrejme  $z \leq z$  (c).
- Ak  $x > y$ , hodnotu  $\text{Min}_3$  určíme porovnaním  $y$  a  $z$ . Postup je podobný ako v predchádzajúcom prípade.

## II. prednáška

# Rekurzia

24. februára 2014

### 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 — rozpoznávané 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

---

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

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

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

$$\text{Max}_3(x, y, z) = z \leftarrow x < y \wedge y < z$$

Dokážte, že

$$\text{Max}_3(a, b, c) = \text{Max}_3(b, a, c) \quad (1)$$

Dokazujeme rovnostnú vlastnosť, preto 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 vyzerat'?

## II.3 Prémiová domáca úloha — dôkaz

---

$$\text{Max}_3(a, b, c) \stackrel{?}{=} \text{Max}_3(b, a, c)$$

Nech  $a$ ,  $b$  a  $c$  sú ľubovoľné prirodzené čísla.

Označme  $L := \text{Max}_3(a, b, c)$  a  $P := \text{Max}_3(b, a, c)$  a uvedomme si dosadenia:

$$\text{Max}_3(a, b, c) = a \leftarrow a \geq b \wedge a \geq c \quad (1) \quad \text{Max}_3(b, a, c) = b \leftarrow b \geq a \wedge b \geq c$$

$$\text{Max}_3(a, b, c) = c \leftarrow a \geq b \wedge a < c \quad (2) \quad \text{Max}_3(b, a, c) = c \leftarrow b \geq a \wedge b < c$$

$$\text{Max}_3(a, b, c) = b \leftarrow a < b \wedge b \geq c \quad (3) \quad \text{Max}_3(b, a, c) = a \leftarrow b < a \wedge a \geq c$$

$$\text{Max}_3(a, b, c) = c \leftarrow a < b \wedge b < c \quad (4) \quad \text{Max}_3(b, a, c) = c \leftarrow b < a \wedge a < c$$

Výhodné je uvažovať *tri* možnosti (*trichotómia*):

**Ak  $a > b$ , tak**

- ak navyše  $a \geq c$ , máme  $L \stackrel{(1)}{=} a$  a  $P \stackrel{(3)}{=} a$ , teda  $L = P$ ;
- ak naopak  $a < c$ , máme  $L \stackrel{(2)}{=} c$  a  $P \stackrel{(4)}{=} c$ , teda opäť  $L = P$ .

**Ak  $a = b$ , tak**

- pre  $a \geq c$  (teda aj  $b \geq c$ ) máme  $L \stackrel{(1)}{=} a = b \stackrel{(1)}{=} P$ ;
- naopak pre  $a < c$  (teda aj  $b < c$ ) dostávame  $L \stackrel{(2)}{=} c \stackrel{(2)}{=} P$ .

**Ak  $a < b$ , tak**

- v prípade  $b \geq c$  máme  $L \stackrel{(3)}{=} b \stackrel{(1)}{=} P$
- a v prípade  $b < c$  máme  $L \stackrel{(4)}{=} c \stackrel{(2)}{=} P$ .

## 5. Rekurzívne definície

### II.4 Rekurzia — princíp

---

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

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

- Rôzne druhy rekurzcie — rôzne zjednodušenia rekurzívnych argumentov

### 5.1. Primitívna rekurzia

#### II.5 Primitívna rekurzia

---

- *Primitívna rekurzia*: rekurzívny 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:

$$\begin{array}{l|l} \sum_{i=0}^n i = 0 & \leftarrow n = 0 \\ \sum_{i=0}^n i = n + \sum_{i=0}^{n-1} i & \leftarrow n \neq 0 \end{array} \quad \left| \quad \begin{array}{l} \text{Sum}(n) = 0 \quad \leftarrow n = 0 \\ \text{Sum}(n) = n + \text{Sum}(n - 1) \quad \leftarrow n \neq 0 \end{array} \right.$$

**Definícia 12.** Funkcia  $F$  s aritou  $n+1$  je definovaná *primitívnou rekurziou* z  $n$ -árnej funkcie  $G$  a  $(n+2)$ -árnej funkcie  $H$ , ak platí:

$$\begin{aligned}F(0, x_1, \dots, x_n) &= G(x_1, \dots, x_n) \\ F(i+1, x_1, \dots, x_n) &= H(i, F(i, x_1, \dots, x_n), x_1, \dots, x_n)\end{aligned}$$

## II.6 Primitívna rekúzia — symbolický výpočet

---

- Výpočet *prepísovaním* výrazov do základného tvaru — čísla
- Prepisuje sa *najvnútornejší* výraz, ktorý nie je v základnom tvare  
⇒ Najprv argumenty, potom aplikácia funkcie
- Aplikácia funkcie  $F$  sa prepisuje podľa klauzúl definície  $F$

Príklad 13.

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

## II.7 Primitívna rekúzia — strojový výpočet

---

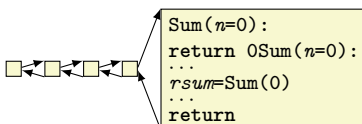
- CL v skutočnosti kompiluje program do *bajtkódu* (strojový kód virtuálneho stroja)
- Čo je potrebné na strojovú realizáciu rekúzívneho výpočtu?

*Zásobník*

- V Pythone:

```
def Sum(n):           Sum(n) = 0           ← n = 0
    if n = 0:         Sum(n) = n + Sum(n ÷ 1) ← n ≠ 0
        return 0
    else:
        rsum = Sum(n - 1)
        return n + rsum
```

- Schematicky:



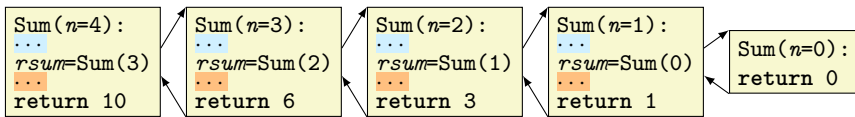
**Pozorovanie 14.** *Rekúzia je dvojfázový proces:*

1. príprava rekurzívneho volania
2. použitie jeho výsledku

Príklad 15.

```
def Sum(n):
    if n = 0:
        return 0
    else:
        rn = n - 1
        rsum = Sum(rn)
        sum = n + rsum
        return sum
```

$Sum(n) = 0 \quad \leftarrow n = 0$   
 $Sum(n) = n + Sum(n - 1) \quad \leftarrow n \neq 0$



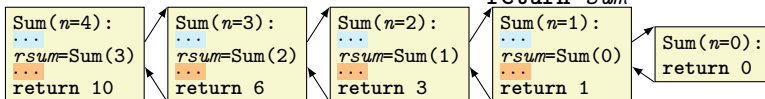
11.9 Primitívna rekúzia a iterácia

**Pozorovanie 16.** *Ak sa v definícii funkcie vyskytuje iba jedno rekurzívne volanie a nemenia sa parametre (iné argumenty ako klesajúce rekurzívne), dá sa rekúzia počítať zdola nahor cyklom (iteráciou). Rekurzívne volanie nahradí pomocná premenná.*

Príklad 17.

```
Sum(n) = 0 \quad \leftarrow n = 0
Sum(n) = n + Sum(n - 1) \quad \leftarrow n \neq 0
```

```
def Sum(x):
    sum = 0 # sum == Sum(0)
    for n in range(1, x + 1):
        # sum == Sum(n - 1)
        sum = n + sum
        # sum == Sum(n)
    return sum
```



## II.10 Aritmetika primitívnou rekurziou

---

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

$$x \cdot y = 0 \quad \leftarrow x = 0$$

$$x \cdot y = \dots (x \div 1) \cdot y \dots \leftarrow x \neq 0$$

- 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.11 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:



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

Je zaručené zmenšovanie rekurzívneho argumentu?

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

## II.12 Všeobecná rekúzia s viacerými rekurzívnymi volaniami

---

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

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

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

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

- Funkcia  $\text{Fib}(n) = \text{fib}_n$  počíta  $n$ -tý prvok Fibonacciho postupnosti
- Definícia všeobecnou rekúziou
  - s *dvoma* rekurzívnymi volaniami, a
  - so zloženými podmienkami:

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

## 5.3. Všeobecná rekúzia so substitúciou v parametri

### II.13 Substitúcia v parametri

---

- Najväčší spoločný deliteľ:

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

- Využijeme

$$\begin{aligned}
 & x \mid 0 \\
 & x \neq 0 \wedge d \mid x \rightarrow d \mid y \leftrightarrow d \mid y \bmod x \\
 & x \neq 0 \rightarrow y \bmod x < x
 \end{aligned}$$

- Definícia všeobecnou rekurziou:

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

- 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 rekurzia so *substitúciou v parametri*

## 5.4. Verifikácia rekurzívne definovaných funkcií indukciou

### II.14 Verifikácia primitívne rekurzívnych funkcií

Funkcia

$$\begin{aligned}
 \text{Sum}(n) &= 0 && \leftarrow n = 0 \\
 \text{Sum}(n) &= n + \text{Sum}(n \div 1) && \leftarrow n \neq 0
 \end{aligned}$$

má dobre známu vlastnosť:

$$\text{Sum}(n) = \frac{n \cdot (n + 1)}{2} \tag{*}$$

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  $\text{Sum}(n) = n + \text{Sum}(n \div 1)$  a  $\frac{n \cdot (n+1)}{2} = \frac{((n \div 1) + 1) \cdot (n+1)}{2} = \frac{(n \div 1) \cdot n + 2 \cdot n}{2} = n + \frac{(n \div 1) \cdot n}{2}$ .

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

Potrebujeme, aby

$$\text{Sum}(n \div 1) = \frac{(n \div 1) \cdot n}{2},$$

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

Inak povedané, potrebujeme *indukčný predpoklad*.

Začnime teda odznova:

### II.15 Primitívna rekurgia a matematická indukcia

$$\begin{array}{l} \text{Sum}(n) = 0 \quad \leftarrow n = 0 \\ \text{Sum}(n) = n + \text{Sum}(n \div 1) \quad \leftarrow n \neq 0 \end{array} \quad \text{Sum}(n) = \frac{n \cdot (n+1)}{2} \quad (*)$$

Dôkaz (\*) matematickou indukciou na  $n$ :

1. Báza: Dokážme (\*) pre 0:

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

2. Indukčný krok:

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

Dokážme pre  $n + 1$ :

$$\begin{aligned} \text{Sum}(n+1) &\stackrel{\text{def}}{=} n+1 + \text{Sum}(n+1 \div 1) \stackrel{\text{IP}}{=} \\ &\stackrel{\text{IP}}{=} n+1 + \frac{n \cdot (n+1)}{2} = \frac{2 \cdot n + 2 + n \cdot (n+1)}{2} = \frac{(n+1) \cdot (n+2)}{2}. \end{aligned}$$

### II.16 Všeobecná rekurgia a ... ?

Funkcie definované primitívnou rekurziou spravidla verifikujeme matematickou indukciou.

Funkcie definované všeobecnou rekurziou spravidla verifikujeme ... ?

# III. prednáška

## Priradujúce diskriminácie

### Chvostová rekurgia

3. marca 2014

## 5.5. Opakovanie

### III.1 Opakovanie: Rekurgia

---

#### Rekurgia

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:

$$\begin{array}{l} n! = 1 \quad \leftarrow n = 0 \\ n! = n \cdot m! \quad \leftarrow n \neq 0 \wedge n \div 1 = m \end{array}$$

$$\begin{array}{l} \text{gcd}(x, y) = y \quad \leftarrow x = 0 \\ \text{gcd}(x, y) = \text{gcd}(x_1, y_1) \quad \leftarrow x \neq 0 \wedge y \bmod x = x_1 \wedge x = y_1 \end{array}$$

Premenné v *deklaratívnych* jazykoch pomenúvajú *hodnoty*

Premenné v *imperatívnych* jazykoch pomenúvajú miesta v pamäti, do ktorých je možné ukladať hodnoty

## 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:

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

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

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

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

### 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:

$$0! = 1$$

$$x^0 = 1$$

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

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



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:

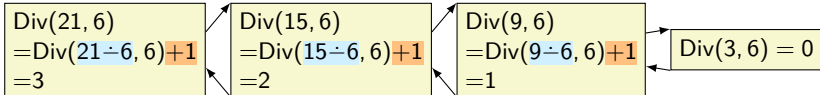
$$\begin{array}{lll}
 \text{fib}_n = 0 & \leftarrow n = 0 & \text{fib}_0 = 0 \\
 \text{fib}_n = 1 & \leftarrow n = 1 & \text{fib}_1 = 1 \\
 \text{fib}_n = \text{fib}_{m+1} + \text{fib}_m & \leftarrow n = m + 2 & \text{fib}_{n+2} = \text{fib}_{n+1} + \text{fib}_n
 \end{array}$$

# 7. Chvostová rekúzia

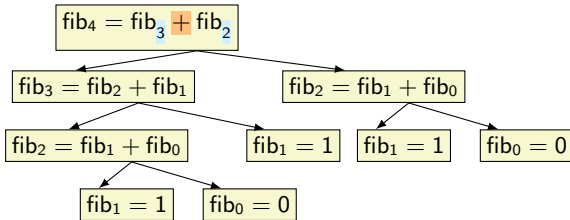
## III.7 Rekúzia

Rekúzia je *dvojfázový* proces:

1. príprava rekúziívneho volania
2. použitie jeho výsledku



Výpočet môže byť stromový:



## III.8 Rekúzia iteráciou

Jednoduché rekúziívne definície sa dajú počítať imperatívnym cyklom

Príklad 18.

```
n! = 1            ← n ≤ 0      def Fact(x):  
n! = n·(n ÷ 1)! ← n > 0      f = 1  
                                 # f == 0!  
                                 for n in range(1,x+1):  
                                 # f == (n ÷ 1)!  
                                 f = n * f  
                                 # f == n!  
                                 return f
```

Cyklus opakuje — *iteruje* výpočet vo svojom tele

Cyklus = *iterácia*

Namiesto výsledku rekúziívneho volania používa pomocnú premennú



**Definícia 19.** Funkcia  $F$  je definovaná *chvostovou* rekúziou (*tail recursion*), ak každá jej rekurzívna klauzula má tvar:

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

kde termy  $\tau_1, \dots, \tau_n$  neobsahujú rekurzívne aplikácie  $F$ .

Chvostová rekúzia – obzvlášť jednoduchý typ rekúzie:

- za každej podmienky najviac jedno rekurzívne volanie,
- žiadne ďalšie spracovanie výsledku rekurzívneho volania

Dá sa *mechanicky* preložiť do cyklu (a naopak)

*Príklad 20.*

Tail_fact( $n, f$ ) = $f$	$\leftarrow n = 0$	while $n \neq 0$ :
Tail_fact( $n, f$ ) = Tail_fact( $n \div 1, n \cdot f$ )	$\leftarrow n \neq 0$	$f = n * f$
		$n = n - 1$

Kompilátory to vedia, naučme sa to aj my!

## 7.1. Rekúzia pre cyklus while

$n! = 1$	$\leftarrow n = 0$	def Fact( $n$ ):
$n! = n \cdot ((n \div 1)!) \leftarrow n \neq 0$		$f = 1$
		while $n \neq 0$ :
		$f = n * f$
		$n = n - 1$
		return $f$

1. Pythonovský program rozdelíme na 2 časti:

a) inicializácia

$$f = 1$$

b) cyklus a vrátenie výsledku

```

while n != 0:
    f = f * n
    n = n - 1
return f

```

Aký je *matematický* význam tejto časti programu?

- ▶ Funkcia, ktorá každej začiatočnej hodnote  $n$  a  $f$  priradí koncovú hodnotu  $f$
- ▶ Nazveme ju `While_fact`

### III.12 Chvost. rek. pre cyklus `while` — cyklus ako funkcia

2. Naprogramujeme funkciu `While_fact` tak, aby počítala to, čo cyklus:

- Dva argumenty — hodnoty pythonovských premenných  $n$  a  $f$
- Aká bude koncová hodnota  $f$ , keď hodnota  $n = 0$ ?  
Rovnaká ako začiatočná hodnota  $f$
- Aká bude koncová hodnota  $f$ , 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  $f$ )

$$\text{While\_fact}(n, f) = \overbrace{\text{While\_fact}(n_1, f_1)}^{\text{opakovanie}} \leftarrow \overbrace{n \neq 0}^{\text{podmienka}} \wedge \underbrace{f \cdot n = f_1 \wedge n \div 1 = n_1}_{\text{telo cyklu}}$$

$$\text{While\_fact}(n, f) = \underbrace{f}_{\text{výsledok}} \leftarrow \underbrace{n = 0}_{\text{negácia podmienky}}$$

Invariant:

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

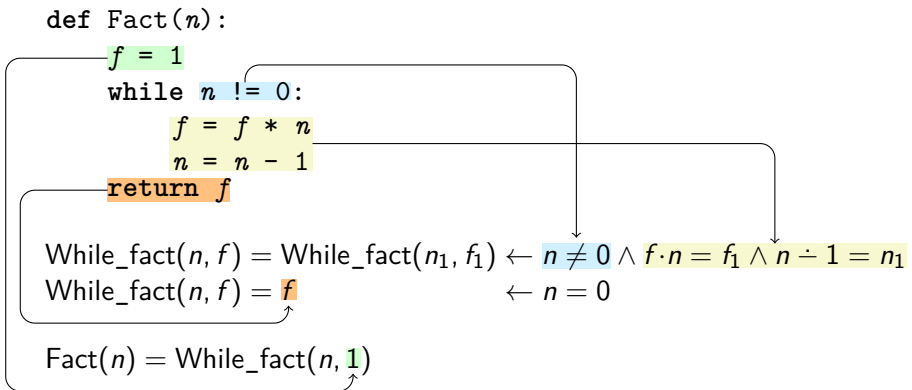
III.13 Chvost. rek. pre cyklus while – inicializácia

3. Inicializácia a spustenie cyklu

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

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

III.14 Chvost. rek. pre cyklus while – rekapitulácia

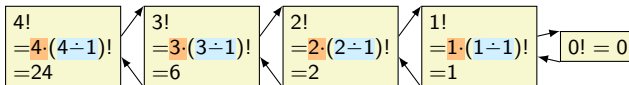


Stručnejšie a elegantnejšie s monadickou diskrimináciou:

$$\begin{aligned} \text{While\_fact}(n + 1, f) &= \text{While\_fact}(n, f \cdot (n + 1)) \\ \text{While\_fact}(0, f) &= f \end{aligned}$$

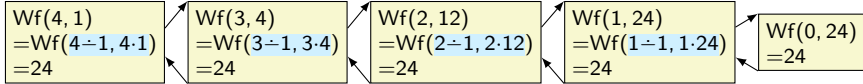
III.15 Chvostová rekúzia pre cyklus while – výpočet

Priebeh výpočtu obyčajnej definície  $n!$ :



Priebeh výpočtu Fact pomocou While\_fact (skrátene na Wf):

Fact(4) = Wf(4, 1) →



While\_fact: *chvostová* rekurzia

- Žiadne výpočty po návrate z rekurzívneho volania
- Nepotrebuje zásobník

### III.16 Chvostová rekurzia — príklad z minulého cvičenia

Chvostovú rekuziu 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?

