Prednášky z Úvodu do deklaratívneho programovania

Ján Kľuka

Letný semester 2011/2012

Obsah

I. prednáška
Organizácia kurzu
Deklaratívne programovanie v CL
Explicitné definície

13. februára 2012

1. Organizácia kurzu

1.1. Rozvrh a kontakty

Prednášky pondelok 12:20, poslucháreň A

Cvičenia	pondelok	16:30	H6	1iai{1,4,5}
	pondelok	18:10	H6	1iai{2,3,4}
				{2,3}iui
	streda	14:00	H6	2iai*

Dodržte termín cvičení podľa ročníka a krúžku! (Inak ste hostia, nemôžeme sa vám venovať.)

Prváci: Možná výmena človek za človeka Tretiaci: Prihláste sa na termín podľa vlastného výberu (e-mail↓)

Konzultácie TBA

Web http://dai.fmph.uniba.sk/courses/udp/

E-mail *udp(zavináč)lists.dai.fmph.uniba.sk*

1.2. Pravidlá hodnotenia

I.2 Organizácia kurzu

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

Náhradný test
- 1. týždeň skúškového obdobia
- Náhrada 1 semestrálneho testu
- Rovnaká časť učiva
- Skóre nahradené bez ohľadu výšku voči nahrádzanej hodnote

Skúška
- Podmienka pripustenia ku skúške zisk $\geq$ 30 bodov za semester
- 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 to, *čo* sa počíta, nie (len) na to, *ako* sa počíta
- Založené na matematickej logike
- Program je zapísaný v nejakom jazyku logiky
- Význam programu: matematický objekt Program *definuje* funkciu/reláciu

- Deklaratívne programovacie jazyky
  - funkcionálne (Lisp, ML, Haskell, ...)
  - logické (Prolog, Trilogy, ...)
- *Výhoda:* uľahčuje tvorbu *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 (spĺňa špecifikáciu)

Deklaratívne programovanie:

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

2.1. Príklad

I.5 Príklad

Neformálne zadanie

- Potrebujeme 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:

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

I.6 Príklad

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 podmienených rovností:

$$\gcd(x, y) = y \qquad \leftarrow x = 0$$
$$\gcd(x, y) = \gcd(y \bmod x, x) \leftarrow x \neq 0$$

- **Výpočet:** Zjednodušovanie výrazov podľa podmienených rovností do základného tvaru (číslo)
- Napríklad $\gcd(21, 12) = \cdots$
- Výpočet vždy skončí, lebo

$$x \neq 0 \rightarrow y \bmod x < x$$

I.7 Príklad

Špecifikácia – požadovaná vlastnosť

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

Implementácia – program v jazyku podmienených rovností

$$\gcd(x, y) = y \qquad \leftarrow x = 0$$
$$\gcd(x, y) = \gcd(y \bmod x, x) \leftarrow x \neq 0$$

Má program požadovanú vlastnosť?

Testovanie *Výpočtom* overíme pre *niektoré* vstupy

Verifikácia *Dôkazom* overíme pre *všetky* vstupy

- Dôkaz úplnou matematickou indukciou

3. Jazyk CL

I.8 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

I.9 Niektoré funkcie zabudované v CL

- Sčítanie $(+)$ a násobenie $(\cdot)$ prirodzených čísel
- Modifikované odčítanie

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

- Delenie a zvyšok po delení na prirodzených číslach

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

4. Explicitné definície

I.10 Explicitné definície

Explicitná definícia funkcie

- Program, ktorý vypočíta výsledok ako výraz zložený iba z premenných, číselných konštánt a predtým definovaných funkcií

- Jednoduché explicitné definície majú iba jednu rovnosť, napríklad:

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

4.1. Explicitné definície s jednou podmienkou

I.11 **Explicitné definície s podmienkami**

- Zložitejšie definície majú viacero podmienených rovností

$\mathrm{sgn}(x) = 0 \leftarrow x = 0$ $\qquad$ $\min(x, y) = y \leftarrow x > y$
$\mathrm{sgn}(x) = 1 \leftarrow x \neq 0$ $\qquad$ $\min(x, y) = x \leftarrow x \leq y$

- Podmienená rovnosť sa nazýva *klauzula* (clause)
- Argumenty funkcie v rôznych klauzulách musia byť rovnaké

$\min(x, y) = x \leftarrow x \leq y$
~~$\min(a, b) = b \leftarrow a > b$~~

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

- Dovolenú sadu podmienok voláme *diskriminácia*
- Niektoré dovolené diskriminácie:

diskriminácia na rovnosť	dichotómia
$f(\cdots) = \cdots \leftarrow t = s$	$f(\cdots) = \cdots \leftarrow t \leq s$
$f(\cdots) = \cdots \leftarrow t \neq s$	$f(\cdots) = \cdots \leftarrow t > s$
$\neg(t = s \wedge t \neq s)$	$\neg(t \leq s \wedge t > s)$
$t = s \vee t \neq s$	$t \leq s \vee t > s$

t a s sú ľubovoľné výrazy

- Je dovolené napríklad:

$f(x) = x \quad \leftarrow x{+}7 = x{\cdot}7$ $\qquad$ $g(x,y) = x{\cdot}y \quad \leftarrow y > x$
$f(x) = x{+}1 \leftarrow x{+}7 \neq x{\cdot}7$ $\qquad$ $g(x,y) = x{+}y \leftarrow y \leq x$

- Nie je dovolené napríklad:

$f(x,y) = 0 \leftarrow x = y$ $\qquad$ $g(x,y,z) = y{+}z \leftarrow y \leq x$
$f(x,y) = 1 \leftarrow y \neq x$ $\qquad$ $g(x,y,z) = x{+}z \leftarrow y = x$

4.2. Explicitné definície s viacerými podmienkami

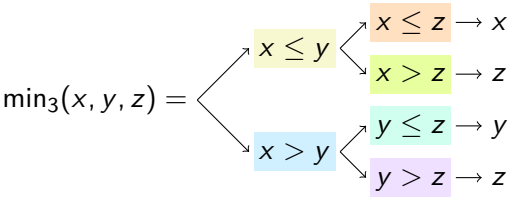
- Podmienky v klauzulách možno kombinovať *iba* logickou spojkou $\wedge$
- Dovolené sú len niektoré kombinácie podmienok
- CL musí rozpoznať, že podmienky sú vzájomne výlučné a pokrývajú všetky prípady
- Nie je dovolené napríklad:

$\min_3(x,y,z) = x \leftarrow x \leq y \wedge x \leq z$
$\min_3(x,y,z) = y \leftarrow y \leq z \wedge y \leq x$
$\min_3(x,y,z) = z \leftarrow z \leq x \wedge z \leq y$

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

$$\begin{aligned}
\min_3(x,y,z) = x &\leftarrow x \leq y \wedge x \leq z \\
\min_3(x,y,z) = z &\leftarrow x \leq y \wedge x > z \\
\min_3(x,y,z) = y &\leftarrow x > y \wedge y \leq z \\
\min_3(x,y,z) = z &\leftarrow x > y \wedge y > z
\end{aligned}$$

vzniknú vnorením diskriminácií:



Zaručene správna konštrukcia podmienok:
kým nevieme určiť výsledok, vnárame dovolené diskriminácie

$$\min_3(x,y,z) = ? \leftarrow ?$$

$$\begin{aligned}
\min_3(x,y,z) = ? &\leftarrow x \leq y \\
\min_3(x,y,z) = ? &\leftarrow x > y
\end{aligned}$$

$$\begin{aligned}
\min_3(x,y,z) = x &\leftarrow x \leq y \wedge x \leq z \\
\min_3(x,y,z) = z &\leftarrow x \leq y \wedge x > z \\
\min_3(x,y,z) = ? &\leftarrow x > y
\end{aligned}$$

$$\begin{aligned}
\min_3(x,y,z) = x &\leftarrow x \leq y \wedge x \leq z \\
\min_3(x,y,z) = z &\leftarrow x \leq y \wedge x > z \\
\min_3(x,y,z) = y &\leftarrow x > y \wedge y \geq z \\
\min_3(x,y,z) = z &\leftarrow x > y \wedge y > z
\end{aligned}$$

4.3. Verifikácia explicitne definovaných funkcií

I.16 Verifikácia explicitne definovaných funkcií

Hodnotou funkcie $\min_3$ je najmenší z jej argumentov, teda hodnota $\min_3$ sa

- rovná niektorému z argumentov:

$$\min_3(x, y, z) = x \vee \min_3(x, y, z) = y \vee \min_3(x, y, z) = z$$

- nanajvýš rovná každému argumentu:

$$\min_3(x, y, z) \leq x \wedge \min_3(x, y, z) \leq y \wedge \min_3(x, y, z) \leq z$$

Dokážme, že tieto vlastnosti platia pre našu definíciu

$$\min_3(x, y, z) = x \leftarrow x \leq y \wedge x \leq z \quad (1)$$
$$\min_3(x, y, z) = z \leftarrow x \leq y \wedge x > z \quad (2)$$
$$\min_3(x, y, z) = y \leftarrow x > y \wedge y \leq z \quad (3)$$
$$\min_3(x, y, z) = z \leftarrow x > y \wedge y > z \quad (4)$$

Ako na to?

I.17 Verifikácia explicitne definovaných funkcií

$$\min_3(x, y, z) \leq x \wedge \min_3(x, y, z) \leq y \wedge \min_3(x, y, z) \leq z \qquad (*)$$

Priamy dôkaz: Nech x, y, z sú ľubovoľné čísla.
Na určenie hodnoty $\min_3(x, y, z)$ podľa našej definície musíme porovnať x a y – tak, ako v samotnej definícii:

- Ak $x \leq y$, hodnotu $\min_3$ určíme porovnaním x a z.
  - Ak $x \leq z$, podľa (1) je $\min_3(x, y, z) = x$. Zrejme $x \leq x$ a podľa aktuálnych predpokladov $x \leq y$ a $x \leq z$. Platia teda všetky konjunkty $(*)$.
  - Ak $x > z$, podľa (2) je $\min_3(x, y, z) = z$. Podľa aktuálnych predpokladov $z < x \leq y$ a zrejme $z \leq z$. Opäť teda platia všetky konjunkty $(*)$.
- Ak $x > y$, hodnotu $\min_3$ určíme porovnaním y a z. Postup je podobný ako v predchádzajúcom prípade.