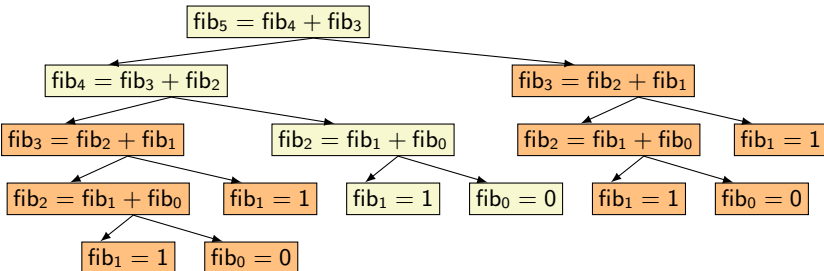
## 7.2. Efektívny výpočet Fibonacciho postupnosti

### III.17 Fibonacciho postupnosť *neefektívne*

Na pripomenutie:  $n$ -tý prvok Fibonacciho postupnosti

$$\text{fib}_0 = 0 \quad \text{fib}_1 = 1 \quad \text{fib}_{n+2} = \text{fib}_{n+1} + \text{fib}_n$$

Výpočet obyčajnou rekuziou je veľmi neefektívny:



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

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

### III.18 Fibonacciho postupnosť a králiky

---

Efektívny výpočet  $\text{fib}_n$  – zdola nahor:

- pamätáme si dva prvky postupnosti (na začiatku prvé dva)
  - počítame nasledujúci prvok
- ⇒ 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.19 Fibonacciho postupnosť efektívne imperatívne

---

```
def Fib(n):
    if n = 0:
        return 0
    else:
        n = n - 1
        m = 0 # fib0
        d = 1 # fib1
        while n != 0:
            # m = fibi ∧ d = fibi+1
            m1 = d
```

```

    d1 = m + d
    m = m1
    d = d1
    # m = fibi+1 ∧ d = fibi+2
    n = n - 1
return d

```

### III.20 Fibonacciho postupnosť efektívne deklaratívne

---

- Chvostová rekúzia pre cyklus while:

$$\text{While\_fib}(n, m, d) = \text{While\_fib}(n - 1, m_1, d_1) \quad \leftarrow n \neq 0 \wedge d = m_1 \wedge d + m = d_1$$

$$\text{While\_fib}(n, m, d) = d \quad \leftarrow n = 0$$

Stručnejšie:

$$\text{While\_fib}(n + 1, m, d) = \text{While\_fib}(n, d, d + m)$$

$$\text{While\_fib}(0, m, d) = d$$

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:

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

## 7.3. Rekúzia pre cyklus for

### III.21 Spätná chvostová rekúzia pre cyklus for

---

Výpočet faktoriálu cyklom for:

```

def Fact(n):
    f = 1
    for i in range(1, n+1):
        f = f * i
    return f

```

Deklaratívne:

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

$$\text{For\_fact}(i, n, f) = \text{For\_fact}(i + 1, n, f \cdot i) \leftarrow i \leq n$$

$$\text{For\_fact}(i, n, f) = f \leftarrow i > n$$

- Inicializácia premenných a štart cyklu:

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

### III.22 Spätná vs. primitívna rekúzia

---

Spätná rekúzia, napríklad

$$\text{For\_fact}(i, n, f) = \text{For\_fact}(i + 1, n, f \cdot i) \leftarrow i \leq n$$

$$\text{For\_fact}(i, n, f) = f \leftarrow i > n,$$

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

# IV. prednáška

## Binárna číselná sústava

### Párovanie

11. marca 2014

#### 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

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

###### Zovšeobecnená monadická diskriminácia

Skratka kaskády monadických diskriminácií

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

##### IV.2 Opakovanie: Chvostová rekúzia

###### Chvostová rekúzia

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

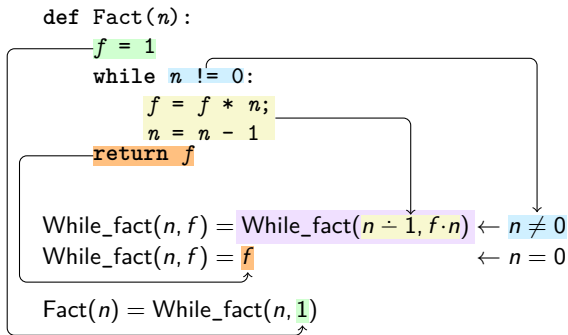


- Hodnota funkcie v rekurzívnom prípade = hodnote rekurzívneho volania b
- Výpočty iba v argumentoch rekurzívneho volania
- Pri výpočte nie je potrebný zásobník
- Do strojového kódu sa kompiluje ako cyklus

### IV.3 Opakovanie: Chvostová rekurzia a cyklus while

**Cyklus while** ~ chvostová rekurzia riadená podmienkou cyklu

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



### IV.4 Opakovanie: Chvostová rekurzia a for

**Cyklus for** ~ spätná chvostová rekurzia

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

```
def Fact(n):
    f = 1
    for i in range(1, n+1):
        f = f * i
    return f
```

$\text{For\_fact}(i, n, f) = \text{For\_fact}(i+1, n, f \cdot i) \leftarrow i \leq n$   $i \leq n \rightarrow$   
 $\text{For\_fact}(i, n, f) = f$   $\leftarrow i > n$   $n+1 - (i+1) < n+1 - i$   
 $\text{Fact}(n) = \text{For\_fact}(1, n, 1)$

## 8. Binárna číselná sústava

### IV.5 Binárna číselná sústava

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

$$13 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (1101)_b$$

- Vo všeobecnosti

$$\sum_{i=0}^k d_i \cdot 2^i = (d_k d_{k-1} \dots d_1 d_0)_b, \quad 0 \leq d_i < 2$$

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

$$\begin{aligned}
 & (d_k d_{k-1} \dots d_1 d_0)_b = \\
 & = (0d_k d_{k-1} \dots d_1 d_0)_b = \\
 & = (00d_k d_{k-1} \dots d_1 d_0)_b = \\
 & = \dots
 \end{aligned}$$

### IV.6 Binárny zápis a Hornerova schéma

- Algebraické vyjadrenie binárneho zápisu:

$$13 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

môžeme previesť do Hornerovej schémy:

$$13 = (((0 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1$$

- Všeobecne:

$$\sum_{i=0}^k d_i \cdot 2^{i-1} = (((\dots (0 \cdot 2 + d_k) \dots) \cdot 2 + d_1) \cdot 2 + d_0$$

## 8.1. Binárne konštruktory

### IV.7 Binárne konštruktory

---

- Zadefinujme

$$S_0(n) = 2 \cdot n + 0$$

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

- Potom môžeme zapísať

$$13 = 01101$$

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

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

$$= S_1 S_0((0 \cdot 2 + 1) \cdot 2 + 1)$$

$$= S_1 S_0 S_1(0 \cdot 2 + 1)$$

$$= S_1 S_0 S_1 S_1(0)$$

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

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

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

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

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

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

- Takže:

$$S_0(n) = n0$$

$$S_1(n) = n1$$

- Funkcie  $S_0$  a  $S_1$  nazývame *binárne konštruktory*

## 8.2. Binárna diskriminácia

### IV.8 Testovanie binárneho zápisu

---

Dané je číslo  $n = 0d_k \dots d_2 d_1 d_0 = \sum_{i=0}^k d_i \cdot 2^i$ ,  $0 \leq d_i < 2$ .

1. Ako potom určíme najmenej významnú číslicu čísla  $n$ ?

$$\blacktriangleright \text{Lsb}(n) = d_0 = n \bmod 2$$

2. Ako určíme číslo, ktoré vznikne z  $n$  odobratím najmenej významnej číslice?

$$\blacktriangleright \text{Msbs}(n) = 0d_k \dots d_1 = \sum_{i=1}^k d_i \cdot 2^i = n \div 2$$

Naprogramujme zmenu najmenej významnej binárnej číslice:

$$\blacktriangleright \text{Flip\_lsb}(n) = 2 * \text{Msbs}(n) + (1 \div \text{Lsb}(n))$$

V CL existuje názornejší spôsob...

### IV.9 Binárna diskriminácia

---

CL pozná *binárnu diskrimináciu*:

$\blacktriangleright$  Každé číslo sa dá rozdeliť na najmenej významnú binárnu číslicu a významnejší zvyšok  $m$

$$\dots \leftarrow \tau = m\mathbf{0}$$

$$\dots \leftarrow \tau = S_0(m)$$

$$\dots \leftarrow \tau = m\mathbf{1}$$

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

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

Tieto prípady pokrývajú všetky možnosti

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

a sú vzájomne výlučné:

$$\neg(\exists m(n = m\mathbf{0}) \wedge \exists m(n = m\mathbf{1}))$$

#### IV.10 Použitie binárnej diskriminácie

v tele klauzuly

$$\text{Flip\_lsb}(n) = m1 \leftarrow n = m0$$

$$\text{Flip\_lsb}(n) = m0 \leftarrow n = m1$$

v argumente funkcie

$$\text{Flip\_lsb}(m0) = m1$$

$$\text{Flip\_lsb}(m1) = m0$$

Počítajme:

$$\text{Flip\_lsb}(5) = \text{Flip\_lsb}(21) = 20 = 4$$

$$\text{Flip\_lsb}(6) = \text{Flip\_lsb}(30) = 31 = 7$$

### 8.3. Binárna rekurgia

#### IV.11 Počet číslic binárneho zápisu

Naprogramujme  $\text{Len}(n)$  – počet binárnych číslic čísla  $n$  pomocou binárnej diskriminácie:

$$\text{Len}(n0) = 1 + \text{Len}(n)$$

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

Počítajme:

$$\text{Len}(5) = \text{Len}(21) = 1 + \text{Len}(2) = 1 + \text{Len}(10) =$$

$$= 1 + 1 + \text{Len}(1) = 1 + 1 + \text{Len}(01) =$$

$$= 1 + 1 + 1 + \text{Len}(0) = 1 + 1 + 1 + \text{Len}(00) =$$

$$= 1 + 1 + 1 + 1 + \text{Len}(0) = 1 + 1 + 1 + 1 + \text{Len}(00) = \dots$$

S nulou opatrne a s nepresnými zadaniami tiež!

#### IV.12 Binárna rekurgia

*Binárna rekurgia* – argument rekurzívneho volania je pôvodný argument *bez* poslednej (najmenej významnej) binárnej číslice:

$$F(n, \dots) = \dots \leftarrow n = m0 \wedge m = 0$$

$$F(n, \dots) = \dots F(m, \dots) \dots \leftarrow n = m0 \wedge m > 0$$

$$F(n, \dots) = \dots F(m, \dots) \dots \leftarrow n = m1$$

alebo s dosadením:

$$\begin{aligned}
F(m0, \dots) &= \dots && \leftarrow m = 0 \\
F(m0, \dots) &= \dots F(m, \dots) \dots && \leftarrow m > 0 \\
F(m1, \dots) &= \dots F(m, \dots) \dots
\end{aligned}$$

Platí

$$m > 0 \rightarrow m < m0 = 2 \cdot m + 0, \quad m < m1 = 2 \cdot m + 1,$$

⇒ Binárna rekúzia je špeciálny prípad všeobecnej rekúzie

#### IV.13 Binárna rekúzia — príklad a dĺžka rekúzie

Naprogramujme  $\text{Len}(n)$  — najmenší počet binárnych konštruktorov potrebných na vytvorenie čísla  $n$  z čísla 0:

$$\begin{aligned}
\text{Len}(m0) &= 0 && \leftarrow m = 0 \\
\text{Len}(m0) &= 1 + \text{Len}(m) && \leftarrow m > 0 \\
\text{Len}(m1) &= 1 + \text{Len}(m)
\end{aligned}$$

Koľko krokov môže najviac mať binárna rekúzia pre  $n$ ?  $\text{Len}(n)$

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

## 8.4. Aritmetické operácie na binárnom zápise čísel

#### IV.14 Efektívna aritmetika s veľkými číslami

##### **Neefektívna definícia aritmetickej operácie:**

počet krokov = hodnota niektorého argumentu

Napríklad násobenie primitívnou rekúziou

$$\begin{aligned}
\text{Mul}(x, 0) &= 0 \\
\text{Mul}(x, y + 1) &= x + \text{Mul}(x, y)
\end{aligned}$$

##### **Efektívna definícia aritmetickej operácie:**

počet krokov = počet číslic (logaritmus) niektorého argumentu

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

#### IV.15 Nasledovník na binárnom zápise

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

$x$	0	01	1	01	2	010	3	011	4	0100	5	0101	6	0110	...
$x + 1$	1	01	2	010	3	011	4	0100	5	0101	6	0110	7	0111	...

Príklady:

$$\begin{array}{r}
 0 \\
 + 01 \\
 \hline
 01
 \end{array}
 \qquad
 \begin{array}{r}
 0100 = 4 \\
 + 01 \\
 \hline
 0101 = 5
 \end{array}
 \qquad
 \begin{array}{r}
 0101 = 5 \\
 + 01 \\
 \hline
 0110 \\
 + 010 \\
 \hline
 0110 = 6
 \end{array}$$

Zovšeobecnenie:

$$\begin{array}{r}
 0 \\
 + 01 \\
 \hline
 01
 \end{array}
 \qquad
 \begin{array}{r}
 x0 \\
 + 01 \\
 \hline
 x1
 \end{array}
 \qquad
 \begin{array}{r}
 x1 \\
 + 01 \\
 \hline
 x \\
 + 010 \\
 \hline
 (x + 1)0
 \end{array}$$

#### IV.16 Nasledovník na binárnom zápise

Algebraické odvodenie

- predpokladáme, že  $\text{Succ}(n)$  počíta  $n + 1$
- hľadáme vyjadrenie zamýšľaného výsledku iba použitím:
  - čísla 0
  - binárnych konštruktorov  $S_0(n) = n0$  a  $S_1(n) = n1$
  - rekurzie so zvyšnými číslicami binárneho zápisu

$$\text{Succ}(x0) = x0 + 1 = (2 \cdot x + 0) + 1 = 2 \cdot x + 1 = x1$$

$$\begin{aligned}
 \text{Succ}(x1) &= x1 + 1 = (2 \cdot x + 1) + 1 = 2 \cdot (x + 1) = (x + 1)0 \\
 &= (\text{Succ}(x))0
 \end{aligned}$$

Klauzálna definícia použitím binárnej rekurzie:

$$\begin{array}{l}
 \text{Succ}(x) = v1 \quad \leftarrow x = v0 \\
 \text{Succ}(x) = (\text{Succ}(v))0 \quad \leftarrow x = v1
 \end{array}
 \quad \Bigg| \quad
 \begin{array}{l}
 \text{Succ}(v0) = v1 \\
 \text{Succ}(v1) = (\text{Succ}(v))0
 \end{array}$$

#### IV.17 Sčítanie na binárnom zápise

---

Príklady sčítania:

$$\begin{array}{r} 0 \\ + 010 \\ \hline 010 \end{array}$$

$$\begin{array}{r} 011 \\ + 0110 \\ \hline 01001 \end{array}$$

$$\begin{array}{r} 011 \\ + 011 \\ \hline 0110 \end{array}$$

Zovšeobecnenie:

	$00$		$y > 0$	
	$+ y$		$\downarrow$	
	$y$			
$x > 0 \rightarrow$	$x0$		$x0$	$x0$
	$+ 00$	$+$	$y0$	$+ y1$
	$x0$		$(x + y)0$	$?$
	$x1$		$x1$	$x1$
	$+ 00$	$+$	$y0$	$+ y1$
	$x1$		$?$	$?$

#### IV.18 Sčítanie na binárnom zápise

---

Klauzálna definícia binárnou rekurziou s dvojitou binárnou diskrimináciou

Odvodíme podľa príkladu z predchádzajúcej snímky alebo algebraicky použitím:

- čísla 0, binárnych konštruktorov  $S_0(n) = n0$  a  $S_1(n) = n1$
- nasledovníka
- rekurzie so zvyšnými číslicami binárneho zápisu



$$\begin{aligned}
\text{Add}(x0, y0) &= 0 + y0 = y0 \quad \leftarrow x = 0 \\
\text{Add}(x0, y1) &= 0 + y1 = y1 \quad \leftarrow x = 0 \\
\text{Add}(x0, y0) &= x0 + y0 = x1 \quad \leftarrow x > 0 \wedge y = 0 \\
\text{Add}(x0, y0) &= x0 + y0 = (2 \cdot x + 0) + (2 \cdot y + 0) = 2 \cdot (x + y) = \\
&= (x + y)0 = (\text{Add}(x, y))0 \quad \leftarrow x > 0 \wedge y > 0 \\
\text{Add}(x0, y1) &= x1 + y0 = (2 \cdot x + 0) + (2 \cdot y + 1) = 2 \cdot (x + y) + 1 = \\
&= (x + y)1 = (\text{Add}(x, y))1 \quad \leftarrow x > 0 \\
\text{Add}(x1, y0) &= x1 + 0 = \dots \quad \leftarrow y = 0 \\
\text{Add}(x1, y0) &= x1 + y0 = \dots \text{Add}(x, y) \dots \quad \leftarrow y > 0 \\
\text{Add}(x1, y1) &= x1 + y1 = \dots \text{Add}(x, y) \dots
\end{aligned}$$

#### IV.19 Násobenie na binárnom zápise

---

Algebraické odvodenie:

$$\begin{aligned}
\text{Mul}(x0, y) &= 0 \cdot y = 0 \quad \leftarrow x = 0 \\
\text{Mul}(x0, y) &= x0 \cdot y = (2 \cdot x + 0) \cdot y = 2 \cdot x \cdot y = \\
&= \dots \text{Mul}(x, y) \dots \quad \leftarrow x > 0 \\
\text{Mul}(x1, y) &= x1 \cdot y = (2 \cdot x + 1) \cdot y = \dots \text{Mul}(x, y) \dots
\end{aligned}$$

## 9. Párovanie

### 9.1. Kódovanie dátových štruktúr

#### IV.20 Dátové štruktúry

---

- Programy zriedka operujú iba na číslach
- Zvyčajne používame dátové štruktúry, napríklad ...
- Funkcie v CL pracujú iba s prirodzenými číslami (s ľubovoľnou presnosťou)
- Najjednoduchšia dátová štruktúra: *usporiadaná dvojica*
  - Umožňuje vybudovať väčšinu prakticky potrebných dátových štruktúr (uvidíme neskôr)

- Ako zakódujeme usporiadané dvojice v CL?