II. prednáška
Rekurzia

20. februára 2012

4.4. Opakovanie a prémiová domáca úloha

II.1 Opakovanie 1. prednášky

- Explicitné definície – skladanie už definovaných funkcií

$$\begin{aligned}\mathrm{Square}(x) &= x{\cdot}x\\ \mathrm{Cube}(x) &= x{\cdot}\mathrm{Square}(x)\end{aligned}$$

- Explicitné klauzálne definície – podmienené výpočty

$$\begin{aligned}\min(x,y) &= x \leftarrow x \leq y\\ \min(x,y) &= y \leftarrow x > y\end{aligned}$$

  Diskriminácie – dovolené kombinácie podmienok

$$\begin{aligned}\cdots &\leftarrow s \leq t & \cdots &\leftarrow s = t\\ \cdots &\leftarrow s > t & \cdots &\leftarrow s \neq t\end{aligned}$$

  vzájomne výlučné, pokrývajúce všetky prípady

$$\begin{aligned}\neg(s \leq t \wedge s > t)\\ (s \leq t \vee s > t)\end{aligned} \qquad \begin{aligned}\neg(s = t \wedge s \neq t)\\ (s = t \vee s \neq t)\end{aligned}$$

- ▶ Chýba nám *opakovanie*

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

Dokážte, že

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

Pozor!

- Vychádzame z našej definície funkcie, musíme ju uviesť.
- Nemôžeme predpokladať, že vlastnosť platí, keď ju máme dokázať. (Dôkaz kruhom.)
- Dokazujeme rovnostnú vlastnosť Median_perm1, teda musíme ukázať, že za každých okolností sú hodnoty na ľavej a pravej strane identické. Nestačí zistiť, čomu sa rovná ľavá strana.

Ako má teda dôkaz vyzerať?

II.3 Prémiová domáca úloha – dôkaz

$\text{Median}(a, b, c) = b \leftarrow a \leq b \wedge b \leq c$ (1)
$\text{Median}(a, b, c) = c \leftarrow a \leq b \wedge b > c \wedge a \leq c$ (2)
$\text{Median}(a, b, c) = a \leftarrow a \leq b \wedge b > c \wedge a > c$ (3)
$\text{Median}(a, b, c) = b \leftarrow a > b \wedge b > c$ (4)
$\text{Median}(a, b, c) = a \leftarrow a > b \wedge b \leq c \wedge a \leq c$ (5)
$\text{Median}(a, b, c) = c \leftarrow a > b \wedge b \leq c \wedge a > c$ (6)

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

- Ak $x \leq y$ a $y \leq z$, tak $\text{Median}(x, y, z) = y$ podľa (1).
  Čomu sa rovná druhá strana $\text{Median}(y, x, z)$ dokazovanej rovnosti (Median_perm1)?
  Dosaďme $a := y$, $b := x$, $c := z$ do (1)–(6).
  Platí $b \leq a \leq c$. Hodnotu $\text{Median}(a, b, c) = \text{Median}(y, x, z)$ určí až ďalšia analýza prípadov:

  - Ak $a = b$, tak $a = b \leq c$ a $\text{Median}(a, b, c) \overset{(1)}{=} b = a$.
  - Ak $a < b$, musíme rozhodnúť medzi klauzulami (4)–(6):
    * Prípad $b > c$ nemôže nastať (spor s predpokladom).
    * Ak $b \leq c$, $\text{Median}(a, b, c) \overset{(5)}{=} a$.

  V oboch prípadoch $\text{Median}(y, x, z) = \text{Median}(a, b, c) = a = y = \text{Median}(x, y, z)$, a teda (Median_perm1) platí.
- V ďalších prípadoch postupujeme podobne . . .

5. Rekurzívne definície

II.4 Princíp rekurzie

- Opakovanie v deklaratívnom programovaní – rekurzia
- *Rekurzia* vyjadruje hodnotu funkcie pre nejaký argument pomocou hodnoty *tej istej funkcie* pre *jednoduchšie argumenty*

$$0! = 1$$
$$(n+1)! = (n+1) \cdot n!$$

- Rôzne druhy rekurzie – 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:

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

- Výpočet:

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

- Čo je potrebné na počítačovú realizáciu výpočtu?
  Zásobník

II.6 Primitívna rekurzia

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

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

- Všetky programy na tomto predmete budú *primitívne rekurzívne*
  - Definovateľné primitívnou rekurziou a skladaním funkcií z nuly $\mathrm{Z}(x) = 0$, nasledovníka $\mathrm{S}(x) = x+1$ a projekcií $\mathrm{I}_k^n(x_1, \ldots, x_k, \ldots, x_n) = x_k$
- 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. Course-of-values rekurzia

II.7 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
- *Course-of-values* rekurzia: rekurzívny argument je *ľubovoľné menšie* číslo
- Príklad: Vieme iba sčitovať, odčitovať, násobiť, chceme delenie $\mathrm{Div}(x, y) = x \div y$:

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

- Definícia pomocou course-of-values rekurzie:

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

  Je zaručené zmenšovanie rekurzívneho argumentu?

$$y \neq 0 \wedge x \geq y \to x \dot{-} y < x$$

II.8 Course-of-values rekurzia

- Fibonacciho postupnosť $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

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

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

- Definícia course-of-values rekurziou
  - s *dvoma* rekurzívnymi volaniami, a
  - so zloženými podmienkami:

$$\mathrm{fib}_n = \cdots \leftarrow \cdots$$

5.3. Course-of-values rekurzia so substitúciou v parametri

II.9 Substitúcia v parametri

- Najväčší spoločný deliteľ:

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

- Využijeme

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

- Definícia course-of-values rekurziou:

$$\gcd(x,y) = y \qquad \leftarrow x = 0$$
$$\gcd(x,y) = \gcd(y \bmod x, x) \leftarrow x \neq 0$$

- Rekurzívny argument (zaručene klesajúci):

  prvý (x)

- Mení sa aj nerekurzívny argument (*parameter*): course-of-values rekurzia *so substitúciou v parametri*

5.4. Verifikácia rekurzívne definovaných funkcií indukciou

II.10 Verifikácia primitívne rekurzívnych funkcií

Funkcia

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

má dobre známu vlastnosť:

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

Ako ju dokážeme? Skúsme analýzu prípadov podľa definície:

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

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

  Zjavne teda $(*)$ platí.

- Ak $n > 0$, podľa druhej klauzuly $\sum_{i=0}^{n} i = n + \sum_{i=0}^{n \dot{-} 1} i$ a

  $$\frac{n \cdot (n+1)}{2} = \frac{((n \dot{-} 1)+1) \cdot (n+1)}{2} = \frac{(n \dot{-} 1) \cdot n + 2 \cdot n}{2} = n + \frac{(n \dot{-} 1) \cdot n}{2} .$$

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

  Potrebujeme, aby

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

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

  Inak povedané, potrebujeme *indukčný predpoklad*.

Začnime teda odznova:

II.11 Primitívna rekurzia a matematická indukcia

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

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

Dôkaz matematickou indukciou na n:

1. Dokážme $(*)$ pre 0:

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

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

   Dokážme pre $n+1$:

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

II.12 Course-of-values rekurzia a . . . ?

Funkcie definované primitívnou rekurziou spravidla verifikujeme matematickou indukciou.
Funkcie definované course-of-values rekurziou spravidla verifikujeme . . . ?

III. prednáška
Priraďujúce diskriminácie, simulácia cyklov chvostovou rekurziou

27. februára 2012

5.5. Opakovanie

III.1 Opakovanie: Rekurzia

Rekurzia

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

Primitívna rekurzia

argument rekurzívneho volania je o 1 menší

$$x^y = 1 \qquad \leftarrow y = 0$$
$$x^y = x \cdot (x^{y \dot{-} 1}) \leftarrow y \neq 0$$

Všeobecná rekurzia (course-of-values)

argument rekurzívneho volania je ľubovoľné číslo ostro menšie ako pôvodný argument

$$\gcd(x, y) = y \qquad \leftarrow x = 0$$
$$\gcd(x, y) = \gcd(y \bmod x, x) \leftarrow x \neq 0 \qquad (x \neq 0 \rightarrow y \bmod x < x)$$

rekurzívnych volaní môže byť viac

$$\mathrm{fib}_n = 0 \qquad \leftarrow n < 2 \wedge n = 0$$
$$\mathrm{fib}_n = 1 \qquad \leftarrow n < 2 \wedge n \neq 0$$
$$\mathrm{fib}_n = \mathrm{fib}_{n \dot{-} 1} + \mathrm{fib}_{n \dot{-} 2} \leftarrow n \geq 2 \quad (n \geq 2 \rightarrow n \dot{-} 1 < n \wedge n \dot{-} 2 < n)$$

III.2 Opakovanie: Známe diskriminácie

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

$$\cdots \leftarrow s \leq t \qquad \cdots \leftarrow s = t$$
$$\cdots \leftarrow s > t \qquad \cdots \leftarrow s \neq t$$

Vzájomne výlučné, pokrývajúce všetky prípady

$$\neg(s \leq t \wedge s > t) \qquad \neg(s = t \wedge s \neq t)$$
$$(s \leq t \vee s > t) \qquad (s = t \vee s \neq t)$$

Obe doterajšie diskriminácie testujú známe hodnoty.

6. Priraďujúce diskriminácie

6.1. Priradenie

III.3 Priradenie

CL umožňuje *priradiť* hodnotu výrazu *novej* premennej

$$\cdots \leftarrow \cdots \wedge t = x$$

Nová premenná na *pravej strane* rovnosti (ako v query), t je ľubovoľný výraz so známou hodnotou (zľava doprava)

Diskriminácia s jediným prípadom (vždy pravdivým)
Vždy platí:

$$\exists x(t = x)$$

Príklady

$n! = 1 \quad \leftarrow n = 0$
$n! = n \cdot m! \leftarrow n \neq 0 \wedge n \dot{-} 1 = m$

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

6.2. Monadická diskriminácia

III.4 Monadická diskriminácia – porovnanie so vzorom

Monadická diskriminácia kombinuje test a priradenie

píšeme	namiesto
$\cdots \leftarrow t = 0$	$\cdots \leftarrow t = 0$
$\cdots \leftarrow t = x + 1$	$\cdots \leftarrow t \neq 0 \wedge t \dot{-} 1 = x$

x je *nová* premenná, t je ľubovoľný výraz

Podmienky sú vzájomne výlučné a pokrývajú všetky prípady:

$$\neg(t = 0 \wedge \exists x(t = x + 1)) \qquad (t = 0 \vee \exists x(t = x + 1))$$

Rovnosť $t = x + 1$ (resp. $t = 0$) – *porovnanie so vzorom*
Výraz $x + 1$ (resp. 0) – *vzor*

Príklad použitia:

$$n! = 1 \quad \leftarrow n = 0$$
$$n! = n \cdot m! \leftarrow n = m + 1$$

III.5 Monadická diskriminácia – dosadenie

Vzory monadickej diskriminácie

$$n! = 1 \quad \leftarrow n = 0$$
$$n! = n \cdot m! \leftarrow n = m + 1$$

môžeme *dosadiť* namiesto premennej na ľavej strane rovnosti:

$$0! = 1$$
$$(m + 1)! = (m + 1) \cdot m!$$

Monadickú diskrimináciu dosádzajte, iba ak je prvá
Dosaďte buď oba prípady alebo žiadny

Zovšeobecnená monadická diskriminácia – skratka kaskády:

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

x je *nová* premenná, k je *konštanta*, t je ľubovoľný výraz

Príklad bez dosadenia a s dosadením:

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

7. Simulácia iterácie chvostovou rekurziou

III.7 Iterácia

- Ako by ste programovali výpočet $n!$ v Pascale?
  - Napríklad `while` cyklom

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

  - Cyklus opakuje – *iteruje* výpočet vo svojom tele

    Cyklus = *iterácia*

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

7.1. Simulácia `while` cyklu

III.8 Deklaratívna simulácia `while` cyklu

1. Pascalovský program rozdelíme na 2 časti:
   a) inicializácia

```
f := 1;
```

   b) cyklus a vrátenie výsledku

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

III.9 Deklaratívna simulácia `while` cyklu

2. Simulácia cyklu pomocnou funkciou While_fact:
   - Dva argumenty – simulujú pascalovské prepisovateľné premenné n a f
   - Keď $n \neq 0$,
     - vykoná sa telo cyklu, teda premenné f a n dostanú nové hodnoty;
     - cyklus sa zopakuje s novými hodnotami.
   - Keď $n = 0$,
     - funkcia skončí a vráti hodnotu z premennej f.

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

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

Invariant:

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

III.10 Deklaratívna simulácia `while` cyklu

3. Inicializácia a spustenie cyklu
   - Hlavná funkcia Fact spustí simuláciu cyklu s počiatočnými hodnotami premenných n a f.

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

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

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

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

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

Stručnejšie a elegantnejšie s monadickou diskrimináciou:

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

Priebeh výpočtu obyčajnej definície $n!$:

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

Priebeh výpočtu Fact:

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

While_fact: *chvostová* rekurzia

- Žiadne výpočty po návrate z rekurzívneho volania
- Nepotrebuje zásobník

III.13 Deklaratívna simulácia `while` cyklu

Cyklus ste simulovali už na minulom cvičení pri programovaní funkcie

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

- Aký cyklus ste simulovali?
- Ako ste funkciu Sqrt_0 naprogramovali?
- Akú vlastnosť (invariant) má pomocná funkcia simulujúca cyklus?

7.2. Efektívny výpočet Fibonacciho postupnosti

III.14 Fibonacciho postupnosť *ne*efektívne

Na pripomenutie: n-tý prvok Fibonacciho postupnosti

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

Výpočet obyčajnou rekurziou je veľmi neefektívny:

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

Zhruba fib_{n+2} rekurzívnych volaní
fib_n rastie ako ϕ^n, $\phi = (\sqrt{5}-1)/2 \doteq 1.618$ – exponenciálne

III.15 Fibonacciho postupnosť a králiky

Efektívny výpočet fib_n:

- pamätáme si dva prvky postupnosti
- počítame nasledujúci prvok

  $\Rightarrow$ stačí $n \dot{-} 1$ 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
0	1	$0 = \text{fib}_0$
1	$0 = \text{fib}_0$	$1 = \text{fib}_1$
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	$3 = \text{fib}_4$	$5 = \text{fib}_5$
6	$5 = \text{fib}_5$	$8 = \text{fib}_6$

rok	mláďatá	dospelí
$i{+}1$	$m{=}\text{fib}_i$	$d{=}\text{fib}_{i+1}$
$i{+}2$	$d{=}\text{fib}_{i+1}$	$m{+}d{=}\text{fib}_{i+2}$

III.16 Fibonacciho postupnosť efektívne v Pascale

Efektívny výpočet v Pascale:

```
function Fib(n: Integer): Integer;
var m, d, m1, d1: Integer;
begin
    if n = 0 then
        Result := 0
    else begin
        n := n - 1;
        m := 0; { fib(0) }
        d := 1; { fib(1) }
```

```
            while n <> 0 do begin
                { m = fib(i) & d = fib(i+1) }
                m1 := d;
                d1 := m + d;
                m := m1; d := d1;
                { m = fib(i+1) & d = fib(i+2) }
                n := n - 1
            end;
            Result := d
        end
    end;
```

III.17 Fibonacciho postupnosť efektívne v CL

Efektívny výpočet v CL:

- Simulácia `while` cyklu:

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

  Stručnejšie

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

  Invariant:

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

- Úvodné testy, inicializácia a spustenie cyklu:

  $$\begin{aligned}\mathrm{Fib}(n) &= 0 \qquad\qquad\qquad \leftarrow n = 0 \\ \mathrm{Fib}(n) &= \mathrm{While\_fib}(n \dot{-} 1, 0, 1) \leftarrow n \neq 0\end{aligned}$$

  Stručnejšie

  $$\begin{aligned}\mathrm{Fib}(0) &= 0 \\ \mathrm{Fib}(n+1) &= \mathrm{While\_fib}(n, 0, 1)\end{aligned}$$

7.3. Simulácia `for` cyklu

III.18 Simulácia `for` cyklu

Výpočet faktoriálu `for` cyklom:

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

Deklaratívna simulácia:

- Simulácia `for` cyklu:

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

  Spätná rekurzia – rekurzívna premenná *i rastie*, kým neprekročí vopred danú hranicu (n)

- Inicializácia premenných a štart cyklu:

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

III.19 Spätná vs. primitívna rekurzia

Spätná rekurzia, napríklad

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

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

IV. prednáška
Dyadická číselná sústava
Dyadické reťazce

5. marca 2012

7.4. Opakovanie

IV.1 Opakovanie: Priraďujúce diskriminácie

Priradenie

Premenná ako skratka za hodnotu výrazu

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

Monadická diskriminácia

Kombinácia testu na 0 a priradenia s odčítaním 1

Zapísaná formou porovnania so vzorom

$$\begin{aligned} \mathsf{F}(n) &= 1 && \leftarrow n = 0 \\ \mathsf{F}(n) &= n \cdot \mathsf{F}(m) && \leftarrow n = m+1 \end{aligned} \quad \Big| \quad \begin{aligned} \mathsf{F}(0) &= 1 \\ \mathsf{F}(m+1) &= (m+1) \cdot \mathsf{F}(m) \end{aligned}$$

Zovšeobecnená monadická diskriminácia

Skratka kaskády monadických diskriminácií

$$\begin{aligned} \mathsf{F}(n) &= 0 && \leftarrow n = 0 \\ \mathsf{F}(n) &= 1 && \leftarrow n = 1 \\ \mathsf{F}(n) &= \mathsf{F}(m+1) + \mathsf{F}(m) && \leftarrow n = m+2 \end{aligned} \quad \Big| \quad \begin{aligned} \mathsf{F}(0) &= 0 \\ \mathsf{F}(1) &= 1 \\ \mathsf{F}(n+2) &= \mathsf{F}(n+1) + \mathsf{F}(n) \end{aligned}$$

Chvostová rekurzia

$$\begin{aligned} F(x, y, z) &= \cdots && \leftarrow \cdots \\ F(x, y, z) &= \emptyset F(\ldots, \ldots, \ldots)\emptyset && \leftarrow \cdots \end{aligned}$$

- 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

`While` cyklus ∼ chvostová rek. riadená podmienkou cyklu

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

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

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

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

`For` cyklus $\sim$ spätná chvostová rek.

- premenné v cykle $\sim$ argumenty funkcie
- inicializácia $\sim$ volanie simulačnej funkcie hlavnou funkciou

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

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

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

8. Dyadická číselná sústava

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

$$17 = \mathbf{1}{\cdot}2^4 + \mathbf{0}{\cdot}2^3 + \mathbf{0}{\cdot}2^2 + \mathbf{0}{\cdot}2^1 + \mathbf{1}{\cdot}2^0 = (10001)_\text{b}$$

- Vo všeobecnosti

$$\sum_{i=0}^{k} b_i{\cdot}2^i = (b_k b_{k-1} \ldots b_1 b_0)_\text{b}$$

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

$$\begin{aligned}&(b_k b_{k-1} \dots b_1 b_0)_\mathrm{b}\\ =\ &(0 b_k b_{k-1} \dots b_1 b_0)_\mathrm{b} =\\ =\ &(00 b_k b_{k-1} \dots b_1 b_0)_\mathrm{b} = \cdots\end{aligned}$$

IV.6 Dyadická číselná sústava

- Každé prirodzené číslo sa dá *jednoznačne* zapísať v *dyadickej* sústave ako postupnosť začínajúca 0 a pokračujúca konečným počtom číslic **1** a **2**
- Napríklad
$$17 = \mathbf{1}{\cdot}2^3 + \mathbf{1}{\cdot}2^2 + \mathbf{2}{\cdot}2^1 + \mathbf{1}{\cdot}2^0 = 0\mathbf{1121}$$
- Vo všeobecnosti
$$\sum_{i=1}^{k} d_i{\cdot}2^{i-1} = 0 d_k \dots d_2 d_1, \quad d_i \in \{1, 2\}$$
- V Hornerovej schéme:
$$17 = (((0{\cdot}2 + \mathbf{1}){\cdot}2 + \mathbf{1}){\cdot}2 + \mathbf{2}){\cdot}2 + \mathbf{1}$$
$$\sum_{i=1}^{k} d_i{\cdot}2^{i-1} = ((\cdots(0{\cdot}2 + d_k)\cdots){\cdot}2 + d_2){\cdot}2 + d_1$$