## 9.2. Párovacie funkcie

### IV.21 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 None
 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

### IV.22 Párovacie funkcie

---

**Definícia 21.** *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:

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

- 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	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

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

←  $x+y=d$

– 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	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

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

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

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

←  $x+y \neq 0$

### 9.2.1. Párovacia funkcia CL

#### IV.24 Párovacia funkcia CL

---

Párovacia funkcia zabudovaná v CL:

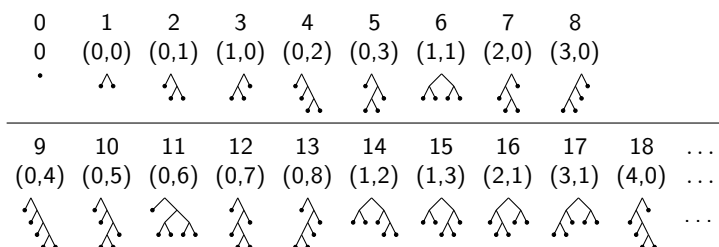
- Zapisuje sa binárnym operátorom , (čiarka)
- Založená na očíslovaní binárnych stromov
- Na aplikáciu párovacej funkcie sa pozeráme ako na vytvorenie stromu z dvoch podstromov
- Požadujeme, aby  $t < s$  práve vtedy, keď
  - $t$  má menej uzlov ako  $s$ , alebo

- $t = x, y$  má rovnaký počet uzlov ako  $s = u, v$ 
  - \*  $x < u$  alebo
  - \*  $x = u$  a  $y < v$ .

#### IV.25 Párovacia funkcia CL

---

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



- Fragment tabuľky párovacej funkcie (,):

$x \setminus y$	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	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

#### IV.26 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:

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

#### IV.27 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:

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

### 9.2.2. Programovanie s párovacou funkciou

#### IV.28 Programovanie s párovacou funkciou

---

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

$$\begin{array}{l} \dots \leftarrow \tau = 0 \\ \dots \leftarrow \tau = u, v \end{array}$$

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

#### IV.29 Programovanie s párovacou funkciou — príklad

---

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

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

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

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

- Prípád v druhej klauzule *nemôže* nastať

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

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

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

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

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

#### IV.30 Programovanie s párovacou funkciou — výpočet

---

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

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

- 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

### 9.2.3. Tupling

#### IV.31 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:

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

spojíme do jednej funkcie

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

#### IV.32 Tupling

---

- Spojená funkcia:

$$\text{Divmod}(x, y) = 0, 0 \leftarrow y = 0$$

$$\text{Divmod}(x, y) = 0, x \leftarrow y \neq 0 \wedge x < y$$

$$\text{Divmod}(x, y) = q + 1, r \leftarrow y \neq 0 \wedge x \geq y \wedge \text{Divmod}(x \dot{-} y, y) = q, r$$



## V. prednáška

# Zoznamy.

# Chvostová rekúzia a tupling na zoznamoch

17. marca 2014

## 10. Zoznamy

### V.1 Opakovanie

---

Párovacia funkcia:

- Operátor  $(\cdot, \cdot)$  (čiarka)
  - najnižšia priorita:  $1 + 2, 3 + 4 \equiv ((1 + 2), (3 + 4))$
  - implicitne sa zátvorkuje doprava:  $1, 2, 3, 4 \equiv 1, (2, (3, 4))$
- Vlastnosti:
  - injektívnosť:  $(x, y) = (u, v) \rightarrow x = u \wedge y = v$
  - surjektívnosť z  $\mathbb{N} \times \mathbb{N}$  na  $\mathbb{N} \setminus \{0\}$ :  $x = 0 \vee \exists u \exists v (x = (u, v))$
  - pár je väčší ako jeho zložky:  $x < (x, y) \wedge y < (x, y)$
- Párová diskriminácia:

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

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

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

- Párovacia funkcia sa počíta „lenivo“
  - iba keď je naozaj požadovaná číselná hodnota
  - inak si CL pamätá pôvodnú dvojicu čísel

## 10.1. Kódovanie konečných postupností

### V.2 Kódovanie konečných postupností

---

- Jedna z najbežnejších úloh: spracovať postupnosť dát
  - Napríklad spočítať priemerný počet bodov z testu
- Ako zakódujeme postupnosti *ľubovoľných čísel ľubovoľnej dĺžky* (vopred neznámej)?
  - Napríklad postupnosť *8, 24, 0, 14, 30, 6* alebo postupnosť *1, 3, 5, 7*

### V.3 Kódovanie konečných postupností

---

- Ak vopred poznáme spracovávané čísla, mohli by sme ich postupnosť zakódovať v  $m$ -árnej sústave, pre  $m$  väčšie ako všetky čísla
  - Napr. z testu dá získať najviac 30 bodov  $\implies m \geq 31$
- Kódovanie postupnosti  $x_1, x_2, x_3, \dots, x_n$ :

$$x_n \cdot m^{n-1} + \dots + x_3 \cdot m^2 + x_2 \cdot m^1 + x_1 \cdot m^0$$

- Napríklad kódom postupnosti *8, 24, 0, 14, 30, 6* by bolo číslo

$$6 \cdot 31^5 + 30 \cdot 31^4 + \dots + 14 \cdot 31^3 + 0 \cdot 31^2 + 24 \cdot 31^1 + 8 \cdot 31^0$$

- Problematické: vopred známe maximum, koncové nuly

### V.4 Kódovanie konečných postupností

---

- Skúsme využiť vnorené párovanie:

$$x = 8, 2, 0, 4, 6 = 8, (2, (0, (4, 6))) = 7798222$$

$$y = 1, 3, 5, 7 = 1, (3, (5, 7)) = 148217$$

- Dekódujme teraz postupnosť zakódovanú číslom  $x$

Dĺžku nepoznáme vopred

$$x = a_1, x_1 \rightsquigarrow a_1 = 8 \quad x_1 = 2, 0, 4, 6 = 45576$$

$$x_1 = a_2, x_2 \rightsquigarrow a_2 = 2 \quad x_2 = 0, 4, 6 = 830$$

$$x_2 = a_3, x_3 \rightsquigarrow a_3 = 0 \quad x_3 = 4, 6 = 401$$

$$x_3 = a_4, x_4 \rightsquigarrow a_4 = 4 \quad x_4 = 6$$

$$x_4 = a_5, x_5 \rightsquigarrow a_5 = 1 \quad x_5 = 1$$

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

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

#### V.5 Kódovanie konečných postupností

---

- Nie každé číslo kóduje dvojicu:

0

- Nie každé číslo kódujúce dvojicu kóduje trojicu:

$x_1, 0$

⋮

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

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

#### V.6 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 označujúce zoznamy majú príponu -s

$xs$      $ys$      $zs$     ...

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

## 10.2. Programovanie so zoznamami

### 10.2.1. Zreťazenie

#### V.7 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?

#### V.8 Programovanie so zoznamami: Zreťazenie

---

Ako hľadať riešenie?

1. Načrtne si príklad

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

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

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

## V.9 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:

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

- Zretazenie je zabudované do CL
- Operátor  $xs \oplus ys$  ( $xs ++ ys$ )

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

- Vyššia priorita ako párovanie:

$$xs \oplus a, ys \equiv (xs \oplus a), ys \quad (\dagger\dagger\dagger)$$

$$xs \oplus (a, ys) \quad (\text{OK})$$

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

Zretazovať môžeme *iba zoznamy!*

## 10.2.2. Porovnanie s Pythonom

- Ako by sme naprogramovali zretazenie

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

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

v Pythone?

- Napríklad takto:

```
def conc(xs, ys):
    if xs is not None:
        x = xs.data
        zs = xs.next
        vs = conc(zs, ys)
        return Vrchol(x, vs)
    else:
        return ys
```

### 10.2.3. Obrátenie

#### V.12 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 zreťazení:

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

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

$xs$

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

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

$zs$

upravíme na výsledok pre *celý zoznam*  $xs$

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

$x$                    $zs$                                    $\text{Rev}(zs)$                    $x$

## 10.3. Chvostová rekúzia na zoznamoch

### 10.3.1. Súčet prvkov zoznamu

#### V.13 Cykly na zoznamoch

---

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

```

def suml(xs):
    s = 0
    while xs is not None:
        x, zs = xs.data, xs.next
        s = s + x
        xs = zs
    return s

```

- V deklaratívnom programovaní úlohu cyklov plní chvostová rekúzia

#### V.14 Chvostová rekúzia na zoznamoch

---

- Chvostová rekúzia simulujúca while cyklus pre súčet prvkov zoznamu:

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

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

- Inicializácia:

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

- S dosadením do argumentov:

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

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

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

### 10.3.2. Obrátenie zoznamu

#### V.15 Neefektívita obrátenia zoznamu zrežovaním

---

- 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:

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


---

$3+2+1+0 = 6$

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

#### V.16 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?

#### V.17 Efektívne obrátenie zoznamu v CL

---

- Efektívne obrátenie zoznamu while cyklom:

```

rs = None
while xs is not None:
    x, zs = xs.data, xs.next
    rs = Vrchol(x, rs);
    xs = zs
return rs

```

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

$$\text{While\_rev}(0, rs) = rs$$

$$\text{While\_rev}(\underline{(x, xs)}, rs) = \text{While\_rev}(xs, \underline{(x, rs)})$$

$$\text{Rev}(xs) = \text{While\_rev}(xs, 0)$$

## 10.4. Tupling zoznamových operácií

### 10.4.1. Split = Take, Drop

V.18 Príklad – L, Take, Drop

---

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

$$i \leq L(xs) \rightarrow$$

$$\text{Take}(i, xs) = ys \leftrightarrow L(ys) = i \wedge \exists zs (xs = ys \oplus zs)$$

$$i > L(xs) \rightarrow \text{Take}(i, xs) = xs$$

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

$$i \leq L(xs) \rightarrow$$

$$\text{Drop}(i, xs) = zs \leftrightarrow \exists ys (L(ys) = i \wedge xs = ys \oplus zs)$$

$$i > L(xs) \rightarrow \text{Drop}(i, xs) = 0$$



- Operácie

$$\text{Take}(0, xs) = []$$

$$\text{Take}(i + 1, []) = []$$

$$\text{Take}(i + 1, (x, xs)) = x, \text{Take}(i, xs)$$

$$\text{Drop}(0, xs) = xs$$

$$\text{Drop}(i + 1, []) = []$$

$$\text{Drop}(i + 1, (x, xs)) = \text{Drop}(i, xs)$$

majú rovnakú štruktúru diskriminácií a rekurzie

- Ak potrebujeme výsledky oboch  $\implies$  dva prechody zoznamom
- Chceme spojenú jednoprechodovú funkciu — [tupling<sup>†9.2.3</sup>](#)

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

- Príklad s náčrtom rekurzie:

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

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

### 10.4.2. Zip, Unzip

- Zip — spojenie dvoch zoznamov rovnakej dĺžky do zoznamu dvojíc

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

- Unzip — rozdelenie zoznamu dvojíc na dva zoznamy

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

- 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 rekurzie:

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

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

## VI. prednáška

# Triedenie zoznamov

## Zoznamová reprezentácia množín

24. marca 2014

### 11. Predikáty

#### VI.1 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  $x$

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

- číslo  $x$  kóduje trojicu čísel

$$\text{Triple}(x) \leftrightarrow \exists u \exists v \exists w(x = u, v, w)$$

- $xs$  je prázdny zoznam

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

- zoznam  $xs$  je palindróm

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

...

## VI.2 Prvok zoznamu

---

- Kedy je číslo  $a$  prvkom zoznamu  $xs$  ( $a \in xs$ )?
- Práve vtedy, keď sa zoznam  $xs$  dá rozdeliť na časti  $ys$  a  $zs$ , medzi ktorými sa nachádza prvok  $a$ , teda

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

- Alternatívne: Práve vtedy, keď sa  $a$  vyskytuje ako  $(i + 1)$ -ý prvok zoznamu  $xs$  pre nejaké  $i$ :

$$a \in xs \leftrightarrow \exists i (xs[i] = a)$$

## VI.3 Charakteristické funkcie

---

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

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

*Príklad 22.*  $\text{Palindrome}_*(xs) = 1 \leftarrow x = \text{Rev}(xs)$

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

*Príklad 23.*  $\text{Member}_*(a, 0) = 0$

$\text{Member}_*(a, (x, xs)) = 1 \leftarrow a = x$

$\text{Member}_*(a, (x, xs)) = \text{Member}_*(a, xs) \leftarrow a \neq x$

## VI.4 Definovanie predikátov v CL

---

V CL môžeme predikáty definovať priamo

- podobne ako funkcie
- *vynechávame* klauzuly pre prípady, v ktorých predikát *neplatí*

*Príklad 24.*  $\text{Palindrome}(x) \leftarrow x = \text{Rev}(x)$

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

## VI.5 Definovanie predikátov v CL –diskriminácia a rekurgia

Ak sa potrebujeme odvolať na iný predikát, alebo na rekurzívne volanie predikátu, použijeme *diskrimináciu na platnosť predikátu*:

$$\begin{aligned}\dots &\leftarrow P(x) \\ \dots &\leftarrow \neg P(x)\end{aligned}$$

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

Príklad 25.  ~~$\neg \text{Member}(a, 0)$~~

$$\text{Member}(a, (x, xs)) \leftarrow x = a$$

$$\text{Member}(a, (x, xs)) \leftarrow x \neq a \wedge \text{Member}(a, xs)$$

~~$$\text{Member}(a, (x, xs)) \leftarrow x \neq a \wedge \neg \text{Member}(y, z)$$~~

Predikát  $a \in xs$  je zabudovaný v CL (syntax:  $a \text{ in } xs$ ; syntax negácie  $a \notin xs$ :  $a \text{ !in } xs$ )

## 12. Triedenie zoznamov

### 12.1. Špecifikácia triedenia

#### VI.6 Špecifikácia triedenia

**Definícia 26.** Funkcia  $\text{Sort}(xs) = ys$  *utriedi* zoznam  $xs$ , ak súčasne platí:

1. zoznam  $ys$  obsahuje všetky prvky zoznamu  $xs$  vrátane prípadných opakovaní, čiže  $ys$  je *permutáciou*  $xs$
2. zoznam  $ys$  je usporiadaný od najmenšieho prvku po najväčší

Formalizácia špecifikácie:

1.  $\text{Sort}(xs) \sim xs$
2.  $\text{Ord}(\text{Sort}(xs))$

## VI.7 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

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

- Samozrejme stačí porovnávať iba počty výskytov tých prvkov, ktoré sa nachádzajú v aspoň jednom zozname:

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

alternatívne

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

alternatívne

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

## VI.8 Permutácia zoznamu – rekurzívny predikát

---

Ako vypočítame  $xs \sim ys$  rekurziou?

1. Rekurzívny výpočet  $\#_a(xs)$

$$\begin{aligned} \#_a(0) &= 0 \\ \#_a(x, xs) &= \#_a(xs) + 1 \leftarrow a = x \\ \#_a(x, xs) &= \#_a(xs) \quad \leftarrow a \neq x \end{aligned}$$

2. Pomocný predikát  $\text{Eq\_counts}(zs, xs, ys)$  – každý prvok zo  $zs$  sa nachádza v  $xs$  rovnaký početkrát ako v  $ys$

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

Naprogramujeme zoznamovou rekurziou na  $zs$



Eq\_counts( 0, xs, ys)  
 Eq\_counts((z, zs), xs, ys) ← ?