8.1. Dyadické konštruktory

IV.7 Dyadické konštruktory

- Zadefinujme
$$S_1(n) = 2{\cdot}n + 1 \qquad\qquad S_2(n) = 2{\cdot}n + 2$$

- Potom môžeme zapísať

$$\begin{aligned} 17 &= (((0{\cdot}2+\mathbf{1}){\cdot}2+\mathbf{1}){\cdot}2+\mathbf{2}){\cdot}2+\mathbf{1} \\ &= \mathsf{S}_1(((0{\cdot}2+\mathbf{1}){\cdot}2+\mathbf{1}){\cdot}2+\mathbf{2}) \\ &= \mathsf{S}_1\,\mathsf{S}_2((0{\cdot}2+\mathbf{1}){\cdot}2+\mathbf{1}) \\ &= \mathsf{S}_1\,\mathsf{S}_2\,\mathsf{S}_1(0{\cdot}2+\mathbf{1}) \\ &= \mathsf{S}_1\,\mathsf{S}_2\,\mathsf{S}_1\,\mathsf{S}_1(0) = 0\mathbf{1121} \end{aligned}$$

- Vo všeobecnosti

$$\begin{aligned} \sum_{1\le i\le k} d_i{\cdot}2^i &= ((\cdots(0{\cdot}2+d_k)\cdots){\cdot}2+d_2){\cdot}2+d_1 \\ &= \mathsf{S}_{d_1}(\cdots(0{\cdot}2+d_k)\cdots){\cdot}2+d_2) \\ &= \mathsf{S}_{d_1}\,\mathsf{S}_{d_2}(\cdots(0{\cdot}2+d_k)\cdots) \\ &= \mathsf{S}_{d_0}\,\mathsf{S}_{d_1}\cdots\mathsf{S}_{d_k}(0) = 0d_k\ldots d_2d_1 \end{aligned}$$

- Takže: $\mathsf{S}_1(n) = n\mathbf{1}$ $\mathsf{S}_2(n) = n\mathbf{2}$
- Funkcie S_1 a S_2 nazývame *dyadické konštruktory*

8.2. Dyadická diskriminácia

IV.8 Testovanie dyadického zápisu

Dané je číslo $n = 0d_k\ldots d_2d_1 = \sum_{i=1}^{k} d_i{\cdot}2^{i-1}$.

1. Kedy nemá n žiadnu dyadickú číslicu?
   - $n = 0$
2. Kedy má n aspoň jednu dyadickú číslicu?
   - $n \neq 0$

   a) Ako potom určíme najmenej významnú číslicu čísla n?
      - $\mathsf{Last}_2(n) = d_1 = 2 \dot{-} (n \bmod 2)$

   b) Ako určíme číslo, ktoré vznikne z n odobratím najmenej významnej číslice?
      - $\mathsf{Init}_2(n) = 0d_k\ldots d_2 = \sum_{i=1}^{k-1} d_{i+1}{\cdot}2^{i-1} = (n \dot{-} 1) \div 2$

Naprogramujme zmenu poslednej dyadickej číslice:

- $\mathsf{Flip\_last}_2(n) = 0 \quad \leftarrow n = 0$
  $\mathsf{Flip\_last}_2(n) = \mathsf{S}_2(m) \leftarrow n \neq 0 \wedge \mathsf{Init}_2(n) = m \wedge \mathsf{Last}_2(n) = 1$
  $\mathsf{Flip\_last}_2(n) = \mathsf{S}_1(m) \leftarrow n \neq 0 \wedge \mathsf{Init}_2(n) = m \wedge \mathsf{Last}_2(n) = 2$

V CL existuje elegantnejší spôsob...

IV.9 Dyadická diskriminácia

CL pozná *dyadickú diskrimináciu*:

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

$$\begin{array}{ll} \cdots \leftarrow t = 0 & \cdots \leftarrow t = 0 \\ \cdots \leftarrow t = m\mathbf{1} & \cdots \leftarrow t = \mathsf{S}_1(m) \\ \cdots \leftarrow t = m\mathbf{2} & \cdots \leftarrow t = \mathsf{S}_2(m) \end{array}$$

t je výraz so známou hodnotou, m je nová premenná

Tieto prípady pokrývajú všetky možnosti

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

a sú vzájomne výlučné:

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

Príklad použitia:

v tele klauzuly	v argumente funkcie
$\mathsf{Flip\_last}_2(n) = 0 \quad \leftarrow n = 0$	$\mathsf{Flip\_last}_2(0) = 0$
$\mathsf{Flip\_last}_2(n) = m\mathbf{2} \leftarrow n = m\mathbf{1}$	$\mathsf{Flip\_last}_2(m\mathbf{1}) = m\mathbf{2}$
$\mathsf{Flip\_last}_2(n) = m\mathbf{1} \leftarrow n = m\mathbf{2}$	$\mathsf{Flip\_last}_2(m\mathbf{2}) = m\mathbf{1}$

8.3. Aritmetické operácie na dyadickom zápise čísel

IV.10 Dyadická aritmetika

- Pomocou dyadického zápisu môžeme zadefinovať *efektívne* aritmetické operácie na veľkých číslach
  - Počet krokov potrebných na výpočet operácie závisí od počtu číslic, nie od hodnoty čísel
- Podobný princíp ako aritmetika na desiatkovom zápise – základná škola
- Hľadáme vyjadrenie aritmetických operácií *iba* pomocou
  - dyadickej diskriminácie
  - dyadických konštruktorov a 0
  - rekurzie *s menším počtom dyadických číslic*
  - použitia už definovaných operácií

IV.11 Nasledovník na dyadickom zápise

Začnime pripočítaním 1: funkcia $\mathrm{Succ}(n) = n+1$ (successor – nasledovník)

n	0	0	1	01	2	02	3	011	4	012	5	021	6	022	$\cdots$
$n+1$	1	01	2	02	3	011	4	012	5	021	6	022	7	0111	$\cdots$

Príklady:

$$\begin{array}{r} 0 \\ +\ 01 \\ \hline 01 \end{array} \qquad \begin{array}{r} 021 \\ +\ 01 \\ \hline 022 \end{array} = 5 \qquad \begin{array}{r} 021\,2 \\ +\ 0\,1 \\ \hline 021 \\ +\ 01\,1 \\ \hline 022\,1 \end{array} = 12 = 13$$

Zovšeobecnenie:

$$\begin{array}{r} 0 \\ +\ 01 \\ \hline 01 \end{array} \qquad \begin{array}{r} n1 \\ +\ 01 \\ \hline n2 \end{array} \qquad \begin{array}{r} n\,2 \\ +\ 0\,1 \\ \hline n \\ +\ 01\,1 \\ \hline (n+1)\,1 \end{array}$$

IV.12 Nasledovník na dyadickom zápise

Algebraické odvodenie:

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

Klauzálna definícia použitím dyadickej diskriminácie:

$$\begin{array}{lll} \mathrm{Succ}(n) = 0\mathbf{1} & \leftarrow n = 0 & \mathrm{Succ}(0) = 0\mathbf{1} \\ \mathrm{Succ}(n) = m\mathbf{2} & \leftarrow n = m\mathbf{1} & \mathrm{Succ}(m\mathbf{1}) = m\mathbf{2} \\ \mathrm{Succ}(n) = (\mathrm{Succ}(m))\mathbf{1} & \leftarrow n = m\mathbf{2} & \mathrm{Succ}(m\mathbf{2}) = (\mathrm{Succ}(m))\mathbf{1} \end{array}$$

Akú rekurziu sme použili?

- Všeobecnú (course-of-values), lebo $m < m\mathbf{2}$

Tento typ nazývame aj *dyadická rekurzia*

- Argument rekurzívneho volania (m) má *menší počet dyadických číslic* ako pôvodný argument ($m\mathbf{2}$)

IV.13 Sčítanie na dyadickom zápise

Príklady sčítania:

$$\begin{array}{r} 0 \\ +\ 0\mathbf{21} \\ \hline 0\mathbf{21} \end{array} \qquad \begin{array}{r} 0\mathbf{21} \\ +\ 0\mathbf{111} \\ \hline 0\mathbf{212} \end{array} \qquad \begin{array}{r} 0\mathbf{212} \\ +\ 0\mathbf{12} \\ \hline 0\mathbf{1112} \end{array}$$

Zovšeobecnenie:

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

$$\begin{array}{r} x\mathbf{1} \\ +\ 0 \\ \hline x\mathbf{1} \end{array} \qquad \begin{array}{r} x\mathbf{1} \\ +\ y\mathbf{1} \\ \hline (x+y)\mathbf{2} \end{array} \qquad \begin{array}{r} x\mathbf{1} \\ +\ y\mathbf{2} \\ \hline ? \end{array}$$

$$\begin{array}{r} x\mathbf{2} \\ +\ 0 \\ \hline x\mathbf{2} \end{array} \qquad \begin{array}{r} x\mathbf{2} \\ +\ y\mathbf{1} \\ \hline ? \end{array} \qquad \begin{array}{r} x\mathbf{2} \\ +\ y\mathbf{2} \\ \hline ? \end{array}$$

IV.14 Sčítanie na dyadickom zápise

Algebraické odvodenie:

$$\begin{aligned}
0 + y &= y \\
x\mathbf{1} + 0 &= x\mathbf{1} \\
x\mathbf{1} + y\mathbf{1} &= (2{\cdot}x + 1) + (2{\cdot}y + 1) = 2{\cdot}(x + y) + 2 = (\,x + y\,)\mathbf{2} \\
x\mathbf{1} + y\mathbf{2} &= (2{\cdot}x + 1) + (2{\cdot}y + 2) = (2{\cdot}x + 2{\cdot}y + 2) + 1 = \\
&= 2{\cdot}(x + y + 1) + 1 = (\,(\,x + y\,) + 1\,)\mathbf{1} \\
&\dots
\end{aligned}$$

Klauzálna definícia použitím dyadickej rekurzie a dvojitej dyadickej diskriminácie:

$$\begin{aligned}
&\mathrm{Add}(0, y) &&= y \\
&\mathrm{Add}(x\mathbf{1}, 0) &&= x\mathbf{1} \\
&\mathrm{Add}(x\mathbf{1}, y\mathbf{1}) &&= (\,\mathrm{Add}(x, y)\,)\mathbf{2} \\
&\mathrm{Add}(x\mathbf{1}, y\mathbf{2}) &&= (\,\mathrm{Succ}(\,\mathrm{Add}(x, y)\,)\,)\mathbf{1} \\
&\mathrm{Add}(x\mathbf{2}, 0) &&= \cdots \\
&\mathrm{Add}(x\mathbf{2}, y\mathbf{1}) &&= \cdots \mathrm{Add}(x, y) \cdots \\
&\mathrm{Add}(x\mathbf{2}, y\mathbf{2}) &&= \cdots \mathrm{Add}(x, y) \cdots
\end{aligned}$$