3.  $xs \sim ys$  naprogramujeme bez rekurzie použitím Eq\_counts:

$xs \sim ys \leftarrow ?$

### VI.9 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 zretazením:

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

- Stačí porovnávať susediace prvky:

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

### VI.10 Usporiadaný zoznam — rekurzívny predikát

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

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

Ord(0 )  
 Ord(a, 0 )  
 Ord(a, b, xs) ←  $a \leq b \wedge \text{Ord}(b, xs)$

### VI.11 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

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

### VI.12 Algoritmy triedenia zoznamov

---

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

## 12.2. Triedenie vsúvaním

### VI.13 Triedenie vsúvaním (insertion sort)

---

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

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



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

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

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

### 12.3. 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  $\text{Ins}$  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  $\sum_{i=0}^{L(xs)} (i+L(ys))/2 \approx (L(xs) + L(ys))^2$  (ako  $\text{Isort}$ )

- ▶ Vôbec sme nevyužili, že  $xs$  je utriedený

#### VI.17 Zlúčenie utriedených zoznamov efektívne

---

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

$xs$	$ys$	$Merge(xs, ys)$
2, 3, 5, 0	<u>0</u> , 1, 7, 0	0,
2, 3, 5, 0	<u>1</u> , 7, 0	1,
<u>2</u> , 3, 5, 0	7, 0	2,
<u>3</u> , 5, 0	7, 0	3,
<u>5</u> , 0	7, 0	5,
0	7, 0	7, 0

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

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

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

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

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

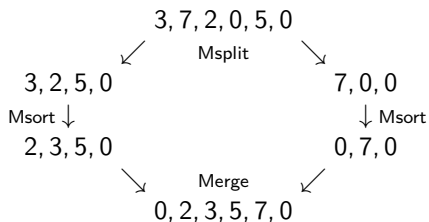
## 12.4. Triedenie zlučováním

### VI.18 Triedenie zlučováním (merge sort)

---

- Efektívny algoritmus, metóda rozdeľuj a panuj
- Zoznam rozdelíme na dve približne rovnako dlhé časti
  - ▶ Funkcia Msplit
- Časti rekurzívne utriedime
- Utriedené časti efektívne zlúčime
  - ▶ Funkcia Merge

- Náčrt Msort(3, 7, 2, 0, 5, 0):



- Kľúčová vlastnosť (pozor na triviálne prípady!):

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

#### VI.19 Triedenie zlučovaním – rozdelenie

- 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:

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

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

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

## 13. Zoznamová reprezentácia konečných množín

### VI.20 Konečné množiny

---

- 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

### VI.21 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

$$\dots \leftarrow s < t$$

$$\dots \leftarrow s = t$$

$$\dots \leftarrow s > t$$

## VI.22 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  $\text{Member}(a, xs) \equiv (a \in xs)$

$$\begin{aligned} & \cancel{a \notin 0} \\ & \cancel{a \notin x, xs \leftarrow a < x} \\ & a \in x, xs \leftarrow a = x \\ & a \in x, xs \leftarrow a > x \wedge a \in xs \\ & \cancel{a \notin x, xs \leftarrow a > x \wedge a \notin xs} \end{aligned}$$

- Platí extenzionalita

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

## VI.23 Operácie na množinách

---

- 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

$$\begin{aligned} 0 \cup ys &= ? \\ (x, xs) \cup 0 &= ? \\ (x, xs) \cup (y, ys) &= ? \leftarrow x < y \\ (x, xs) \cup (y, ys) &= ? \leftarrow x = y \\ (x, xs) \cup (y, ys) &= ? \leftarrow x > y \end{aligned}$$

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



## VII. prednáška

# Kombinatorické funkcie na zoznamoch

31. marca 2014

### VII.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*

# 14. Kombinatorické funkcie na zoznamoch

## VII.2 Kombinatorické funkcie na zoznamoch

---

- Daný je binárny predikát na zoznamoch  $R(xs, ys)$
- Hľadáme funkciu  $F$ , ktorá pre daný zoznam  $xs$  skonštruuje zoznam  $yss$  všetkých zoznamov  $ys$ , pre ktoré platí  $R(xs, ys)$ :

$$Rs(xs) = yss \leftrightarrow \forall ys(ys \in yss \leftrightarrow R(xs, ys))$$

alebo ekvivalentne stručnejšie:

$$ys \in Rs(xs) \leftrightarrow R(xs, ys)$$

- Postup:
  1. Vyšetříme platnosť  $R(xs, ys)$  pre prípady prázdneho a neprázdneho zoznamu  $xs$
  2. Zadefinujeme funkciu  $Rs$  zoznamovou rekurziou:

$$Rs(0) = \dots$$

$$Rs(x, xs) = \dots Rs(xs) \dots$$

## 14.1. Súvislé úseky

### 14.1.1. Sufixy

#### VII.3 Sufix

---

- Sufix — koncový úsek zoznamu

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

- Napríklad

$$(7, 11, 0) \sqsupseteq (7, 11, 0) \tag{1}$$

$$(2, 3, 5, 7, 11, 0) \sqsupseteq (7, 11, 0) \tag{2}$$

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

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

- Všimnite si:

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

#### VII.4 Sufix — analýza prípadov

---

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

- $xs \sqsupseteq zs$  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

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

#### VII.5 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:

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

- Všimnite si:

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

- Využime analýzu prípadov pre  $xs \sqsupseteq zs$

## VII.6 Všetky sufixy — odvodenie riešenia

---

$$\begin{aligned}zs \in \text{Suffixes}(xs) &\leftrightarrow xs \sqsupseteq zs & 0 \sqsupseteq zs &\leftrightarrow zs = 0 \\a \in (x, xs) &\leftrightarrow a = x \vee a \in xs & (x, xs) \sqsupseteq zs &\leftrightarrow zs = (x, xs) \vee xs \sqsupseteq zs\end{aligned}$$

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

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

- ▶  $\text{Suffixes}(0) = 0, 0$
- Sufixami neprázdneho zoznamu  $(x, xs)$  sú
  - \* zoznam  $(x, xs)$  sám, a
  - \* všetky sufixy zoznamu  $xs$ ,

teda

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

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

## 14.1.2. Prefixy

### VII.7 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:

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

- Rozoberme prípady pre  $xs$

## VII.8 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

## VII.9 Všetky prefixy

---

- Naprogramujeme 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:

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

- Využime analýzu prípadov pre  $ys \sqsubseteq xs$

## VII.10 Všetky prefixy — odvodenie riešenia

---

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

- Prefixom prázdneho zoznamu je iba prázdny zoznam:

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

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

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

- Potrebujeme pomocnú funkciu ???

## VII.11 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  $vss$

### 14.1.3. Segmenty

## VII.12 Segment

---

- Segment — všeobecný súvislý úsek zoznamu

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

- Napríklad

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

- Všimnite si:

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

- Čo vieme povedať o segmentoch  $us$  prázdneho a neprázdneho  $xs$ ?
- Ktorý z predchádzajúcich predikátov nám pomôže?

## 14.2. Podpostupnosti

### VII.13 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:

$$\begin{aligned} \text{Subseq}(ys, xs) &\equiv (ys \triangleleft xs) \leftrightarrow \\ &\exists is(\text{Set}(is) \wedge L(is) = L(ys) \wedge \\ &\quad \forall i(i < L(is) \rightarrow is[i] < L(xs) \wedge ys[i] = xs[is[i]])) \end{aligned}$$

- Neformálne:

- $ys$  vznikne vynechaním niektorých prvkov zo zoznamu  $xs$
- vzájomné poradie zostávajúcich prvkov sa nezmení

- Napríklad:

$$\begin{aligned} xs &= 3, 5, 2, 11, 17, 7, 13, 0 \\ ys &= 5, 2, \quad 7, \quad 0 \end{aligned}$$

### VII.14 Podpostupnosť – analýza prípadov

---

Podpostupnosť môžeme zadefinovať 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

$$ys \triangleleft (x, xs) \leftrightarrow \exists zs (ys = (x, zs) \wedge zs \triangleleft xs) \vee ys \triangleleft xs$$

### VII.15 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:

$$a \in xs \oplus ys \leftrightarrow a \in xs \vee a \in ys$$

### VII.16 Všetky podpostupnosti — odvodenie riešenia

---

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:

$$\text{Subseqs}('ak') = 'ak', 'a', 'k', '', 0$$

$$\text{Subseqs}('tak') = ('tak', 'ta', 'tk', 't', 0) \quad \text{a)}$$

$$\oplus ('ak', 'a', 'k', '', 0) \quad \text{b)}$$

## 14.3. Permutácie

### VII.17 Permutácia

---

- Permutáciu zoznamu môžeme zdefinovať aj inak ako

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

- Potrebujeme pomocný pojem — vsunutie:

$$\begin{aligned} \text{Insertion}(xs, ys, a) &\equiv (xs \approx ys[\downarrow a]) \leftrightarrow \\ &\exists us \exists vs (xs = us \oplus (a, vs) \wedge ys = us \oplus vs) \end{aligned}$$

- 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  $xs$ :

$$(x, xs) \sim ys \leftrightarrow \exists zs (ys \approx zs[\downarrow x] \wedge xs \sim zs)$$

### VII.18 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  $xs$ :

$$\begin{aligned} ys \approx (x, xs)[\downarrow a] &\leftrightarrow \\ ys = (a, x, xs) \vee \exists zs (ys = (x, zs) \wedge zs \approx xs[\downarrow a]) \end{aligned}$$

## 14.4. Ďalšie úlohy

### VII.19 Ďalšie kombinatorické úlohy

---

Premyslite si:

- $k$ -prvkové podpostupnosti,
- variácie (ako permutácie, ale niektoré prvky možno vynechať)
- $k$ -prvkové variácie,
- [segmenty](#)<sup>↑14.1.3</sup>

# VIII. prednáška

## Binárne stromy

7. apríla 2014

### 15. Binárne stromy

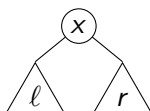
#### 15.1. Kódovanie stromov párovacími konštruktormi

##### VIII.1 Binárne stromy

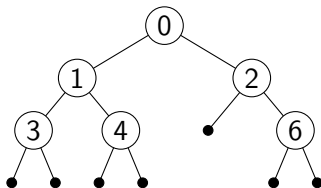
---

**Definícia 27.** Binárnym stromom je:

- prázdny strom: •
- vnútorný vrchol s hodnotou  $x$  a dvoma deťmi  $l$  a  $r$ , ak  $l$  a  $r$  sú binárnymi stromami



Príklad 28.



##### VIII.2 Párovacie konštruktory

---

- Binárne stromy by sme v CL mohli zakódovať párovaním:
  - Prázdny strom: 0
  - Uzol s hodnotou  $x$  a deťmi  $l$  a  $r$ : trojica  $(x, l, r)$

- Použijeme všeobecnejší 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 konštruktov s rôznymi tagmi

$$m \neq n \rightarrow (m, x) \neq (n, y)$$

⇒ Konštruktory s rôznymi tagmi sú *diskriminovatelné*

### VIII.3 Kódovanie binárných 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:

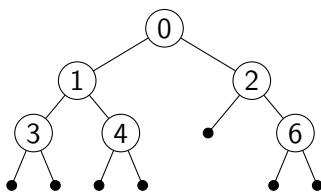
$$\begin{array}{ll} \dots \leftarrow t = E & \dots \leftarrow t = \bullet \\ \dots \leftarrow t = Nd(x, \ell, r) & \dots \leftarrow t = \frac{x}{\ell|r} \end{array}$$

Po dosadení do argumentov:

$$\begin{array}{ll} F(E) = \dots & F(\bullet) = \dots \\ F(Nd(x, \ell, r)) = \dots & F\left(\frac{x}{\ell|r}\right) = \dots \end{array}$$

### VIII.4 Kódovanie binárnych stromov — príklad

Napríklad strom



zakódujeme

$$\begin{array}{l}
 \text{Nd}(0, \\
 \quad \text{Nd}(1, \\
 \quad \quad \text{Nd}(3, E, E), \\
 \quad \quad \text{Nd}(4, E, E)), \\
 \quad \text{Nd}(2, \\
 \quad \quad E, \\
 \quad \quad \text{Nd}(6, E, E)))
 \end{array}
 \equiv
 \begin{array}{c}
 \hline
 0 \\
 \hline
 \begin{array}{c|c}
 \hline
 1 & 2 \\
 \hline
 \begin{array}{c|c}
 \hline
 3 & 4 \\
 \hline
 \bullet & \bullet \\
 \hline
 \bullet & \bullet
 \end{array}
 &
 \begin{array}{c|c}
 \hline
 \bullet & 6 \\
 \hline
 \bullet & \bullet \\
 \hline
 \bullet & \bullet
 \end{array}
 \end{array}
 \end{array}$$

### VIII.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:

$$\begin{array}{l}
 \text{Bt}(E) \\
 \text{Bt}(\text{Nd}(x, \ell, r)) \leftarrow N(x) \wedge \text{Bt}(\ell) \wedge \text{Bt}(r) \\
 \equiv \\
 \text{Bt}(\bullet) \\
 \text{Bt}\left(\frac{x}{\ell \mid r}\right) \leftarrow N(x) \wedge \text{Bt}(\ell) \wedge \text{Bt}(r)
 \end{array}$$

- V CL takéto predikáty slúžia aj na formátovanie výsledkov query

## 15.2. Operácie na binárnych stromoch

### VIII.6 Rekurzia 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 vrcholu

$$\ell < \text{Nd}(x, \ell, r) \wedge r < \text{Nd}(x, \ell, r)$$

### VIII.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

### VIII.8 Operácie na binárnych stromoch

Veľkosť (počet vrcholov) binárneho stromu  $Sz(t) \equiv |t|_b$ :

$$Sz(E) = 0 \quad \equiv |\bullet|_b = 0$$

$$Sz Nd(x, \ell, r) = 1 + Sz(\ell) + Sz(r) \equiv \left| \frac{x}{\ell | r} \right|_b = 1 + |\ell|_b + |r|_b$$

Predikát „x je prvkom t“  $Inbt(a, t) \equiv a \varepsilon_b t$ :

$$a \varepsilon_b \frac{x}{\ell | r} \leftarrow a = x$$

$$a \varepsilon_b \frac{x}{\ell | r} \leftarrow a \neq x \wedge a \varepsilon_b \ell$$

$$a \varepsilon_b \frac{x}{\ell | r} \leftarrow a \neq x \wedge \neg(a \varepsilon_b \ell) \wedge a \varepsilon_b r$$

## 15.3. Prechody binárnymi stromami

### VIII.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  $Bt(t)$ ):

$$Inorder \left( \begin{array}{c} 0 \\ \hline \frac{1}{\frac{3}{\bullet | \bullet} | \frac{4}{\bullet | \bullet}} \quad \frac{2}{\bullet | \frac{6}{\bullet | \bullet}} \end{array} \right) = 3, 1, 4, 0, 2, 6, 0$$

$\nearrow ? \nwarrow$

$$Inorder \left( \frac{1}{\frac{3}{\bullet | \bullet} | \frac{4}{\bullet | \bullet}} \right) = 3, 1, 4, 0 \quad Inorder \left( \frac{2}{\bullet | \frac{6}{\bullet | \bullet}} \right) = 2, 6, 0$$

Ako skonštruujeme inorderový zoznam pre strom zo zoznamov pre jeho podstromy?



### VIII.10 Prechod binárnym stromom — výpočet

$$\begin{aligned}
 & \text{Inorder} \left( \frac{0}{\frac{1}{\frac{3}{\bullet} | \frac{4}{\bullet}} \mid \frac{2}{\bullet | \frac{6}{\bullet}}} \right) \\
 &= \text{Inorder} \left( \frac{1}{\frac{3}{\bullet} | \frac{4}{\bullet}} \right) \oplus \left( 0, \text{Inorder} \left( \frac{2}{\bullet | \frac{6}{\bullet}} \right) \right) \\
 &= \left( \text{Inorder} \left( \frac{3}{\bullet} \right) \oplus \left( 1, \text{Inorder} \left( \frac{4}{\bullet} \right) \right) \right) \\
 &\quad \oplus \left( 0, \text{Inorder}(\bullet) \oplus \left( 2, \text{Inorder} \left( \frac{6}{\bullet} \right) \right) \right) \\
 &= \dots \\
 &= ((0 \oplus (3, 0)) \oplus (1, 0 \oplus (4, 0))) \oplus (0, 0 \oplus (2, 0 \oplus (6, 0))) \\
 &= ((3, 0) \oplus (1, 4, 0)) \oplus (0, 2, 6, 0) \\
 &= (3, 1, 4, 0) \oplus (0, 2, 6, 0) \\
 &= 3, 1, 4, 0, 2, 6, 0
 \end{aligned}$$

### VIII.11 Efektívne prechody binárnymi stromami

- Výpočet Inorder je neefektívny — zreťazenie prechádza znova zoznam  $\text{Inorder}(\ell)$
- Zefektívnenie — odstránenie zreťazenia
- Trik — vhodné *zovšeobecnenie*:

Naprogramujme funkciu  $\text{Inordera}(t, as)$ , ktorá

– pripojí inorder. výpis stromu *pred* zoznam v akumulátore *as*

$$\text{Inordera}(t, as) = \text{Inorder}(t) \oplus as \quad (*)$$

– prvky pridáva na začiatok akumulátora párovaním  $(,)$ , po jednom, *bez* zreťazenia

- Program odvodíme z požadovanej vlastnosti  $(*)$

VIII.12 Efektívne prechody binárnymi stromami

$$\text{Inordera}(t, as) = \text{Inorder}(t) \oplus as \quad (*)$$

$$\begin{aligned} \text{Inordera} \left( \begin{array}{c|c} \overline{0} & \\ \hline \overline{1} & \overline{2} \\ \hline \overline{3} \mid \overline{4} & \overline{6} \\ \bullet \mid \bullet & \bullet \mid \bullet \end{array}, \frac{98, 99, 0}{as} \right) &= \overline{3, 1, 4, 0, 2, 6, 98, 99, 0} \\ &\quad \uparrow ? \\ \text{Inordera} \left( \begin{array}{c|c} \overline{1} & \\ \hline \overline{3} \mid \overline{4} & \\ \bullet \mid \bullet & \bullet \mid \bullet \end{array}, \frac{77, 78, 0}{as_1} \right) &= \overline{3, 1, 4, 77, 78, 0} \\ \text{Inordera} \left( \begin{array}{c|c} \overline{2} & \\ \hline \overline{6} & \\ \bullet \mid \bullet & \bullet \mid \bullet \end{array}, \frac{88, 89, 0}{as_2} \right) &= \overline{2, 6, 88, 89, 0} \end{aligned}$$

VIII.13 Efektívne prechody binárnymi stromami

$$\text{Inordera}(t, as) = \text{Inorder}(t) \oplus as \quad (*)$$

$$\begin{aligned} \text{Inordera}(\bullet, as) &\stackrel{(*)}{=} \text{Inorder}(\bullet) \oplus as = 0 \oplus as = as \\ \text{Inordera} \left( \frac{x}{\ell \mid r}, as \right) &\stackrel{(*)}{=} \text{Inorder} \left( \frac{x}{\ell \mid r} \right) \oplus as \\ &= (\text{Inorder}(\ell) \oplus (x, \text{Inorder}(r))) \oplus as \\ &= \text{Inorder}(\ell) \oplus ((x, \text{Inorder}(r)) \oplus as) \\ &= \text{Inorder}(\ell) \oplus (x, \underline{\text{Inorder}(r) \oplus as}) \\ &\stackrel{(*)}{=} \text{Inorder}(\ell) \oplus (x, \underline{\text{Inordera}(r, as)}) \\ &\stackrel{(*)}{=} \underline{\text{Inordera}(\ell, (x, \text{Inordera}(r, as)))} \end{aligned}$$

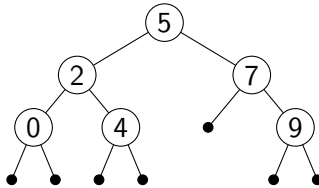
## 15.4. Binárne vyhľadávacie stromy

### VIII.14 Binárne vyhľadávacie stromy

---

**Definícia 29.** Binárny vyhľadávací strom je

- binárny strom;
- v každom vrchole 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.