IV.15 Násobenie na dyadickom zápise

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

$$\begin{aligned}
2{\cdot}0 &= 0 \\
2{\cdot}(x\mathbf{1}) &= 2{\cdot}(2{\cdot}x + 1) = \cdots \\
2{\cdot}(x\mathbf{2}) &= \cdots \quad \text{(bez rekurzie)}
\end{aligned}$$

Algebraické odvodenie násobenia:

$$\begin{aligned}
0{\cdot}y &= 0 \\
x\mathbf{1}{\cdot}y &= (2{\cdot}x + 1){\cdot}y = \cdots \\
x\mathbf{2}{\cdot}y &= (2{\cdot}x + 2){\cdot}y = \cdots
\end{aligned}$$

Klauzálna definícia násobenia:

$$\begin{aligned}\mathsf{Mul}(0, y) &= 0\\ \mathsf{Mul}(x\mathbf{1}, y) &= \cdots \mathsf{Mul}(x, y) \cdots\\ \mathsf{Mul}(x\mathbf{2}, y) &= \cdots \mathsf{Mul}(x, y) \cdots\end{aligned}$$

8.4. Symbolické operácie na dyadickom zápise čísel

IV.16 Symbolický pohľad na dyadický zápis

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

  0 prázdny reťazec

  $n\mathbf{1}$ neprázdny reťazec končiaci symbolom 1

  $n\mathbf{2}$ neprázdny reťazec končiaci symbolom 2

- Na dyadických reťazcoch môžeme vykonávať symbolické operácie:
  - Zistenie dĺžky reťazca (počtu znakov)
  - Zreťazenie dvoch reťazcov
  - Obrátenie reťazca
  - Vyhľadanie podreťazca
  - ...

IV.17 Dĺžka dyadického reťazca

Dĺžku reťazca vypočítame dyadickou rekurziou veľmi jednoducho:

$$\begin{aligned}\mathsf{Len}(0) &= 0\\ \mathsf{Len}(x\mathbf{1}) &= \mathsf{Len}(x) + 1\\ \mathsf{Len}(x\mathbf{2}) &= \mathsf{Len}(x) + 1\end{aligned}$$

$$\begin{aligned}\mathsf{Len}(0\mathbf{1}\mathbf{2}\mathbf{2}\mathbf{1}) &= \mathsf{Len}(0\mathbf{1}\mathbf{2}\mathbf{2}) + 1 =\\ &= \mathsf{Len}(0\mathbf{1}\mathbf{2}) + 1 + 1 =\\ &= \mathsf{Len}(0\mathbf{1}) + 1 + 1 + 1 =\\ &= \mathsf{Len}(0) + 1 + 1 + 1 + 1 =\\ &= 0 + 1 + 1 + 1 + 1 = 4\end{aligned}$$

IV.18 Zreťazenie dyadických reťazcov

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

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

$$0d_{n+k} \ldots d_{2+k} d_{1+k} \star 0 d_k \ldots d_2 d_1 = 0 d_{n+k} \ldots d_{2+k} d_{1+k} d_k \ldots d_2 d_1$$

Ako vybrať rekurzívny argument?

- Porovnajme očakávané výsledky po odobratí posledného znaku:
  - z prvého argumentu:

$$\begin{aligned} 0211\,2 \star 0221 &= 0211\,2\,221 \\ 0211 \star 0221 &= 0211221 \end{aligned}$$

  - z druhého argumentu:

$$\begin{aligned} 02112 \star 022\,1 &= 0211222\,1 \\ 02112 \star 022 &= 0211222 \end{aligned}$$

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

IV.19 Zreťazenie dyadických reťazcov – riešenie

$$\begin{aligned} x \star 0 &= x \\ x \star (y1) &= (x \star y)1 \\ x \star (y2) &= (x \star y)2 \end{aligned}$$

$$\begin{aligned} 02112 \star 0221 &= (02112 \star 022)1 \\ &= (02112 \star 02)21 \\ &= (02112 \star 0)221 \\ &= (02112)221 \\ &= 02112221 \end{aligned}$$

IV.20 Obrátenie dyadického reťazca

Rev(x) je dyadický reťazec, ktorý obsahuje všetky znaky reťazca x v obrátenom poradí

$$\text{Rev}(0\mathbf{1121}) = 0\mathbf{1211}$$
$$\text{Rev}(0d_k \dots d_2 d_1) = 0d_1 d_2 \dots d_k$$

Ako nájsť rekurzívnu definíciu Rev?

- Porovnajme očakávané výsledky po odobratí posledného znaku:

$$\text{Rev}(\ 0\mathbf{112}\,\mathbf{1}\ ) = \ 0\mathbf{1}\,\mathbf{211}$$
$$\text{Rev}(\ 0\mathbf{112}\ \ ) = \ \ 0\mathbf{211}$$

  Ako dostaneme posledný znak na *začiatok* výsledku rekurzívneho volania?

$\text{Rev}(0) \ = 0$
$\text{Rev}(x\mathbf{1}) = \cdots \text{Rev}(x) \cdots$
$\text{Rev}(x\mathbf{2}) = \cdots \text{Rev}(x) \cdots$

V. prednáška
Párovanie. Predikáty. Celočíselná aritmetika

12. marca 2012

9. Párovanie

9.1. Kódovanie dátových štruktúr

V.1 Dátové štruktúry

- Programy zriedka operujú iba na číslach
- Zvyčajne používame dátové štruktúry, napríklad ...
- Funkcie v CL pracujú iba s prirodzenými číslami (s ľubovoľnou presnosťou)
- Minulý týždeň sme pomocou dyadického zápisu *zakódovali* do prirodzených čísel dátovú štruktúru: reťazce symbolov 1 a 2
- Najjednoduchšia dátová štruktúra: *usporiadaná dvojica*
  - Umožňuje vybudovať väčšinu prakticky potrebných dátových štruktúr (uvidíme neskôr)
- V Pascale: ...?
- Ako zakódujeme usporiadané dvojice v CL?

9.2. Párovacie funkcie

V.2 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 dekódovať dvojicu
- $p(x, y)$ je teda *injektívna* funkcia z $\mathbb{N} \times \mathbb{N}$ do $\mathbb{N}$
- Výhodné je, ak niektorému číslu (napr. 0) nie je priradená dvojica
  - také číslo je *atóm*
  - môže predstavovať `nil`

  a všetky ostatné čísla predstavujú dvojice
- $p(x, y)$ je teda *surjektívna* funkcia z $\mathbb{N} \times \mathbb{N}$ na $\mathbb{N} \setminus \{0\}$
- Súhrnne $p \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N} \setminus \{0\}$ je *bijektívna* funkcia

V.3 Párovacie funkcie

Párovacou funkciou nazývame každé zobrazenie $p \colon \mathbb{N}^2 \to \mathbb{N} \setminus \{0\}$, pre ktoré platí:

- injektívnosť:

$$p(x, y) = p(u, v) \to x = u \wedge y = v$$

- surjektívnosť:

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

- pár je väčší ako jeho zložky:

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

V.4 Párovacie funkcie – príklady

- Párovacích funkcií je nekonečne veľa
- Napríklad:

– modifikovaná Cantorova funkcia

$x \backslash y$	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	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

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

– modifikované „zipsovanie“ binárnych zápisov

$x \backslash y$	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	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

$$p'_b(x,y) = p_b(x,y)+1$$
$$p_b(x,y) = 0 \leftarrow x+y = 0$$
$$p_b(x,y) = 4\cdot p_b(x \div 2, y \div 2) + 2\cdot(y \bmod 2) + x \bmod 2 \leftarrow x+y \neq 0$$

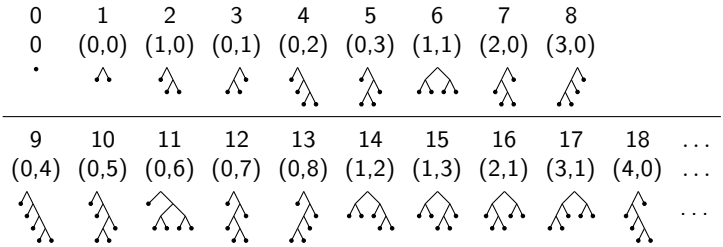
9.2.1. Párovacia funkcia CL

V.5 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 $s = u, v$ a
    - $x < u$ alebo
    - $x = u$ a $y < v$.

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



- Fragment tabuľky párovacej funkcie (,):

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

- 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 $x = 0$

– $\mathsf{T}(x)$ vráti druhú zložku (chvost, tail) páru x alebo 0 ak $x = 0$

Platí teda:

$$H(0) = 0 \qquad H(u, v) = u$$
$$T(0) = 0 \qquad T(u, v) = v$$

V.8 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:

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

9.2.2. Programovanie s párovacou funkciou

V.9 Programovanie s párovacou funkciou

- Pár vytvoríme výrazom „s, t“
- Párová diskriminácia:

$$\cdots \leftarrow t = 0$$
$$\cdots \leftarrow t = u, v$$

Druhý prípad priradí zložky páru do nových premenných u a v

- Projekcie H a T nebudeme používať, párová diskriminácia je skratkou za:

$$\cdots \leftarrow t = 0$$
$$\cdots \leftarrow t \neq 0 \wedge \mathsf{H}(t) = u \wedge \mathsf{T}(t) = v$$

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

  $\text{Twofib}(0) = 0, 1$
  $\text{Twofib}(n+1) = 0 \qquad \leftarrow \text{Twofib}(n) = 0$
  $\text{Twofib}(n+1) = b, a+b \leftarrow \text{Twofib}(n) = a, b$

- Prípad v druhej klauzule nemôže nastať, môžeme ju vynechať

  CL si výsledok 0 doplní *default*om

  $\text{Twofib}(0) = 0, 1$
  $\text{Twofib}(n+1) = b, a+b \leftarrow \text{Twofib}(n) = a, b$

- Defaulty slúžia pre prípady, keď výpočet nemá zmysel

  Nevynechávajte klauzuly, keď funkcia vracia 0 ako riadny, očakávaný výsledok!

$\text{Twofib}(0) = 0, 1$
$\text{Twofib}(n+1) = b, a+b \leftarrow \text{Twofib}(n) = a, b$

- Výpočet:

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

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

  V pamäti vytvorí pascalovský `record`

9.2.3. Tupling

V.12 Tupling

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

  Technika sa nazýva *tupling*

- Napríklad celočíselné delenie a zvyšok:

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

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

  spojíme do jednej funkcie

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

V.13 Tupling

- Spojená funkcia:

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

9.3. Predikáty

V.14 Predikáty

- *Predikát* je vlastnosť objektu alebo n-tice objektov
  - x je prázdny dyadický reťazec

$$\mathrm{Empty}_2(x) \leftrightarrow x = 0$$

  - dyadický reťazec x je palindróm

$$\mathrm{Palindrome}_2(x) \leftrightarrow x = \mathrm{Rev}(x)$$

  - s je sufixom (koncovým úsekom) dyadického reťazca x

$$\mathrm{Suffix}_2(s, x) \leftrightarrow \exists r(x = r \star s)$$

  - číslo x kóduje dvojicu čísel

$$\mathrm{Pair}(x) \leftrightarrow \exists u \exists v(x = u, v)$$

  ...

V.15 Charakteristické funkcie

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

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

- Napríklad

$\mathrm{Palindrome}_{2*}(x) = 1 \leftarrow x = \mathrm{Rev}(x)$
$\mathrm{Palindrome}_{2*}(x) = 0 \leftarrow x \neq \mathrm{Rev}(x)$

$\mathrm{Suffix}_{2*}(0, x) = 1$
$\mathrm{Suffix}_{2*}(s\mathbf{1}, 0) = 0$
$\mathrm{Suffix}_{2*}(s\mathbf{1}, x\mathbf{1}) = 0 \leftarrow \mathrm{Suffix}_{2*}(s, x) = 0$
$\mathrm{Suffix}_{2*}(s\mathbf{1}, x\mathbf{1}) = 1 \leftarrow \mathrm{Suffix}_{2*}(s, x) = 1$
$\mathrm{Suffix}_{2*}(s\mathbf{1}, x\mathbf{2}) = 0$
$\mathrm{Suffix}_{2*}(s\mathbf{2}, 0) = 0$
$\mathrm{Suffix}_{2*}(s\mathbf{2}, x\mathbf{1}) = 0$
$\mathrm{Suffix}_{2*}(s\mathbf{2}, x\mathbf{2}) = 0 \leftarrow \mathrm{Suffix}_{2*}(s, x) = 0$
$\mathrm{Suffix}_{2*}(s\mathbf{2}, x\mathbf{2}) = 1 \leftarrow \mathrm{Suffix}_{2*}(s, x) = 1$

V.16 Definovanie predikátov v CL

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

$\mathrm{Palindrome}_2(x) \leftarrow x = \mathrm{Rev}(x)$
~~$\neg\mathrm{Palindrome}_2(x) \leftarrow x \neq \mathrm{Rev}(x)$~~

$\mathrm{Suffix}_2(0, x)$
~~$\neg\mathrm{Suffix}_2(s\mathbf{1}, 0)$~~
~~$\neg\mathrm{Suffix}_2(s\mathbf{1}, x\mathbf{1}) \leftarrow \neg\mathrm{Suffix}_2(s, x)$~~
$\mathrm{Suffix}_2(s\mathbf{1}, x\mathbf{1}) \leftarrow \mathrm{Suffix}_2(s, x)$
~~$\neg\mathrm{Suffix}_2(s\mathbf{1}, x\mathbf{2})$~~
~~$\neg\mathrm{Suffix}_2(s\mathbf{2}, 0)$~~
~~$\neg\mathrm{Suffix}_2(s\mathbf{2}, x\mathbf{1})$~~
~~$\neg\mathrm{Suffix}_2(s\mathbf{2}, x\mathbf{2}) \leftarrow \neg\mathrm{Suffix}_2(s, x)$~~
$\mathrm{Suffix}_2(s\mathbf{2}, x\mathbf{2}) \leftarrow \mathrm{Suffix}_2(s, x)$

V.17 Diskriminácia na platnosť predikátu

- Diskriminácia na platnosť predikátu

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

- $\neg P(x)$ zapisujeme `~P(`x`)`

VI. prednáška
Celočíselná aritmetika. Opakovanie

19. marca 2012

9.4. Kódovanie celých čísel párovaním

VI.1 Celé čísla ako triedy usporiadaných dvojíc

- Na DM ste videli nasledujúci *rozklad* množiny $\mathbb{N}^2$:

$$\mathcal{Z} = \{\, [(a, b)] \mid a \in \mathbb{N} \wedge b \in \mathbb{N} \,\},$$

  kde

$$[(a, b)] = \{\, (c, d) \in \mathbb{N}^2 \mid a + d = c + b \,\}$$

- Všimnite si, že platí

$$(c, d) \in [(a, b)] \leftrightarrow c - d = a - b,$$

  kde „$-$“ označuje odčítanie na $\mathbb{R}$ (nie modifikované odčítanie $\dot{-}$ na $\mathbb{N}$)

- Rozklad $\mathcal{Z}$ je izomorfný so $\mathbb{Z}$

VI.2 Celé čísla ako triedy usporiadaných dvojíc

Rozklad $\mathcal{Z}$ je izomorfný so $\mathbb{Z}$

- Každá trieda rozkladu $[(a, b)]$ predstavuje jednoznačne nejaké celé číslo
  - Ktoré?
  - Ktorá trieda v rozklade $\mathcal{Z}$ predstavuje $0 \in \mathbb{Z}$?
- Na triedach rozkladu môžeme zadefinovať aritmetické operácie zodpovedajúce operáciám na $\mathbb{Z}$

- ▶ Zadefinujme napríklad sčítanie
  - Pre každé a, b, c, d, e, $f \in \mathbb{N}$ má platiť

$$[(a,b)] +_{\mathcal{Z}} [(c,d)] = [(e,f)] \leftrightarrow (a-b)+(c-d) = (e-f)$$

  - Takže:

$$[(a,b)] +_{\mathcal{Z}} [(c,d)] = [(\ldots,\ldots)]$$

- ▶ Všimnite si, že $+_{\mathcal{Z}} \colon \mathcal{Z} \times \mathcal{Z} \to \mathcal{Z}$ je operácia na triedach rozkladu $\mathcal{Z}$, teda na nekonečných množinách

VI.3 Celé čísla v CL

Ako by sme mohli pracovať s celými číslami v CL?

- Nemôžeme manipulovať s triedami rozkladu $\mathcal{Z}$
- Môžeme pracovať s ich *reprezentantmi* – konkrétnymi dvojicami čísel
- Dvojice zakódujeme párovaním

~~Z(0)~~
Z$(a,b) \leftarrow$ N$(a) \wedge$ N(b)

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

- Dvojice $(2, 4)$ a $(53, 55)$ sú rôzne, $(2, 4) \neq (53, 55)$, ale reprezentujú rovnaké celé číslo $2 - 4 = 53 - 55 = -2$
- Zadefinujme predikát $x =_{\mathsf{Z}} y$ ktorý platí práve vtedy, keď x a y reprezentujú to isté celé číslo

$$\neg(x =_{\mathsf{Z}} y) \leftarrow x = 0$$
$$\neg(x =_{\mathsf{Z}} y) \leftarrow x = (a, b) \wedge y = 0$$
$$x =_{\mathsf{Z}} y \leftarrow x = (a, b) \wedge y = (c, d) \wedge a + d = c + b$$
$$\neg(x =_{\mathsf{Z}} y) \leftarrow x = (a, b) \wedge y = (c, d) \wedge a + d \neq c + b$$

Stručnejšie

$$(a, b) =_{\mathsf{Z}} (c, d) \leftarrow a + d = c + b$$

- Zadefinujme funkciu $\mathsf{C}_{\mathsf{Z}}(x)$, ktorá pre ľubovoľného reprezentanta x vráti kanonického reprezentanta

$$\mathsf{C}_{\mathsf{Z}}(x) = 0 \leftarrow x = 0$$
$$\mathsf{C}_{\mathsf{Z}}(x) = (a \dot{-} b, 0) \leftarrow x = (a, b) \wedge a \geq b$$
$$\mathsf{C}_{\mathsf{Z}}(x) = (0, b \dot{-} a) \leftarrow x = (a, b) \wedge a < b$$

9.4.1. Operácie na celých číslach

- Na reprezentantoch (ľubovoľných, nie len kanonických) môžeme zadefinovať aritmetické operácie
- Zadefinujme operáciu $x +_{\mathsf{Z}} y$
- Hodnotou je reprezentant súčtu celých čísel, ktoré sú reprezentované argumentami sčítania:

$$(a, b) +_{\mathsf{Z}} (c, d) = (e, f) \leftrightarrow (a - b) + (c - d) = e - f$$

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

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

- Sčítanie teda má explicitnú klauzálnu definíciu

$$(a,b) +_{\mathsf{Z}} (c,d) = (a+c, b+d)$$

- Podobne môžeme špecifikovať a zadefinovať operácie

$$0_{\mathsf{Z}} \qquad -_{\mathsf{Z}} x \qquad x -_{\mathsf{Z}} y \qquad 1_{\mathsf{Z}} \qquad a \cdot_{\mathsf{Z}} x \qquad x \times_{\mathsf{Z}} y$$

9.4.2. Rozšírený euklidovský algoritmus

VI.6 Rozšírený euklidovský algoritmus

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

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

- Hľadáme funkciu

$$\begin{aligned}\mathsf{N}(a) \wedge \mathsf{N}(b) &\to \exists x \exists y \big(\mathsf{Z}(x) \wedge \mathsf{Z}(y) \wedge \overline{\gcd}(a,b) = (x,y)\big) \\ \overline{\gcd}(a,b) = (x,y) &\leftrightarrow a \cdot_{\mathsf{Z}} x +_{\mathsf{Z}} b \cdot_{\mathsf{Z}} y = \gcd(a,b) \cdot_{\mathsf{Z}} 1_{\mathsf{Z}}\end{aligned}$$

- Pripomenutie: euklidovský algoritmus

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

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

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

- Kontext:

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

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

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

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

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

- $a \neq 0 \implies \gcd(a,b) = \gcd(b \bmod a, a) \wedge b \bmod a < a$

  Predpokladajme, že $\overline{\gcd}(b \bmod a, a) = (x', y')$ a platí

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

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

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

- Ako zapíšeme tieto x a y v *našej celočíselnej aritmetike*?