### VIII.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

$$x \succ t \leftrightarrow \forall y (y \in_b t \rightarrow x > y)$$

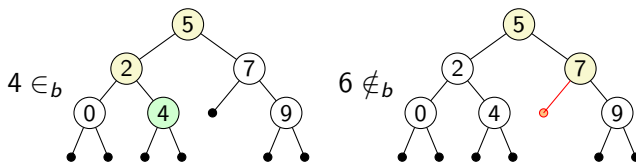
$$x \prec t \leftrightarrow \forall y (y \in_b t \rightarrow x < y)$$

### VIII.16 Využitie vyhľadávacej vlastnosti

---

- Binárne vyhľadávacie stromy reprezentujú množiny
  - ▶ Podobne ako ostro usporiadané zoznamy v 8. prednáške

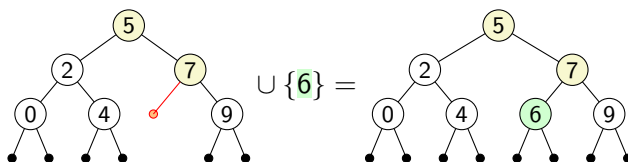
- Využitie vyhľadávacej vlastnosti pri predikáte „byť prvkom“  $a \in_b t$  pre  $t = \frac{x}{\ell|r}$ :
  - Ak  $a < x$ , prehľadávame iba  $\ell$ , lebo  $a$  nemôže byť v  $r$
  - Ak  $a > x$ , prehľadávame iba  $r$ , lebo  $a$  nemôže byť v  $\ell$
- ⇒ Prehľadávame *iba jednu cestu* stromom



### VIII.17 Vkladanie do vyhľadávacieho stromu

Princíp vkladania hodnoty do vyhľadávacieho stromu:

- Pokúsime sa nájsť vkladajú hodnotu
  - a) Hodnota sa v strome nenachádza (na jej mieste je **prázdny strom**)  
⇒ vyrobíme **nový uzol**



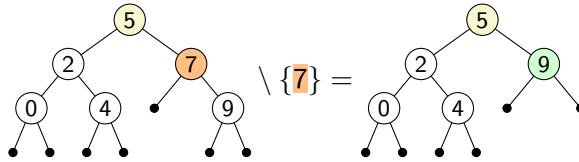
- b) Hodnota sa v strome nachádza ⇒ strom sa nemení
- Cestou naspäť strom **zrekonštruujeme**

### VIII.18 Zmazanie z vyhľadávacieho stromu I

Zmazanie hodnoty je komplikovanejšie ako vloženie

- Pokúsime sa nájsť zmazávanú hodnotu:
  - a) Hodnotu nenájdem ⇒ strom sa nezmení

- b) Hodnotu **nájdeme** a niektoré dieťa je prázdne  $\Rightarrow$  nahradíme vrchol **druhým dieťaťom**



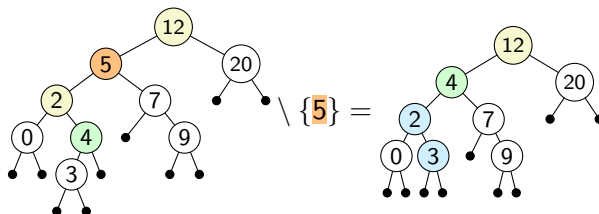
### VIII.19 Zmazanie z vyhľadávacieho stromu II

- Pokúsime sa nájsť zmazávanú hodnotu:

...

- c) Hodnotu **nájdeme** a obe deti sú neprázdne:

1. Nájdeme 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**

- 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 vrchole:

$$\text{Maxt}\left(\frac{x}{\ell|r}\right) = x \quad \leftarrow r = \bullet$$

$$\text{Maxt}\left(\frac{x}{\ell|r}\right) = \text{Maxt}(r) \quad \leftarrow r \neq \bullet$$

$$\text{Delmaxt}\left(\frac{x}{\ell|r}\right) = \ell \quad \leftarrow r = \bullet$$

$$\text{Delmaxt}\left(\frac{x}{\ell|r}\right) = \frac{x}{\ell|\text{Delmaxt}(r)} \quad \leftarrow r \neq \bullet$$

- Rovnaké diskriminácie a argumenty rekurzie  $\Rightarrow$  kandidáti na *tupling*

- 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:

$$\text{Extract\_max}\left(\frac{x}{\ell|r}\right) = ?, ? \leftarrow r = \bullet$$

$$\text{Extract\_max}\left(\frac{x}{\ell|r}\right) = ?, ? \leftarrow r \neq \bullet \wedge \text{Extract\_max}(?) = ?, ?$$

## IX. prednáška

# Číslovanie binárnych stromov

## Všeobecné stromy

13. apríla 2014

## 15. Binárne stromy

### 15.5. Úplné binárne stromy

#### IX.1 Úplné binárne stromy

---

**Definícia 30.** *Hĺbkou stromu* nazývame dĺžku najdlhšej cesty koreň–list:

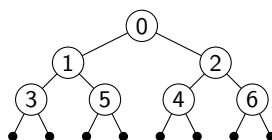
$$d(\bullet) = 0$$

$$d\left(\frac{x}{\ell|r}\right) = 1 + \max(d(\ell), d(r))$$

**Definícia 31.** Binárny strom nazveme *úplným*, ak

- na každej úrovni okrem poslednej sú všetky vrcholy neprázde,
- na poslednej úrovni sú iba prázdne stromy.

*Príklad 32.*



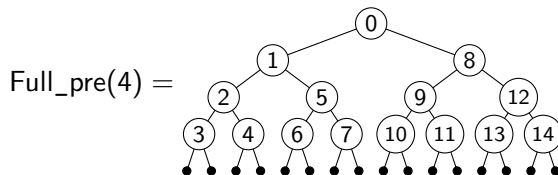
**Tvrdenie 33** (Vzťah veľkosti a hĺbky úplného stromu).

$$|t|_b = 2^{d(t)} - 1$$

#### IX.2 Číslovanie vrcholov v úplných stromoch I

---

- Číslovanie vrcholov úplného stromu číslami  $0, 1, 2, \dots$  v poradí preorder:



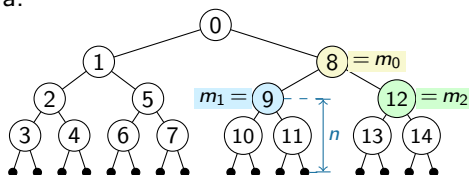
- Na nájdenie rekurzívneho riešenia musíme problém *zovšeobecniť*
- Zovšeobecnenie umožní číslať podstromy

### IX.3 Číslovanie vrcholov v úplných stromoch II

- Zovšeobecnenie — Full\_pre<sub>1</sub>( $n, m$ ):

Číslovanie vrcholov úplného stromu hĺbky  $n$  číslami  $m, m+1, m+2, \dots$  v poradí preorder

- Schéma riešenia:



Číslovanie v poradí *preorder*:

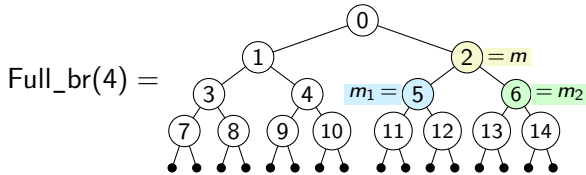
1. očísľujeme koreň podstromu  $m_0 = m$
2. očísľujeme ľavý podstrom od  $m_1 = m_0 + 1$
3. očísľujeme pravý podstrom od  $m_2 = m_1 + s$

- Aké je  $s$  pre úplný strom hĺbky  $n$ ?
- Ako sa číslovanie zmení pre inorder, postorder?



IX.4 Číslovanie úplných stromov do šírky

- Číslovanie vrcholov úplného stromu číslami 0, 1, 2, ... v poradí prechodu do šírky:

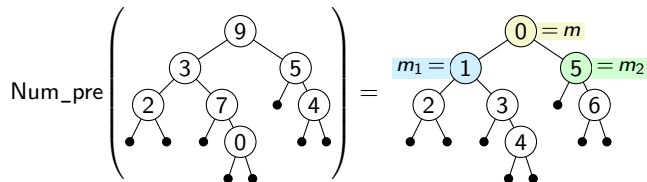


- Problém zovšeobecníme pre podstrom s koreňom s číslom  $m$
- Aký je vzťah medzi číslom vrcholu  $m$  a číslami jeho detí  $m_1$  a  $m_2$ ?

## 15.6. Číslovanie vrcholov v binárnych stromoch

IX.5 Číslovanie vrcholov v ľubovoľných stromoch

- Číslovanie vrcholov ľubovoľného stromu číslami 0, 1, 2, ... v poradí preorder:



- Rovnaký postup ako pri úplných stromoch:
  - Zovšeobecníme na číslovanie od  $m$  — Num\_pre<sub>1</sub>( $t, m$ )
  - Aký je rozdiel medzi číslom ľavého a pravého dieťaťa?
  - Rozdiel vieme počítať aj bez pomocnej funkcie — tupling

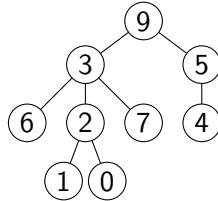
# 16. Všeobecné stromy

## 16.1. Kódovanie

### IX.6 Všeobecné stromy — kódovanie

---

Všeobecný strom — ľubovoľne veľa detí každého vrcholu:



Ako zakódujeme takýto strom v CL?

- Počet detí je premenlivý a neobmedzený
- Uložíme ich do zoznamu
- Stačí nám jeden párovací konštruktor:

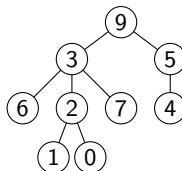
$$\text{Gnd}(x, ts) \equiv \frac{x}{ts} = 1, x, ts$$

$x$  — hodnota uložená vo vrchole stromu,  $ts$  — zoznam detí

- Ako zakódujeme strom na obrázku?

### IX.7 Všeobecné stromy — kódovanie

---



- Ako zakódujeme strom na obrázku?

$$\begin{aligned}
T_1 &= \text{Gnd}(9, (\text{Gnd}(3, (\text{Gnd}(6, 0), \\
&\quad \text{Gnd}(2, 0), \\
&\quad \text{Gnd}(7, (\text{Gnd}(1, 0), \text{Gnd}(0, 0), 0)), \\
&\quad 0)), \\
&\quad \text{Gnd}(5, (\text{Gnd}(4, 0), 0), \\
&\quad 0)) \\
&\quad 9 \\
&\equiv \frac{3}{\frac{6}{0}, \frac{2}{0}, \frac{7}{\frac{1}{0}, \frac{0}{0}}, 0}, \frac{5}{\frac{4}{0}, 0}, 0
\end{aligned}$$

## IX.8 Všeobecné stromy – dekodovanie

---

- Všeobecný strom dekodujeme diskrimináciou

$$\dots \leftarrow t = \frac{x}{ts} \quad \equiv \quad \dots \leftarrow t = \text{Gnd}(x, ts)$$

- Zoznam všeobecných stromov dekodujeme 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:

$$\begin{aligned}
F\left(\frac{x}{ts}\right) = \dots &\quad \equiv \quad F \text{Gnd}(x, ts) = \dots \\
F(0) = \dots &\quad \equiv \quad F(0) = \dots \\
F\left(\frac{x}{ts_1}, ts_2\right) = \dots &\quad \equiv \quad F(\text{Gnd}(x, ts_1), ts_2) = \dots
\end{aligned}$$

## IX.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*)

$$\begin{array}{l} \text{Lgt}(0) \\ \text{Lgt}(t, ts) \leftarrow \text{Gt}(t) \wedge \text{Lgt}(ts) \end{array} \quad \text{Gt}\left(\frac{x}{ts}\right) \leftarrow N(x) \wedge \text{Lgt}(ts)$$

- Predikáty  $Gt$  a  $Lgt$  sú *vzájomne rekurzívne*
- Problém: CL vzájomnú rekurziu nepodporuje
- Riešenie: dosadíme definíciu  $Gt$  do definície  $Lgt$

$$\begin{array}{l} \text{Lgt}(0) \\ \text{Lgt}\left(\frac{x}{ts_1}, ts_2\right) \leftarrow N(x) \wedge \text{Lgt}(ts_1) \wedge \text{Lgt}(ts_2) \\ \text{Gt}\left(\frac{x}{ts}\right) \leftarrow N(x) \wedge \text{Lgt}(ts) \end{array}$$

Rekurziu na zoznamoch všeobecných stromov využijeme aj ďalej

#### IX.10 Odbočka o zovšeobecňovaní

**Pozorovanie 34.** Zovšeobecnenie často vedie k (efektívnemu) riešeniu.

*Príklad 35.* Riešenia vďaka zovšeobecneniu:

- Rozpoznať kód zoznamu stromov je všeobecnejší problém ako rozpoznať kód stromu.
- $\text{Full\_pre}_1(n, m)$  je všeobecnejšia, ako  $\text{Full\_pre}(n) = \text{Full\_pre}_1(n, 0)$
- $\text{Num\_pre}_1(t, m)$  je všeobecnejšia, ako  $\text{Num\_pre}(t) = \text{Num\_pre}_1(t, 0)$

Efektívne riešenia vďaka zovšeobecneniu:

- 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 ako  $\text{Rev}(xs) = \text{While\_rev}(xs, 0)$
- $\text{Preorder}_a(t, as) = \text{Preorder}(t) \oplus as$  je všeobecnejšia ako  $\text{Preorder}(t) = \text{Preorder}_a(t, 0)$

## 16.2. Programovanie so všeobecnými stromami

### IX.11 Programovanie so všeobecnými stromami

- Pri programovaní funkcií na všeobecných stromoch využívame pomocné funkcie na zoznamoch všeobecných stromov (podobne ako pri  $Gt(t)$ ):

$$F_L(0) = \dots$$

$$F_L\left(\frac{x}{ts_1}, ts_2\right) = \dots F_L(ts_1) \dots F_L(ts_2) \dots$$

$$F\left(\frac{x}{ts}\right) = \dots F_L(ts) \dots$$

- Príklad: Zadefinujeme funkciu  $|t|_g$  počítajúcu veľkosť všeobecného stromu  $t$

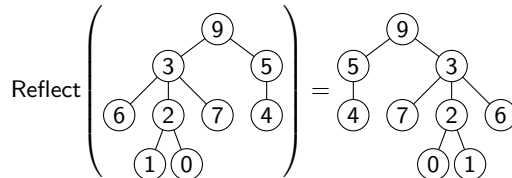
$$|0|_L = 0$$

$$\left|\frac{x}{ts_1}, ts_2\right|_L = 1 + |ts_1|_L + |ts_2|_L$$

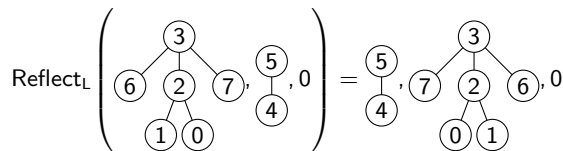
$$\left|\frac{x}{ts}\right|_g = 1 + |ts|_L$$

### IX.12 Zrkadlový obraz

- Zadefinujeme zrkadlový obraz stromu  $t$  — funkciu  $\text{Reflect}(t)$



- Zovšeobecňujeme na zoznamy stromov  $\text{Reflect}_L(ts)$



- Skombinujeme obrátenie zoznamu a zrkadlový obraz binárneho stromu

$$\begin{aligned} \text{Rev}(0) &= 0 \\ \text{Rev}(x, xs) &= \text{Rev}(xs) \oplus (x, 0) \times \text{Reflect}(\bullet) = \bullet \\ &= \frac{\text{Reflect}(r) \times}{\text{Reflect}(\ell)} \end{aligned}$$