Prémiová domáca úloha du05b:

- Odovzdajte súbor `ex05b.cl` s naprogramovaným $\overline{\gcd}$
- E-mailom na `udp@lists.dai.fmph.uniba.sk`
- Predmet správy: du05b
- Termín: piatok 23. marca 2012 o 12:00
- Hodnota: 2 body

10. Opakovanie

10.1. Chvostová rekurzia

VI.9 Opakovanie – chvostová rekurzia

- Funkcia je definovaná *chvostovou rekurziou*, ak sa jej hodnota v rekurzívnom prípade *rovná* hodnote rekurzívneho volania (*bez ďalších úprav*)
- Napríklad definícia

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

  je chvostovorekurzívna
- Definícia

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

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

  $$f(n) = g(n, 1)$$

VI.10 Opakovanie – chvostová rekurzia

Príklad: efektívny výpočet rekurentného vzťahu

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

  $$f(0) = 0$$
  $$f(1) = 1$$
  $$f(2) = 2$$
  $$f(n+3) = f(n+2) + f(n+1) + f(n)$$

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

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

VI.11 Opakovanie – chvostová rekurzia

Príklad: inverzná funkcia

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

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

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

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

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

10.2. Dyadická aritmetika

VI.12 Opakovanie – dyadický zápis čísel

- Dyadický zápis čísel konštruujeme z 0 konštruktormi

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

- Dyadický zápis čísel testujeme dyadickou diskrimináciou:

$$\begin{aligned} \cdots &\leftarrow t = 0 & \cdots &\leftarrow t = 0 \\ \cdots &\leftarrow t = m\mathbf{1} & \cdots &\leftarrow t = S_1(m) \\ \cdots &\leftarrow t = m\mathbf{2} & \cdots &\leftarrow t = S_2(m) \end{aligned}$$

VI.13 Opakovanie – dyadická aritmetika

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

$$
\begin{aligned}
0 + y &= y \\
x\mathbf{1} + 0 &= x\mathbf{1} \\
x\mathbf{1} + y\mathbf{1} &= (2{\cdot}x + 1) + (2{\cdot}y + 1) = 2{\cdot}(x + y) + 2 = (x + y)\mathbf{2} \\
x\mathbf{1} + y\mathbf{2} &= (2{\cdot}x + 1) + (2{\cdot}y + 2) = (2{\cdot}x + 2{\cdot}y + 2) + 1 = \\
&= 2{\cdot}(x + y + 1) + 1 = ((x + y) + 1)\mathbf{1} \\
&\dots
\end{aligned}
$$

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

10.3. Symbolické operácie na dyadickom zápise

VI.14 Opakovanie – dyadický zápis ako reťazec

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

  0 prázdny reťazec

  $n\mathbf{1}$ neprázdny reťazec končiaci symbolom 1

  $n\mathbf{2}$ neprázdny reťazec končiaci symbolom 2

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

VII. prednáška
Zoznamy.
Chvostová rekurzia a tupling na zoznamoch

26. marca 2012

11. Zoznamy

VII.1 Rekapitulácia

Doteraz sme videli dve kódovania dátových štruktúr:

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

Teraz si ukážeme kódovanie konečných postupností

11.1. Kódovanie konečných postupností

VII.2 Kódovanie konečných postupností

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

- Skúsme využiť vnorené párovanie:

$$x = 8, 2, 0, 4, 6 = 8, (2, (0, (4, 6))) = 7798222$$
$$y = 1, 3, 5, 7 \quad = 1, (3, (5, 7)) \qquad = 148217$$

- Dekódujme teraz postupnosť zakódovanú číslom x

  Dĺžku nepoznáme vopred

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

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

$$x = 8, 45576 = 8, 2, 830 = 8, 2, 0, 401 =$$
$$= 8, 2, 0, 4, 6 = 8, 2, 0, 4, 1, 1 = 8, 2, 0, 4, 1, 0, 0$$

- Nie každé číslo kóduje dvojicu:

  0

- Nie každé číslo kódujúce dvojicu kóduje trojicu:

  $x_1, 0$

  $\vdots$

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

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

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

  $$x_1, x_2, \ldots, x_n, 0$$

  Ukončenie 0 zaručí jednoznačné dekódovanie

  $$\begin{aligned} xs &= 8, 2, 0, 4, 6, 0 = 8, (2, (0, (4, (6, 0)))) \\ ys &= 1, 3, 5, 7, 0 \quad = 1, (3, (5, (7, 0))) \end{aligned}$$

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

  Premenenné obsahujúce zoznamy majú príponu -*s*

  $$xs \qquad ys \qquad zs \qquad \ldots$$

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

11.2. Programovanie so zoznamami

11.2.1. Zreťazenie

VII.6 Programovanie so zoznamami

Naprogramujme funkciu Conc(xs, ys), ktorá *zreťazí* zoznamy *xs* a *ys*

- ▶ Conc vytvorí *nový* zoznam, ktorý obsahuje najprv všetky prvky *xs* a potom všetky prvky *ys*

Ako hľadať riešenie?

Ako hľadať riešenie?

1. Načrtneme si príklad
   - $\mathrm{Conc}\big((1,2,3,0),(4,5,6,0)\big) = 1,2,3,4,5,6,0$
2. Uvedomíme si, že vstup je zoznam:
   - je buď *prázdny*: $xs = 0$
   - alebo má *prvý prvok a zvyšok*: $xs = x, zs$

   $\Rightarrow$ párová diskriminácia
3. Určíme výsledok pre prázdny zoznam
   - $\mathrm{Conc}\big(\underbrace{0}_{xs}, \underbrace{(4,5,6,0)}_{ys}\big) = \underbrace{4,5,6,0}_{ys}$
4. *Výsledok pre zvyšok zs* zoznamu *xs*
   - $\mathrm{Conc}\big(\underbrace{(2,3,0)}_{zs}, \underbrace{(4,5,6,0)}_{ys}\big) = 2,3,4,5,6,0$

   upravíme na výsledok pre *celý zoznam xs*
   - $\mathrm{Conc}\big(\underbrace{(\underbrace{1}_{x}, \underbrace{2,3,0}_{zs})}_{xs}, \underbrace{(4,5,6,0)}_{ys}\big) = \underbrace{1}_{x}, \underbrace{\underbrace{2,3,4,5,6,0}_{\mathrm{Conc}(zs,ys)}}_{\mathrm{Conc}(xs,ys)}$

- Výsledná funkcia

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

- Párovú diskrimináciu môžeme dosadiť do argumentov a premenovať premennú *zs* na *xs*:

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

- Výpočet:

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

VII.9 Zabudované zreťazenie

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

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

- Vyššia priorita ako párovanie:

$$xs \oplus a, ys \equiv (xs \oplus a), ys \qquad (\dagger\dagger\dagger)$$
$$xs \oplus (a, ys) \qquad (\mathrm{OK})$$

Chyba v (†††): zreťazujeme zoznam xs s prvkom a

Zreťazovať môžeme *iba zoznamy*!

11.2.2. Porovnanie s Pascalom

VII.10 Zoznamy v CL a v Pascale

- Ako by sme naprogramovali zreťazenie

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

v Pascale?

- Napríklad takto:

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

11.2.3. Obrátenie

VII.11 Programovanie so zoznamami: Obrátenie

Naprogramujme funkciu Rev(xs), ktorá vytvorí nový zoznam s obráteným poradím prvkov ako v xs

Riešenie hľadáme rovnakým postupom ako pri zreťazení:

1. Príklad:
   - $\text{Rev}(3, 1, 0, 2, 4, 0) = 4, 2, 0, 1, 3, 0$
2. Uvedomíme si, že vstup je zoznam:
   - je buď *prázdny*: $xs = 0$
   - alebo má *prvý prvok a zvyšok*: $xs = x, zs$

   $\Rightarrow$ párová diskriminácia
3. Určíme výsledok pre prázdny zoznam
   - $\text{Rev}(\underset{xs}{0}) = 0$
4. *Výsledok pre zvyšok zs* zoznamu xs

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

upravíme na výsledok pre *celý zoznam xs*

- $\text{Rev}(\underbrace{3}_{x},\underbrace{(1,0,2,4,0)}_{zs}) = \underbrace{(4,2,0,1,0)}_{\text{Rev}(zs)}\,\mathbf{??}\,\underbrace{3}_{x}$

11.3. Chvostová rekurzia na zoznamoch

11.3.1. Súčet prvkov zoznamu

VII.12 Cykly na zoznamoch

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

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

- V deklaratívnom programovaní úlohu cyklov plní chvostová rekurzia

VII.13 Chvostová rekurzia na zoznamoch

- Chvostová rekurzia simulujúca `while` cyklus pre súčet prvkov zoznamu:

$$\text{While\_suml}(xs, s) = s \qquad \leftarrow xs = 0$$
$$\text{While\_suml}(xs, s) = \text{While\_suml}(zs, s + x) \leftarrow xs = x, zs$$

- Inicializácia:

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

- S dosadením do argumentov:

  $\mathsf{While\_suml}(\quad 0, s) = s$
  $\mathsf{While\_suml}((x, xs), s) = \mathsf{While\_suml}(xs, s + x)$

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

11.3.2. Obrátenie zoznamu

VII.14 Neefektivita obrátenia zoznamu zreťazovaním

- Pred chvíľou sme obrátenie zoznamu naprogramovali:

$$\begin{aligned}\mathsf{Rev}(0) &= 0\\ \mathsf{Rev}(x, xs) &= \mathsf{Rev}(xs) \oplus (x, 0)\end{aligned}$$

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

- Výpočet Rev:

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

- Vo všeobecnosti $(n^2 + n)/2$ krokov, kde $n = \mathsf{L}(xs) \dot{-} 1$

- 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?