- Výsledná definícia

$$\begin{aligned} \text{Reflect}_L(0) &= ? \\ \text{Reflect}_L\left(\frac{x}{ts_1}, ts_2\right) &= ? \end{aligned}$$

- Podobne ako 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:

$$\text{Reflecta}_L\left(\left(\left(\begin{array}{c} \textcircled{2} \\ \textcircled{1} \textcircled{0} \end{array}, \textcircled{7}, 0\right), \textcircled{6}, 0\right)\right) = \textcircled{7}, \begin{array}{c} \textcircled{2} \\ \textcircled{0} \textcircled{1} \end{array}, \textcircled{6}, 0$$

## 16.3. Prechody všeobecnými stromami

- Podobne ako pri binárnych stromoch
- Význam má preorder a postorder, ale nie inorder
- Napríklad:

$$\text{Preorder} \left( \begin{array}{c} \textcircled{9} \\ \textcircled{6} \textcircled{3} \textcircled{5} \\ \textcircled{1} \textcircled{2} \textcircled{7} \textcircled{4} \\ \textcircled{1} \textcircled{0} \end{array} \right) = 9, 3, 6, 2, 1, 0, 7, 5, 4, 0$$

- Opäť zovšeobecníme na zoznamy stromov — funkcia  $\text{Preorder}_L(ts)$

$$\text{Preorder}_L \left( \left( \begin{array}{c} \textcircled{3} \\ \textcircled{6} \quad \textcircled{2} \quad \textcircled{7} \\ \quad \textcircled{1} \quad \textcircled{0} \end{array}, \begin{array}{c} \textcircled{5} \\ \textcircled{4} \end{array}, 0 \right) \right) = 3, 6, 2, 1, 0, 7, 5, 4, 0$$

### IX.15 Prechody všeobecnými stromami

---

- Podobne ako na binárnych stromoch,  $\text{Preorder}$  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:

$$\text{Preorder}_L \left( \left( \left( \begin{array}{c} \textcircled{2} \\ \textcircled{1} \quad \textcircled{0} \end{array}, \textcircled{7}, 0 \right), 5, 4, 0 \right) \right) = 2, 1, 0, 7, 5, 4, 0$$

## X. prednáška

# Aritmetické výrazy a výrokové formuly

27. apríla 2014

## 17. Aritmetické výrazy a výrokové formuly

### 17.1. Aritmetické výrazy

#### X.1 Aritmetické výrazy

---

**Definícia 36.** *Aritmetický výraz* je

- číselná konštanta  $k \in \mathbb{N}$ ,
- premenná  $x_i$  s indexom  $i \in \mathbb{N}$ ,
- súčtový výraz  $(t_1 + t_2)$ ,  
ak  $t_1$  a  $t_2$  sú aritmetické výrazy,
- súčinový výraz  $(t_1 \times t_2)$ ,  
ak  $t_1$  a  $t_2$  sú aritmetické výrazy.

Nič iné nie je aritmetický výraz.

*Príklad 37.*

- 3, 0, 777, ...
- $x_2, x_{17}, x_0, \dots$
- $(x_1 + 5), (x_5 + x_0), (x_3 + (x_0 + x_2)), \dots$
- $(2 \times x_0), (x_4 \times x_1), ((x_0 \times x_2) \times (2 + x_1)), \dots$
- $((x_1 \times x_1) + (2 \times x_1)) + 1, \dots$

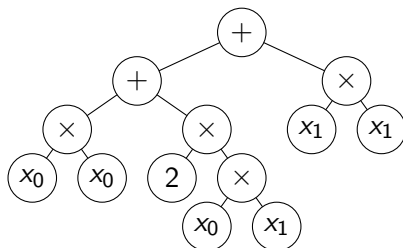
#### X.2 Aritmetické výrazy ako stromy

---

- Aritmetické výrazy majú stromovú štruktúru



- Napríklad  $((x_0 \times x_0) + (2 \times (x_0 \times x_1))) + (x_1 \times x_1)$



- Poznáte z programovania pod menom *aritmetický strom*

### X.3 Kódovanie aritmetických výrazov

---

- Aritmetické výrazy by sme mohli zakódovať (*aritmetizovať*) pomocou binárnych stromov
- Kvôli ďalšiemu rozširovaniu ich ale radšej zakódujeme priamo, *podobne* ako binárne stromy:
  - Párovacie konštruktory (podobne ako E a Nd)

konštanta	$n^\bullet$	$\equiv \text{Ct}(n)$	$= 0, n$
premenná	$x_i^\bullet$	$\equiv \text{Vt}(i)$	$= 1, i$
súčtový výraz	$t_1 +^\bullet t_2$	$\equiv \text{At}(t_1, t_2)$	$= 2, t_1, t_2$
súčinový výraz	$t_1 \times^\bullet t_2$	$\equiv \text{Mt}(t_1, t_2)$	$= 3, t_1, t_2$

- Predikát  $\text{Term}(t)$  platí, ak  $t$  kóduje aritmetický výraz:

$\text{Term}(n^\bullet)$	$\leftarrow \text{N}(n)$
$\text{Term}(x_i^\bullet)$	$\leftarrow \text{N}(i)$
$\text{Term}(t_1 +^\bullet t_2)$	$\leftarrow \text{Term}(t_1) \wedge \text{Term}(t_2)$
$\text{Term}(t_1 \times^\bullet t_2)$	$\leftarrow \text{Term}(t_1) \wedge \text{Term}(t_2)$

#### X.4 Kódovanie aritmetických výrazov — príklad

Príklad 38. Výraz  $((x_0 \times x_0) + (2 \times (x_0 \times x_1))) + (x_1 \times x_1)$  zakódujeme ako

$$\begin{aligned} & ((x_0^\bullet \times^\bullet x_0^\bullet) +^\bullet (2^\bullet \times^\bullet (x_0^\bullet \times^\bullet x_1^\bullet))) +^\bullet (x_1^\bullet \times^\bullet x_1^\bullet) \\ & \equiv \text{At}(\text{At}(\text{Mt}(\text{Vt}(0), \text{Vt}(0)), \\ & \quad \text{Mt}(\text{Ct}(2), \\ & \quad \quad \text{Mt}(\text{Vt}(0), \text{Vt}(1))))), \\ & \quad \text{Mt}(\text{Vt}(1), \text{Vt}(1))) \end{aligned}$$

#### X.5 Štruktúrálna rekúzia na výrazoch

**Definícia 39.** Funkcia je definovaná štruktúrálnou rekúziou na výrazoch (tiež rekúziou na štruktúru výrazu), ak je hodnota  $F$  v rekúzivných prípadoch odvodená od hodnoty  $F$  pre podvýrazy.

Typická definícia štruktúrálnou rekúziou na výrazoch má tvar:

$$\begin{aligned} F(n^\bullet, \dots) &= \dots \\ F(x_i^\bullet, \dots) &= \dots \\ F(t_1 +^\bullet t_2, \dots) &= \dots F(t_1) \dots F(t_2) \dots \\ F(t_1 \times^\bullet t_2, \dots) &= \dots F(t_1) \dots F(t_2) \dots \end{aligned}$$

Príklad 40. Veľkosť výrazu  $Sz(t) \equiv |t|$

$$\begin{aligned} |n^\bullet| &= 1 \\ |x_i^\bullet| &= 1 \\ |t_1 +^\bullet t_2| &= 1 + |t_1| + |t_2| \\ |t_1 \times^\bullet t_2| &= 1 + |t_1| + |t_2| \end{aligned}$$

#### X.6 Ohodnotenie premenných

Na pripomenutie — funkcia indexovania do zoznamu

$$\begin{aligned} \text{Sub}(0, i) &\equiv 0[i] = 0 \\ \text{Sub}((x, xs), 0) &\equiv (x, xs)[0] = x \\ \text{Sub}((x, xs), i + 1) &\equiv (x, xs)[i + 1] = xs[i] \end{aligned}$$

**Definícia 41.** Ohodnotením (valuáciou) premenných nazveme ľubovoľný zoznam.

Hodnotou premennej  $x_i$  pri ohodnotení  $vs$  je číslo  $vs[i]$ .

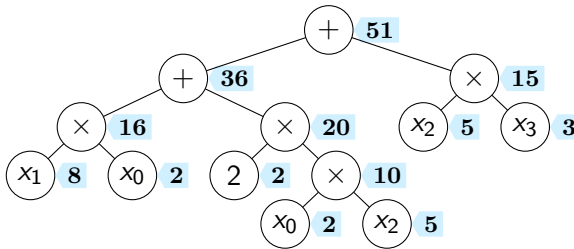
Príklad 42. Pri ohodnotení 2, 8, 5, 3, 0, 12, 9, 0 majú premenné nasledujúce hodnoty:

$$\begin{array}{lll} x_0 = 2 & x_3 = 3 & x_6 = 9 \\ x_1 = 8 & x_4 = 0 & x_i = 0 \text{ pre } i \geq 7 \\ x_2 = 5 & x_5 = 12 & \end{array}$$

### X.7 Hodnota výrazu – príklad

Ak je dané ohodnotenie premenných, môžeme vyhodnotiť celý aritmetický výraz:

Príklad 43. Pri ohodnotení 2, 8, 5, 3, 0, 12, 9, 0 vyhodnotíme výraz  $((x_1 \times x_0) + (2 \times (x_0 \times x_2))) + (x_2 \times x_3)$ :



### X.8 Hodnota výrazu pri ohodnotení premenných

Hodnota (*denotácia*) výrazu pre dané ohodnotenie premenných —  $\text{Den}(t, vs) \equiv \llbracket t \rrbracket^{vs}$

Zadefinujeme štruktúrálnou rekurziou na výrazoch:

$$\begin{aligned} \llbracket n^\bullet \rrbracket^{vs} &= n \\ \llbracket x_i^\bullet \rrbracket^{vs} &= \dots \\ \llbracket t_1 +^\bullet t_2 \rrbracket^{vs} &= \dots \llbracket t_1 \rrbracket^{vs} \dots \llbracket t_2 \rrbracket^{vs} \dots \\ \llbracket t_1 \times^\bullet t_2 \rrbracket^{vs} &= \dots \end{aligned}$$

## 17.2. Transformácia aritmetických výrazov

### X.9 Prezátvorkovanie sčítania doľava

- Sčítanie je asociatívna operácia:  $x + (y + z) = (x + y) + z$

- Zjednotíme uzátvorkovanie sčítaní prezátvorkovaním doľava

*Príklad 44.*

$$\begin{aligned} \text{Lassoc\_from} & \left( (x_0^\bullet +^\bullet x_1^\bullet) +^\bullet (1^\bullet +^\bullet (x_2^\bullet +^\bullet (2^\bullet +^\bullet x_3^\bullet))) \right) \\ & = (((x_0^\bullet +^\bullet x_1^\bullet) +^\bullet 1^\bullet) +^\bullet x_2^\bullet) +^\bullet 2^\bullet +^\bullet x_3^\bullet \end{aligned}$$

- Výsledný tvar bude:

**Definícia 45.** *Výraz s doľava asociovaným sčítaním* je výraz, v ktorom pravý argument žiadneho sčítania (ktoré nie je potomkom násobenia) nie je sčítanie.

$$\text{Lassoc}(n^\bullet)$$

$$\text{Lassoc}(x_i^\bullet)$$

$$\text{Lassoc}(t +^\bullet n^\bullet) \leftarrow \text{Lassoc}(t)$$

$$\text{Lassoc}(t +^\bullet x_i^\bullet) \leftarrow \text{Lassoc}(t)$$

$$\text{Lassoc}(t +^\bullet (t_1 \times^\bullet t_2)) \leftarrow \text{Lassoc}(t)$$

$$\text{Lassoc}(t_1 \times^\bullet t_2)$$

## X.10 Prezátvorkovanie sčítania doľava

---

- Špecifikácia prezátvorkovacej funkcie `Lassoc_from`

$$\begin{aligned} \text{Term}(t) \rightarrow \text{Term } \text{Lassoc\_from}(t) \wedge \llbracket \text{Lassoc\_from}(t) \rrbracket_{\text{vs}} &= \llbracket t \rrbracket_{\text{vs}} \\ \wedge \text{Lassoc } \text{Lassoc\_from}(t) \wedge |\text{Lassoc\_from}(t)| &= |t| \end{aligned}$$

- Samotná funkcia jednoducho popíše, ako sa zmení nevhodný tvar výrazov:

$$\text{Lassoc\_from}(n^\bullet) = n^\bullet$$

$$\text{Lassoc\_from}(x_i^\bullet) = x_i^\bullet$$

$$\text{Lassoc\_from}(t_1 +^\bullet t_2) = ?$$

$$\text{Lassoc\_from}(t_1 \times^\bullet t_2) = t_1 \times^\bullet t_2$$

- Skončí táto rekurzia?

Aká *miera* sa znižuje pri rekurzívnom volaní?

## X.11 Súčtový normálny tvar

---

- Násobenie distribuuje cez sčítanie:  $a \times (b + c) = a \times b + a \times c$
- Roznásobme úplne daný výraz  $t$
- Čo to znamená?

**Definícia 46.** *Súčinový výraz* je aritmetický výraz tvorený iba konštantami, premennými a násobením.

$Mterm(x_i^\bullet)$

$Mterm(n^\bullet)$

$Mterm(t_1 \times^\bullet t_2) \leftarrow Mterm(t_1) \wedge Mterm(t_2)$

**Definícia 47.** Výraz je v *súčtovom normálnom tvare*, ak je súčinovým výrazom, alebo súčtom výrazov v súčtovom normálnom tvare. Nič iné nie je výrazom v súčtovom normálnom tvare.

$Anf(t) \leftarrow Mterm(t)$

$Anf(t) \leftarrow \neg Mterm(t) \wedge t = t_1 +^\bullet t_2 \wedge Anf(t_1) \wedge Anf(t_2)$

- Roznásobenie je prevod do súčtového normálneho tvaru

## X.12 Prevod do súčtového normálneho tvaru

---

- Špecifikácia funkcie  $Anf\_from(t)$ :

$$\begin{aligned} Term(t) &\rightarrow Term Anf\_from(t) \wedge Anf Anf\_from(t) \\ &\wedge \llbracket Anf\_from(t) \rrbracket_{vs} = \llbracket t \rrbracket_{vs} \end{aligned}$$

- Výraz  $s \times^\bullet t$  prevedieme do súčtového norm. tvaru zdola nahor:
  - Najprv prevedieme do súčtového norm. tvaru podvýrazy:

$$\begin{aligned} as &= Anf\_from(s) = s_1 +^\bullet s_2 +^\bullet \cdots +^\bullet s_m \\ at &= Anf\_from(t) = t_1 +^\bullet t_2 +^\bullet \cdots +^\bullet t_n \end{aligned}$$

$s_i$  a  $t_j$  budú súčinové výrazy (platí pre ne  $Mterm$ )



## XI. prednáška

# Výrokové formuly

5. mája 2014

## 17. Aritmetické výrazy a výrokové formuly (pokračovanie)

### 17.3. Výrokové formuly

#### XI.1 Výrokové formuly

---

**Definícia 48.** *Výrokovou formulou (skrátene formulou) je*

- výroková premenná  $X_i$  s indexom  $i \in \mathbb{N}$ ,
- konštanta nepravda  $\perp$ , konštanta pravda  $\top$ ,
- negácia  $\neg A$ , ak  $A$  je výroková formula,
- disjunkcia  $A \vee B$ , konjunkcia  $A \wedge B$  a implikácia  $A \rightarrow B$ , ak  $A$  a  $B$  sú výrokové formuly.

Nič iné nie je výroková formula.

*Príklad 49.*

- $X_5 \vee (X_5 \rightarrow \perp)$
- $(\neg(X_0 \wedge X_1) \rightarrow (\neg X_0 \vee \neg X_1)) \wedge ((\neg X_0 \vee \neg X_1) \rightarrow \neg(X_0 \wedge X_1))$

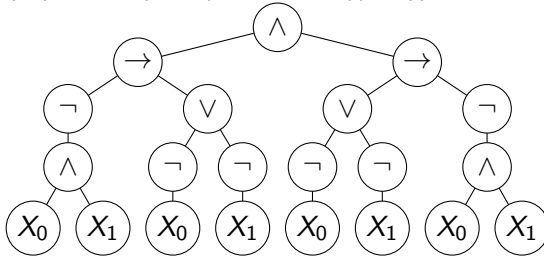
#### XI.2 Výrokové formuly ako stromy a ich kódovanie

---

- Výrokové formuly majú stromovú štruktúru

Príklad 50.

$$(\neg(X_0 \wedge X_1) \rightarrow (\neg X_0 \vee \neg X_1)) \wedge ((\neg X_0 \vee \neg X_1) \rightarrow \neg(X_0 \wedge X_1))$$



- Zakódujeme ich párovacími konštruktormi

$$\begin{array}{lll} X_i^\bullet & \equiv \text{Vf}(i) & \perp^\bullet & \equiv \text{Ff} & \text{T}^\bullet & \equiv \text{Tf} \\ \neg^\bullet f_1 & \equiv \text{Nf}(f_1) & f_1 \vee^\bullet f_2 & \equiv \text{Of}(f_1, f_2) & & \\ f_1 \wedge^\bullet f_1 & \equiv \text{Af}(f_1, f_2) & f_1 \rightarrow^\bullet f_2 & \equiv \text{If}(f_1, f_2) & & \end{array}$$

### XI.3 Kódovanie výrokových formúl

Príklad 51. Formulu  $(\neg(X_0 \wedge X_1) \rightarrow (\neg X_0 \vee \neg X_1)) \wedge ((\neg X_0 \vee \neg X_1) \rightarrow \neg(X_0 \wedge X_1))$  kóduje konštanta

$$\begin{aligned} \text{De\_morgan}_1 &= \text{Af}(\text{If}(\text{Nf Af}(\text{Vf}(0), \text{Vf}(1)), \\ &\quad \text{Of}(\text{Nf Vf}(0), \text{Nf Vf}(1))), \\ &\quad \text{If}(\text{Of}(\text{Nf Vf}(0), \text{Nf Vf}(1)), \\ &\quad \text{Nf Af}(\text{Vf}(0), \text{Vf}(1)))) \end{aligned}$$

Predikát  $\text{Formula}(f)$  platí, ak  $f$  kóduje výrokovú formulu:

$$\begin{aligned} \text{Formula}(X_i^\bullet) &\leftarrow \text{N}(i) \\ \text{Formula}(\perp^\bullet) & \\ \text{Formula}(\text{T}^\bullet) & \\ \text{Formula}(\neg^\bullet f_1) &\leftarrow \text{Formula}(f_1) \\ \text{Formula}(f_1 \vee^\bullet f_2) &\leftarrow \text{Formula}(f_1) \wedge \text{Formula}(f_2) \\ \text{Formula}(f_1 \wedge^\bullet f_2) &\leftarrow \text{Formula}(f_1) \wedge \text{Formula}(f_2) \\ \text{Formula}(f_1 \rightarrow^\bullet f_2) &\leftarrow \text{Formula}(f_1) \wedge \text{Formula}(f_2) \end{aligned}$$

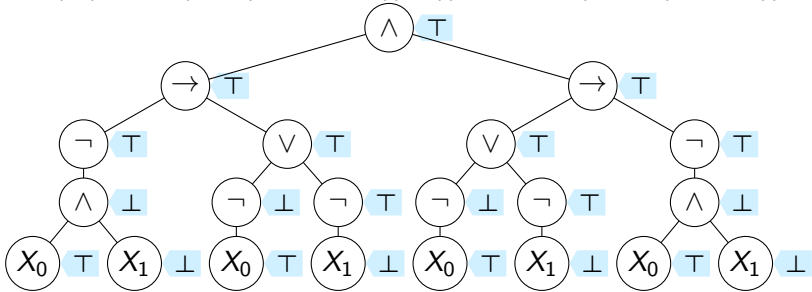
### XI.4 Ohodnotenie premenných a pravdivosť formúl



**Definícia 52.** Ohodnotením (valuáciou) výrokových premenných nazveme ľubovoľný zoznam obsahujúci iba čísla 0 a 1.

Premenná  $X_i$  je pravdivá pri ohodnotení  $vs$  práve vtedy, keď  $vs[i] = 1$ .

Príklad 53. Pri danom ohodnotení  $vs = 1, 0, 0$  vyhodnotíme pravdivosť formuly  $(\neg(X_0 \wedge X_1) \rightarrow (\neg X_0 \vee \neg X_1)) \wedge ((\neg X_0 \vee \neg X_1) \rightarrow \neg(X_0 \wedge X_1))$  takto:



### XI.5 Pravdivosť formuly pri ohodnotení premenných

Pravdivosť výrokovej formuly pre dané ohodnotenie premenných – predikát  $\text{True}(vs, f) \equiv (vs \models f)$

Zadefinujeme štruktúrnou rekurziou na formulách:

- $vs \models T^\bullet$
- $vs \models X_i^\bullet \quad \leftarrow vs[i] = 1$
- $vs \models (\neg^\bullet f_1) \quad \leftarrow \neg(vs \models f_1)$
- $vs \models (f_1 \wedge^\bullet f_2) \quad \leftarrow \dots vs \models f_1 \dots vs \models f_2 \dots$
- $vs \models (f_1 \vee^\bullet f_2) \quad \leftarrow \dots$
- $vs \models (f_1 \rightarrow^\bullet f_2) \quad \leftarrow \dots$

Samozrejme, nepíšeme prípady, v ktorých je predikát True neplatný (teda formula  $f$  je nepravdivá), napríklad

- ~~$\neg(vs \models \perp^\bullet)$~~
- ~~$\neg(vs \models X_i^\bullet) \quad \leftarrow vs[i] \neq 1$~~
- ~~$\neg(vs \models (\neg^\bullet f_1)) \quad \leftarrow vs \models f_1$~~

## 17.4. Transformácia výrokových formúl

## X1.6 Redukcia množiny spojok

---

Niektoré logické spojky postačujú na vyjadrenie ostatných, napríklad:

**Definícia 54.** Výroková formula je *konjunktívna* práve vtedy, keď sa skladá iba z výrokových premenných, negácií a konjunkcií.

$$\text{Cform}(X_i^\bullet)$$

$$\text{Cform}(\neg^\bullet f_1) \leftarrow \text{Cform}(f_1)$$

$$\text{Cform}(f_1 \wedge^\bullet f_2) \leftarrow \text{Cform}(f_1) \wedge \text{Cform}(f_2)$$

**Tvrdenie 55.** *Ku každej výrokovkej formule existuje konjunktívna formula, ktorá je s ňou ekvivalentná.*

Naprogramujeme funkciu `Cform_from`, ktorá ju zostrojí, teda:

$$\text{Form}(f) \rightarrow \text{Form } \text{Cform\_from}(f) \wedge \text{Cform } \text{Cform\_from}(f)$$

$$\text{Form}(f) \rightarrow \forall vs (vs \models \text{Cform\_from}(f) \leftrightarrow vs \models f)$$

## X1.7 Redukcia množiny spojok

---

Na vyjadrenie iných spojok iba pomocou konjunktívnych formúl využijeme vlastnosti:

$$\perp \iff (A \wedge \neg A)$$

$$(A \vee B) \iff \neg(\neg A \wedge \neg B)$$

$$(A \rightarrow B) \iff (\neg A \vee B),$$

ktoré platia pre ľubovoľné výrokové formuly  $A$  a  $B$

Transformačnú funkciu potom zdefinujeme jednoduchou štruktúrnou rekurziou:

$$\text{Cform\_from}(X_i^\bullet) = X_i^\bullet$$

$$\text{Cform\_from}(\neg^\bullet f_1) = \neg^\bullet \text{Cform\_from}(f_1)$$

$$\text{Cform\_from}(f_1 \wedge^\bullet f_2) = \text{Cform\_from}(f_1) \wedge^\bullet \text{Cform\_from}(f_2)$$

$$\text{Cform\_from}(\perp^\bullet) = X_0^\bullet \wedge^\bullet \neg^\bullet X_0^\bullet$$

$$\text{Cform\_from}(\top^\bullet) = \dots$$

$$\text{Cform\_from}(f_1 \vee^\bullet f_2) = \dots \text{Cform\_from}(f_1) \dots \text{Cform\_from}(f_2) \dots$$

$$\text{Cform\_from}(f_1 \rightarrow^\bullet f_2) = \dots \text{Cform\_from}(f_1) \dots \text{Cform\_from}(f_2) \dots$$

**Definícia 56.** Výroková formula je v *negačnom normálnom tvare* práve vtedy, keď sa skladá iba z výrokových premenných, negácií výrokových premenných, disjunkcií a konjunkcií.

$$\text{Nnf}(X_i^\bullet)$$

$$\text{Nnf}(\neg^\bullet X_i^\bullet)$$

$$\text{Nnf}(f_1 \vee^\bullet f_2) \leftarrow \text{Nnf}(f_1) \wedge \text{Nnf}(f_2)$$

$$\text{Nnf}(f_1 \wedge^\bullet f_2) \leftarrow \text{Nnf}(f_1) \wedge \text{Nnf}(f_2)$$

**Tvrdenie 57.** *Ku každej výrokovkej formule existuje formula v negačnom normálnom tvare, ktorá je s ňou ekvivalentná.*

Naprogramujeme funkciu `Nnf_from`, ktorá ju zostrojí, teda:

$$\text{Form}(f) \rightarrow \text{Form } \text{Nnf\_from}(f) \wedge \text{Nnf } \text{Nnf\_from}(f)$$

$$\text{Form}(f) \rightarrow \forall vs (vs \models \text{Nnf\_from}(f) \leftrightarrow vs \models f)$$

Na prevod do negačného normálneho tvaru potrebujeme:

1. Distribuovať negáciu cez všetky spojky až k výrokovým premenným  
Využijeme de Morganove zákony:

$$\neg(A \vee B) \iff (\neg A \wedge \neg B)$$

$$\neg(A \wedge B) \iff (\neg A \vee \neg B)$$

2. Odstrániť implikáciu a konštantné formuly (pravda, nepravda)  
Využijeme vyjadrenia  $\perp$  a implikácie ako pri `Cform_from`

## XI.10 Prevod do negačného normálneho tvaru — program

Výsledný program musí pokryť mnoho prípadov:

$$\begin{aligned} \text{Nnf\_from}(X_i^\bullet) &= X_i^\bullet \\ \text{Nnf\_from}(f_1 \vee^\bullet f_2) &= \text{Nnf\_from}(f_1) \vee^\bullet \text{Nnf\_from}(f_2) \\ \text{Nnf\_from}(f_1 \wedge^\bullet f_2) &= \text{Nnf\_from}(f_1) \wedge^\bullet \text{Nnf\_from}(f_2) \\ \text{Nnf\_from}(\perp^\bullet) &= X_0^\bullet \wedge^\bullet \neg^\bullet X_0^\bullet \\ \text{Nnf\_from}(\top^\bullet) &= \dots \\ \text{Nnf\_from}(f_1 \rightarrow^\bullet f_2) &= \dots \\ \text{Nnf\_from}(\neg^\bullet X_i^\bullet) &= \dots \\ \text{Nnf\_from}(\neg^\bullet \top^\bullet) &= \dots \\ \text{Nnf\_from}(\neg^\bullet \perp^\bullet) &= \dots \\ \text{Nnf\_from}(\neg^\bullet \neg^\bullet f_1) &= \dots \\ \text{Nnf\_from}(\neg^\bullet (f_1 \vee^\bullet f_2)) &= \text{Nnf\_from}(\neg^\bullet f_1) \wedge^\bullet \text{Nnf\_from}(\neg^\bullet f_2) \\ \text{Nnf\_from}(\neg^\bullet (f_1 \wedge^\bullet f_2)) &= \dots \\ \text{Nnf\_from}(\neg^\bullet (f_1 \rightarrow^\bullet f_2)) &= \dots \end{aligned}$$

Negačiu musíme poslať do rekurzie.  
Ak by ostala mimo a  $f_1$  nie je premenná,  
výsledok by nebol v neg. norm. tvare

Rekurzia nie je štruktúrna (rekurzívny argument nie je vždy podformulou pôvodného), ale existuje *miera*, v ktorej sú rekurzívne argumenty menšie — aká?

## Organizácia skúšky a opravného testu

### Opravný test

#### XI.11 Opravný test

**Termín:** streda 28. mája o 13:30 v H6

**Účel:** nahradit' skóre z jedného semestrálneho testu

**Materiál:** ako opravovaný test

#### Podmienky:

- **záväzná prihlásenie** (AIS) do nedele 25. mája 23:59:59
  - 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

## Skúška

### XI.12 Skúška

---

Termíny:	Dátum	Čas	Typ (max. počet študentov)
	streda 28. mája	13:30	riadny (32)
	4. júna	13:30	riadny (16), 1. opravný (16)
	18. júna	13:30	1. opravný (16), 2. opravný (16)
	25. júna	13:30	2. opravný (16)

### Podmienky:

- súčet skóre z testov  $\geq 30$
- záväzné prihlásenie (AIS) najneskôr 24 hodín pred termínom

**Materiál:** úlohy najmä podľa ex10–ex12

**Konzultácie:** utorky 27. mája, 3., 17. a 24. júna od 13:00 do 15:00 v H6

## XII. prednáška

# Postfixový stroj. Kompilátor výrazov

12. mája 2014

## 18. Postfixový stroj. Kompilátor výrazov

### 18.1. Postfixový stroj

#### XII.1 Postfixový stroj

---

- Jednoduchý výpočtový model, virtuálny stroj
- Pamäť: zásobník čísel

5
3
7
2

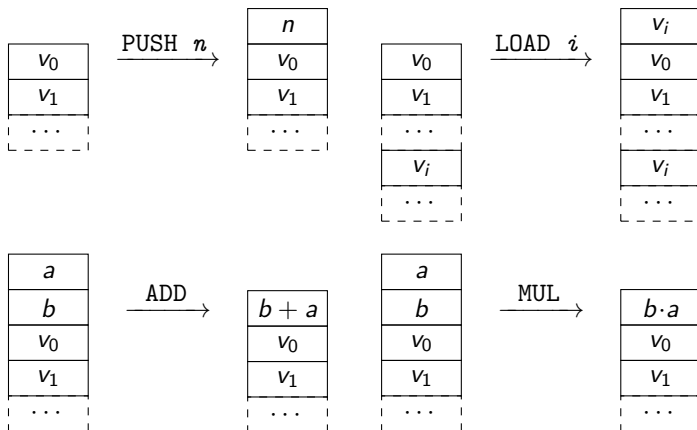
- Program: postupnosť inštrukcií

```
LOAD 2  
PUSH 4  
MUL  
LOAD 2  
LOAD 2  
MUL  
ADD
```

Inštrukcie modifikujú zásobník

Pridávajú/odoberajú čísla na vrch/z vrchu zásobníka

## XII.2 Inštrukcie postfixového stroja

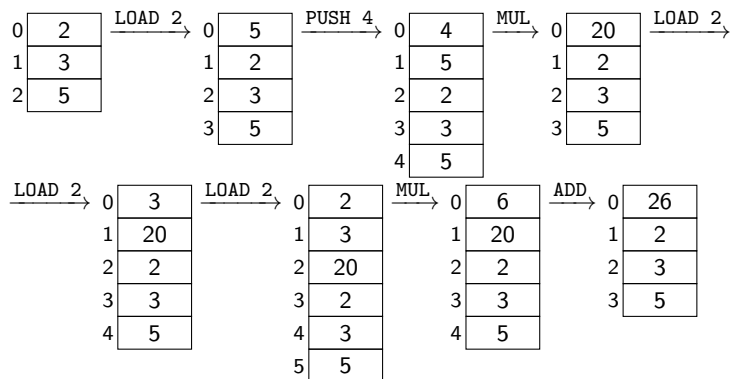


## XII.3 Príklad výpočtu programu

Počítajme program

LOAD 2, PUSH 4, MUL, LOAD 2, LOAD 2, MUL, ADD

so zásobníkom, v ktorom sú čísla 2, 3, 5:

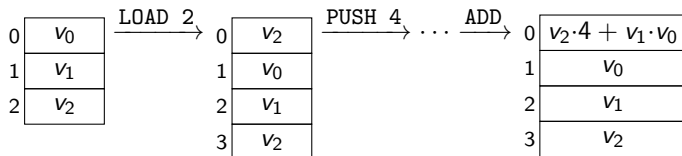


## XII.4 Príklad výpočtu programu — všeobecnejšie

Počítajme program

LOAD 2, PUSH 4, MUL, LOAD 2, LOAD 2, MUL, ADD

so zásobníkom, v ktorom sú čísla  $v_0, v_1, v_2$ :



## XII.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)

$$\begin{aligned} \text{PUSH}(n) &\equiv \text{Push}_i(n) = 0, n & \text{ADD} &\equiv \text{Add}_i = 2, 0 \\ \text{LOAD}(i) &\equiv \text{Load}_i(i) = 1, i & \text{MUL} &\equiv \text{Mul}_i = 3, 0 \end{aligned}$$

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

$$\begin{aligned} \text{Program}(0) & \\ \text{Program}(i, is) &\leftarrow \text{Instr}(i) \wedge \text{Program}(is) \end{aligned}$$

## XII.6 Simulácia postfixových strojov

Zakódovaný postfixový stroj môžeme *simulovať*:

- Funkcia  $\text{Run}(is, vs)$
- Simuluje beh postfixového stroja s programom  $is$  a zásobníkom  $vs$
- Interpretuje program  $is$



- 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. Kompilácia aritmetických výrazov

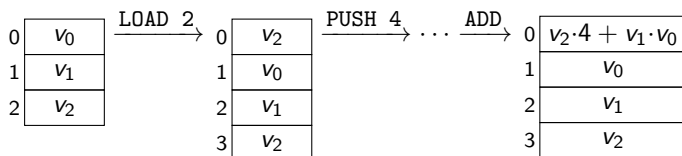
### XII.7 Programy a výrazy — príklad

---

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 aritmetického výrazu

$$(x_2 \times 4) + (x_1 \times x_0)$$

pri ohodnotení premenných  $vs = v_0, v_1, v_2, 0$

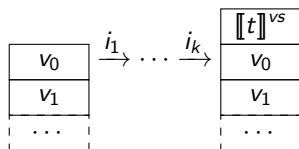
$$\begin{aligned} &\text{Run}(\text{LOAD}(2), \text{PUSH}(4), \text{MUL}, \text{LOAD}(2), \text{LOAD}(2), \text{MUL}, \text{ADD}, 0), vs) \\ &= \llbracket (x_2 \cdot 4) + (x_1 \cdot x_0) \rrbracket^{vs} \end{aligned}$$

### XII.8 Programy a výrazy — kompilácia

---

- Aritmetické výrazy — jazyk vyššej úrovne

- Programy postfixového stroja — jazyk nižšej úrovně
- 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  $t$



$\text{Term}(t) \rightarrow \text{Program } \text{Comp}(t)$

$\text{Term}(t) \rightarrow \text{Run}(\text{Comp}(t), vs) = \llbracket t \rrbracket^{vs}$

## XII.9 Kompilácia — príklad

---

Videli sme, že

$$\begin{aligned} & \text{Run}(\text{LOAD}(2), \text{PUSH}(4), \text{MUL}, \text{LOAD}(2), \text{LOAD}(2), \text{MUL}, \text{ADD}, 0, vs) \\ &= \llbracket (x_2 \bullet \times \bullet 4 \bullet) + \bullet (x_1 \bullet \times \bullet x_0 \bullet) \rrbracket^{vs} \end{aligned}$$

Chceme, aby kompilátor skompiloval výraz

$$(x_2 \bullet \times \bullet 4 \bullet) + \bullet (x_1 \bullet \times \bullet x_0 \bullet)$$

do programu

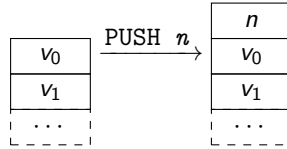
$\text{LOAD}(2), \text{PUSH}(4), \text{MUL}, \text{LOAD}(2), \text{LOAD}(2), \text{MUL}, \text{ADD}, 0$

teda

$$\begin{aligned} & \text{Comp}((x_2 \bullet \times \bullet 4 \bullet) + \bullet (x_1 \bullet \times \bullet x_0 \bullet)) \\ &= \text{LOAD}(2), \text{PUSH}(4), \text{MUL}, \text{LOAD}(2), \text{LOAD}(2), \text{MUL}, \text{ADD}, 0 \end{aligned}$$

XII.10 Kompilácia — konštanty

- Hodnotou výrazu  $n^\bullet$  je číslo  $n$
- Aký program vypočíta na vrch zásobníka hodnotu  $n$ ?

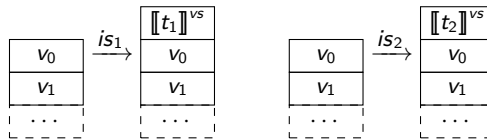


- Klauzula kompilátora:

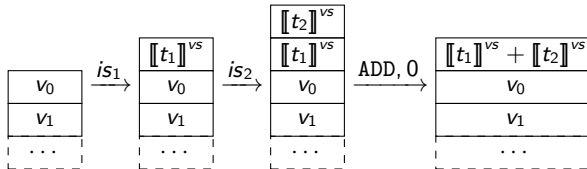
$$\text{Comp}(n^\bullet) = \text{PUSH}(n), 0$$

XII.11 Kompilácia — sčítanie

- Hodnotou výrazu  $t_1 +^\bullet t_2$  je číslo  $\llbracket t_1 \rrbracket^{vs} + \llbracket t_2 \rrbracket^{vs}$
- 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. Zretážíme ich a pridáme inštrukciu pre sčítanie

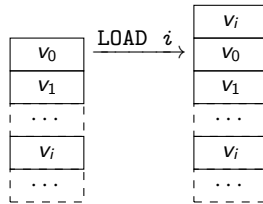


- Klauzula kompilátora:

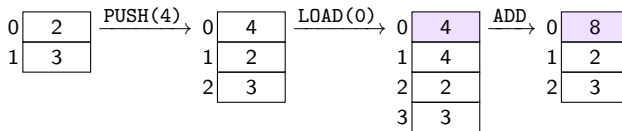
$$\text{Comp}(t_1 +^\bullet t_2) = \text{Comp}(t_1) \oplus \text{Comp}(t_2) \oplus (\text{ADD}, 0)$$

XII.12 Kompilácia — premenná

- 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

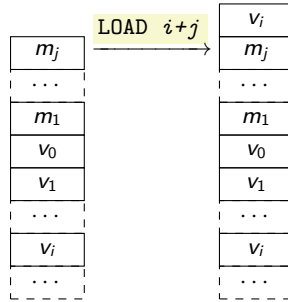
XII.13 Kompilácia — medzivýsledky

- Kompilátor si musí pamätať počet  $j$  medzivýsledkov v zásobníku
- Potrebujeme o niečo všeobecnejšiu kompilačnú funkciu  $\text{Comp}_1(t, j)$ :

$$\begin{aligned} \text{Term}(t) &\rightarrow \text{Program } \text{Comp}_1(t, j) \\ \text{Term}(t) \wedge L(ms) = j &\rightarrow \text{Run}(\text{Comp}_1(t, j) \oplus p, ms \oplus vs) \\ &= \text{Run}(p, (\llbracket t \rrbracket^{vs}, ms \oplus vs)) \end{aligned}$$

- $\text{Comp}(t) = \text{Comp}_1(t, 0)$

- Aký program dá na vrch zásobníka hodnotu  $vs[i]$ ?  
 $\text{Comp}_1(x_i^\bullet, j) = ?$



- Počet medzivýsledkov vzrastie po vypočítaní  $t_1$  vo výrazoch  $t_1 +^\bullet t_2$  a  $t_1 \times^\bullet t_2$ :

$$\text{Comp}_1(t_1 +^\bullet t_2, j) = \text{Comp}_1(t_1, j) \oplus \text{Comp}_1(t_2, j+1) \oplus (\text{ADD}, 0)$$

### 18.3. Podmienené výrazy

XII.15 Podmienené výrazy

Rozšírime výrazy o dva ďalšie typy:

- rozdielový výraz:  $t_1 \dot{-} t_2$
- podmienený výraz:  $D(t_1, t_2, t_3)$  (if  $t_1 \neq 0$  then  $t_2$  else  $t_3$ )

Zakódujeme ich novými konštruktormi:

$$t_1 \dot{-}^\bullet t_2 \equiv \text{St}(t_1, t_2) = 4, t_1, t_2$$

$$D^\bullet(t_1, t_2, t_3) \equiv \text{Dt}(t_1, t_2, t_3) = 5, t_1, t_2, t_3$$

Rozšírime predikát Term:

$$\text{Term}(n^\bullet) \leftarrow N(n)$$

...

$$\text{Term}(t_1 \dot{-}^\bullet t_2) \leftarrow \text{Term}(t_1) \wedge \text{Term}(t_2)$$

$$\text{Term}(D^\bullet(t_1, t_2, t_3)) \leftarrow \text{Term}(t_1) \wedge \text{Term}(t_2) \wedge \text{Term}(t_3)$$

a denotačnú funkciu:

...

$$\begin{aligned} \llbracket t_1 \dot{\div} t_2 \rrbracket^{vs} &= \llbracket t_1 \rrbracket^{vs} \dot{\div} \llbracket t_2 \rrbracket^{vs} \\ \llbracket \mathbf{D}^\bullet(t_1, t_2, t_3) \rrbracket^{vs} &= ? \quad \leftarrow ? \end{aligned}$$

### XII.16 Podmienené výrazy – použitie

- Rozdielovým a podmieneným výrazom vyjadríme napríklad maximum:

$$\text{Dterm}_1 = \mathbf{D}^\bullet(x_0 \dot{\div} x_1, x_0, x_1)$$

vďaka vlastnosti

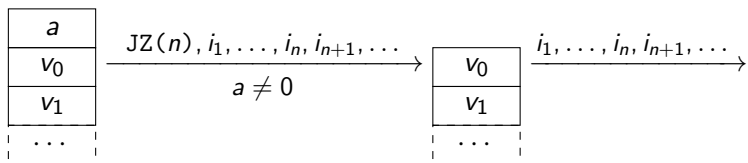
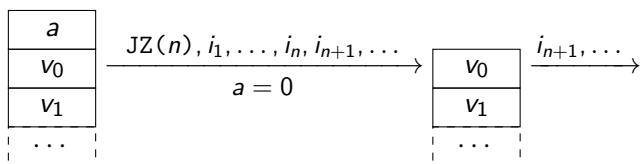
$$x \dot{\div} y \neq 0 \leftrightarrow x > y$$

- Mohli by sme si zdefinovať aj skratku

$$\text{Term\_dich}(ta, tb, tgt, tleq) = \mathbf{D}^\bullet(ta \dot{\div} tb, tgt, tleq)$$

### XII.17 Podmienený skok a odčítanie

- Podmienené výrazy a príkazy sa v strojovom kóde vyhodnocujú pomocou skokov
- Stačí nám jeden podmienený skok – inštrukcia  $\text{JZ}(n)$ :



- Potrebujeme aj inštrukciu odčítania  $\text{SUB}$

Funguje podobne ako  $\text{ADD}$  a  $\text{MUL}$

XII.18 Kódovanie a simulácia skoku a odčítania

- Zakódujme nové inštrukcie:

$$\text{SUB} \equiv \text{Subi} = 4, 0 \qquad \text{JZ}(n) \equiv \text{Jzi}(n) = 5, n$$

Instr( $\dots$ )  
 Instr(SUB)  
 Instr JZ( $n$ )  $\leftarrow$  N( $n$ )

- Rozšírme simulačnú funkciu:

$$\begin{aligned} \text{Run}(\dots, \dots) &= \dots \\ \text{Run}(\text{JZ}(n), is) &= \text{Run}(?, ?) \leftarrow a = 0 \\ \text{Run}(\text{JZ}(n), is) &= \text{Run}(?, ?) \leftarrow a \neq 0 \end{aligned}$$

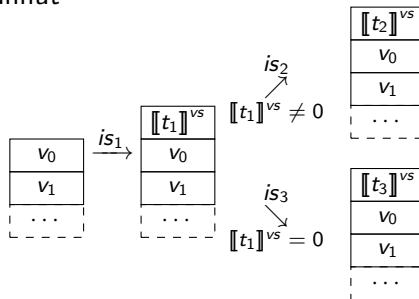
Potrebujeme preskočiť časť programu:

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

XII.19 Kompilácia – podmienený výraz

- Hodnotou výrazu  $\mathbf{D}^\bullet(t_1, t_2, t_3)$  je číslo  $\llbracket t_2 \rrbracket^{vs}$ , ak  $\llbracket t_1 \rrbracket^{vs} \neq 0$ , a číslo  $\llbracket t_3 \rrbracket^{vs}$ , ak  $\llbracket t_1 \rrbracket^{vs} = 0$ ;
- Akým programom ju vypočítame bez počítania hodnoty nepotrebného výrazu?

1. Skompilujeme výrazy  $t_1$  až  $t_3$  do programov  $is_1$  až  $is_3$
2. Chceme dosiahnuť



V čase kompilácie nevieme vybrať medzi  $is_2$  a  $is_3$ ,  
ale vieme inštruovať stroj, aby nesprávny program *preskočil*

## XII.20 Kompilácia – podmienený výraz

---

- Programy  $is_1$ ,  $is_2$ ,  $is_3$  pre výrazy  $t_1$ ,  $t_2$ ,  $t_3$  teda zoradíme za seba a oddelíme skokmi:

$$\text{Comp}_1(\mathbf{D}^\bullet(t_1, t_2, t_3), j) = is_1 \oplus (\text{JZ}(\text{?}), 0) \oplus is_2 \oplus (\text{?}, \text{JZ}(\text{?}), 0) \oplus is_3$$

$$\leftarrow \text{Comp}_1(t_1, j) = is_1 \wedge \text{Comp}_1(t_2, j) = is_2 \wedge \text{Comp}_1(t_3, j) = is_3$$

- Druhý skok sa musí vykonať vždy a nesmie porušiť stav zásobníka

Ako to dosiahneme?



## XIII. prednáška

# Generovanie XML/XHTML

19. mája 2014

## 19. XML/XHTML

### 19.1. Štruktúra

#### XIII.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>
```

#### XIII.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>

```

### XIII.3 Strom elementov

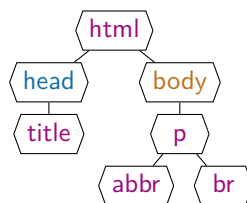
---

Elementy vytvárajú stromovú štruktúru:

```

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Príklad</title>
  </head>
  <body>
    ...
  </body>
</html>

```



Čo v strome chýba?

### XIII.4 Text dokumentu

---

```

<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>

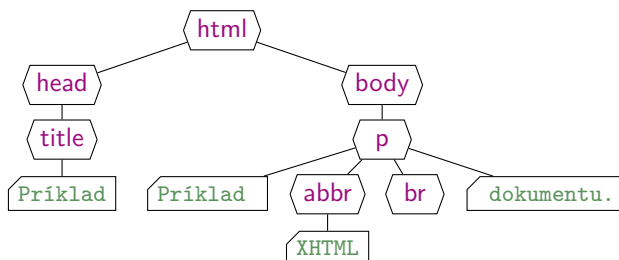
```

V strome elementov chýba samotný text dokumentu

### XIII.5 Strom elementových a textových vrcholov

---

Pridáme vrcholy pre neštruktúrovaný text:



Všeobecný strom s dvoma druhmi vrcholov:

- **elementové vrcholy** — ľubovoľne veľa detí (aj 0, napr. `br`)
- **textové vrcholy** — listy, žiadne deti

## 19.2. Kódovanie

### XIII.6 Kódovanie stromu dokumentu v CL

---

- Strom XML dokumentu má 2 druhy vrcholov
- Každý druh zakódujeme osobitným konštruktorom (podobne ako `E` a `Nd`)
- Textové vrcholy:  $X_s(t)$ 
  - $t$  — text vo vrchole — reťazec
- Elementové vrcholy:  $X_e(n, as, cs)$ 
  - $n$  — meno elementu (`html`, `body`, `p`, ...) — reťazec
  - $as$  — atribúty elementu
  - $cs$  — deti elementu

### XIII.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:

$$\text{Lattr}(0)$$
$$\text{Lattr}(a, as) \leftarrow \text{Attr}(a) \wedge \text{Lattr}(as)$$

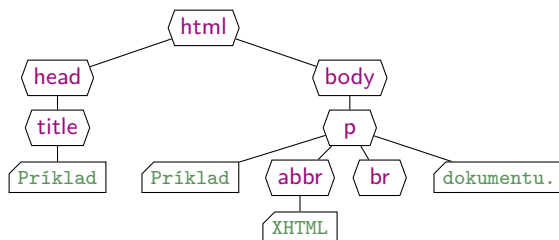
- Napríklad:

$$('href', 'http://example.com/'), ('class', 'external'), 0$$

### XIII.8 Kódovanie detí elementu

$Xe(n, as, cs)$

- Elementové vrcholy majú rôzny počet detí:



- Deti zakódujeme do zoznamu vrcholov XML stromu

### XIII.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)
```

### XIII.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 vrchol XML stromu
  - Pomocný predikát  $Lx(xs)$  —  $xs$  je zoznam XML vrcholov

- Prirodzená definícia *vzájomnou rekurziou*:

$$\begin{aligned} X X_e(n, as, cs) &\leftarrow \text{Str}(n) \wedge \text{Lattr}(as) \wedge Lx(cs) \\ X X_s(t) &\leftarrow \text{Str}(t) \end{aligned}$$

$$\begin{aligned} Lx(0) \\ Lx(x, xs) &\leftarrow X(x) \wedge Lx(xs) \end{aligned}$$

- CL neumožňuje vzájomnú rekurziu:

$$\begin{aligned} Lx(0) \\ Lx(X_e(n, as, cs), xs) &\leftarrow \text{Str}(n) \wedge \text{Lattr}(as) \wedge Lx(cs) \wedge Lx(xs) \\ Lx(X_s(t), xs) &\leftarrow \text{Str}(t) \wedge Lx(xs) \end{aligned}$$

$$\begin{aligned} X X_e(n, as, cs) &\leftarrow \text{Str}(n) \wedge \text{Lattr}(as) \wedge Lx(cs) \\ X X_s(t) &\leftarrow \text{Str}(t) \end{aligned}$$

## XIII.12 Zobrazenie XML

- Ak v CL zakódujeme strom XML dokumentu, zobrazíme ho ako skutočné XML formátom Xml
- Formátom Xml môžeme zobrazovať *iba* hodnoty  $xs$ , ktoré:
  - sú jednoprvkovými zoznamami vrcholov:  $Lx(xs) \wedge L(xs) = 1$
  - ich jediným prvkom je  $X_e$
- ▶ pretože XML dokument musí mať jeden koreňový element

Príklad 58.

```
Query: .....
Xe('pokus', 0, Xs('text'), 0), 0 = xs: Xml
Results: .....
xs = <pokus>text</pokus>
```

- Skratková funkcia  $Xhtml(t, cs)$  — kostra XHTML dokumentu  
 $t$  — titulok — reťazec  
 $cs$  — obsah tela (body) — Lx
- Formát Xml zobrazí výsledok ako skutočné XHTML

Príklad 59.

Query: .....  
 $Xhtml('Príklad dokumentu', Xe('h1', 0), Xs('Nadpis'), 0), 0) = xs: Xml$   
 Results: .....  
 $xs =$

Príklad dokumentu
<b>Nadpis</b>

## 19.3. Neformálne typovanie

- CL je beztypový jazyk — všetky dáta sú čísla
- Môžeme typovať neformálne predikátmi  
 $(T_1, T_2$  sú predikáty,  $F$  je funkcia)

$$F :: T_1 \rightarrow T_2$$

– skratka za  $T_1(x) \rightarrow T_2(F(x))$

– ak argument funkcie  $F$  je typu  $T_1$ , tak výsledok je typu  $T_2$

- 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  $Xe$ ,  $Xs$ ,  $Xhtml$ ?

## 19.4. Generovanie

### XIII.15 Generovanie XML

---

Príklad 60. Zadefinujme funkciu  $Mk\_ul(items)$ , ktorá

- zo zoznamu reťazcov  $items$
- vytvorí nečíslovaný XHTML zoznam ( $ul$ ),
- ktorého položky ( $li$ ) obsahujú reťazce zo zoznamu  $items$ .

Otypujme: Funkcia vytvorí jeden vrchol XML stromu:

$$Mk\_ul :: Lstr \rightarrow X$$

Naprogramujme:

- Potrebujeme pomocnú funkciu  $Mk\_lis(items)$ , ktorá vytvorí zoznam vrcholov XML stromu (elementov  $li$ ):

$$Mk\_lis :: Lstr \rightarrow Lx$$

$$Mk\_lis(0) = 0$$

$$Mk\_lis(item, items) = Xe('li', 0, Xs(item), 0), Mk\_lis(items)$$

- Vytvorené položky použijeme ako deti elementu  $ul$ :

$$Mk\_ul(items) = Xe('ul', 0, Mk\_lis(items))$$