- Efektívne obrátenie zoznamu `while` cyklom:

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

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

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

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

11.4. Tupling zoznamových operácií

11.4.1. Split = Take,Drop

VII.17 Príklad – L, Take, Drop

1. Naprogramujme funkciu L(xs), ktorej hodnotou je dĺžka (počet prvkov) zoznamu xs

2. Naprogramujme funkciu Take(i, xs), ktorej hodnotou je zoznam prvých i prvkov zoznamu xs

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

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

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

VII.18 Tupling zoznamových operácií – Split

- Operácie

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

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

  majú rovnakú štruktúru diskriminácií a rekurzie

- Ak potrebujeme výsledky oboch $\rightarrow$ dva prechody zoznamom

- Chceme spojenú jednoprechodovú funkciu – *tupling*$^{\uparrow 9.2.3}$

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

- Príklad s náčrtom rekurzie:

$$\begin{aligned}\mathrm{Split}(2, \quad (3,5,7,11,13,0)) &= \quad (3,5,0),(7,11,13,0)\\ \mathrm{Split}(3,(2,3,5,7,11,13,0)) &= (2,3,5,0),(7,11,13,0)\end{aligned}$$

VII.19 Tupling zoznamových operácií – Split

$$\begin{aligned}\mathrm{Split}(\quad 0, \quad xs\,) &= \quad 0, xs\\ \mathrm{Split}(i+1, \quad 0\,) &= \quad 0, \ 0\\ \mathrm{Split}(i+1, \ (x,xs)) &= (x,ts), ds \leftarrow \mathrm{Split}(i,xs) = ts, ds\end{aligned}$$

11.4.2. Zip, Unzip

VII.20 Tupling zoznamových operácií – Unzip

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

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

- Unzip – rozdelenie zoznamu dvojíc na dva zoznamy

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

- V podstate tuplingová úloha: spájame funkcie

$$\begin{aligned}\mathrm{Firsts}((0,7),(1,6),(2,5),0) &= 0,1,2,0\\ \mathrm{Seconds}((0,7),(1,6),(2,5),0) &= 7,6,5,0\end{aligned}$$

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

- Príklad s náčrtom rekurzie:

$$\begin{aligned}\mathrm{Unzip}(\quad (1,6),(2,5),0) &= \quad (1,2,0), \quad (6,5,0)\\ \mathrm{Unzip}((0,7),(1,6),(2,5),0) &= (0,1,2,0),(7,6,5,0)\end{aligned}$$

$$\begin{aligned}\mathsf{Unzip}(0) &= 0, 0\\ \mathsf{Unzip}((x,y),xys) &= (x,xs),(y,ys) \leftarrow \mathsf{Unzip}(xys) = xs, ys\end{aligned}$$

VIII. prednáška
Triedenie zoznamov
Zoznamová reprezentácia množín

2. apríla 2012

11.5. Prvok zoznamu

VIII.1 Prvok zoznamu

- Kedy je číslo *a prvkom* zoznamu *xs* ($a \varepsilon xs$)?
- Môžeme explicitne vyjadriť párovaním a zreťazením:

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

  alebo indexovaním

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

- Nájdime definíciu zoznamovou rekurziou:

~~$a \notin 0$~~
$a \varepsilon (x, xs) \leftarrow a = x$
$a \varepsilon (x, xs) \leftarrow a \neq x \wedge a \varepsilon xs$
~~$a \notin (x, xs) \leftarrow a \neq x \wedge a \notin xs$~~

- Predikát $a \varepsilon xs$ je zabudovaný v CL (syntax: `a in xs`; syntax negácie $a \notin xs$: `a !in xs`)

12. Triedenie zoznamov

12.1. Špecifikácia triedenia

VIII.2 Špecifikácia triedenia

Funkcia Sort(xs) = ys *utriedi* zoznam xs, ak súčasne platí:

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

Formalizácia špecifikácie:

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

VIII.3 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 Count$(a, xs) \equiv \#_a(xs)$
- Potom máme

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

VIII.4 Permutácia zoznamu – rekurzívny predikát

Ako vypočítame $xs \sim ys$ rekurziou?

1. Rekurzívny výpočet $\#_a(xs)$

   $\#_a(0) = 0$
   $\#_a(x, xs) = \#_a(xs) + 1 \leftarrow a = x$
   $\#_a(x, xs) = \#_a(xs) \quad \leftarrow a \neq x$

2. Pomocný predikát Eq_counts:

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

   Naprogramujeme zoznamovou rekurziou na zs

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

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

VIII.5 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:

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

- Vyjadrenie zreťazením:

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

- Stačí porovnávať susediace prvky:

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

VIII.6 Usporiadaný zoznam – rekurzívny predikát

Ako vypočítame Ord(xs) rekurziou bez indexovania?

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

$\mathrm{Ord}(0)$
$\mathrm{Ord}(a, 0)$
$\mathrm{Ord}(a, b, xs) \leftarrow a \leq b \wedge \mathrm{Ord}(b, xs)$

VIII.7 Špecifikácia triedenia – rekapitulácia

Funkcia Sort(xs) utriedi zoznam xs, ak spĺňa

$$\mathrm{Sort}(xs) \sim xs$$
$$\mathrm{Ord}(\mathrm{Sort}(xs)),$$

pričom

$$xs \sim ys \leftrightarrow \forall a(\#_a(xs) = \#_a(ys))$$
$$\mathrm{Ord}(us) \leftrightarrow$$
$$\forall a \forall b \forall xs \forall ys \forall zs(us = xs \oplus (a, ys) \oplus (b, zs) \rightarrow a \leq b)$$

VIII.8 Algoritmy triedenia zoznamov

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

12.2. Triedenie vsúvaním

VIII.9 Triedenie vsúvaním (insertion sort)

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

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

  - Predpokladáme, že Isort utriedi xs

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

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

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

    Teda napríklad

    $$\begin{aligned} \mathsf{Isort}(\underbrace{7,2,0,5,0}_{xs}) &= 0,2,5,7,0 \\ &\quad\downarrow ? \\ \mathsf{Isort}(\underbrace{3}_{x}, \underbrace{7,2,0,5,0}_{xs}) &= \underbrace{0,2,3,5,7,0}_{?(x,\mathsf{Isort}(xs))} \end{aligned}$$

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

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

VIII.10 Triedenie vsúvaním: vsunutie

- Funkcia Ins vsunie prvok do utriedeného zoznamu

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

- Opäť zoznamovou rekurziou:
  - Vsunutím prvku a do prázdneho zoznamu vzniká jednoprvkový zoznam obsahujúci iba a:

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

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

    Ako vložíme a do utriedeného zoznamu x, xs?

    Závisí od vzájomného vzťahu a a x:

$$\mathsf{Ins}(\underset{a}{3}, (\underset{x}{5}, \underbrace{7, 9, 9, 0}_{xs})) = \underset{a}{3}, \underset{x}{5}, \underbrace{7, 9, 9, 0}_{xs}$$
$$\mathsf{Ins}(\underset{a}{3}, (\underset{x}{0}, \underbrace{2, 5, 7, 0}_{xs})) = \underset{x}{0}, \underbrace{2, 3, 5, 7, 0}_{\mathsf{Ins}(a,xs)}$$

VIII.11 Triedenie vsúvaním: výpočet

$$\begin{aligned}
&\mathsf{Isort}(3, 7, 2, 0, 5, 0)\\
&\quad = \mathsf{Ins}(3, \mathsf{Isort}(7, 2, 0, 5, 0))\\
&\quad = \mathsf{Ins}(3, \mathsf{Ins}(7, \mathsf{Isort}(2, 0, 5, 0)))\\
&\quad \ldots\\
&\quad = \mathsf{Ins}(3, \mathsf{Ins}(7, \mathsf{Ins}(2, \mathsf{Ins}(0, \mathsf{Ins}(5, \mathsf{Isort}(0))))))\\
&\quad = \mathsf{Ins}(3, \mathsf{Ins}(7, \mathsf{Ins}(2, \mathsf{Ins}(0, \mathsf{Ins}(5, 0)))))\\
&\quad = \mathsf{Ins}(3, \mathsf{Ins}(7, \mathsf{Ins}(2, \mathsf{Ins}(0, (5, 0)))))\\
&\quad = \mathsf{Ins}(3, \mathsf{Ins}(7, \mathsf{Ins}(2, (0, 5, 0))))\\
&\quad = \mathsf{Ins}(3, \mathsf{Ins}(7, (0, 2, 5, 0)))\\
&\quad = \mathsf{Ins}(3, (0, 2, 5, 7, 0))\\
&\quad = 0, 2, 3, 5, 7, 0
\end{aligned}$$

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

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

12.3. Zlúčenie utriedených zoznamov

VIII.12 Zlúčenie utriedených zoznamov

- Ako zlúčime dva utriedené zoznamy do utriedeného zoznamu?

$$\mathsf{Merge}(xs, ys) \sim xs \oplus ys$$
$$\mathsf{Ord}(xs) \wedge \mathsf{Ord}(ys) \to \mathsf{Ord}\,\mathsf{Merge}(xs, ys)$$

  Napríklad

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

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

$$\begin{aligned}\mathsf{Merge}(\quad 0, ys) &= ys\\ \mathsf{Merge}((x, xs), ys) &= \mathsf{Ins}(x, \mathsf{Merge}(xs, ys))\end{aligned}$$

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

▶ Vôbec sme nevyužili, že xs je utriedený

VIII.13 Zlúčenie utriedených zoznamov efektívne

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

xs	ys	$\mathsf{Merge}(xs, ys)$
2, 3, 5, 0	**0**, 1, 7, 0	0,
2, 3, 5, 0	**1**, 7, 0	1,
2, 3, 5, 0	**7**, 0	2,
3, 5, 0	**7**, 0	3,
5, 0	**7**, 0	5,
0	7, 0	7, 0

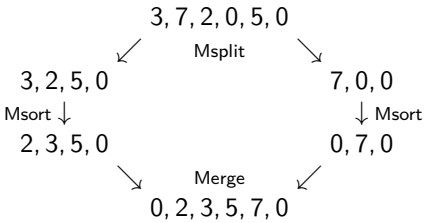
- Dĺžka výpočtu Merge(xs, ys) je teraz $\mathsf{L}(xs) + \mathsf{L}(ys)$
- Kostra definície:

$$\begin{aligned}
&\mathsf{Merge}(xs, ys) = ? \leftarrow xs = 0\\
&\mathsf{Merge}(xs, ys) = ? \leftarrow xs = x, us \wedge ys = 0\\
&\mathsf{Merge}(xs, ys) = ? \leftarrow xs = x, us \wedge ys = y, vs \wedge x \leq y\\
&\mathsf{Merge}(xs, ys) = ? \leftarrow xs = x, us \wedge ys = y, vs \wedge x > y
\end{aligned}$$

12.4. Triedenie zlučovaním

VIII.14 Triedenie zlučovaním (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!):

$$\begin{aligned}
&\mathsf{Msplit}(xs) = ys, zs \rightarrow\\
&\quad \mathsf{Msort}(xs) = \mathsf{Merge}(\mathsf{Msort}(ys), \mathsf{Msort}(zs))
\end{aligned}$$

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

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

- Dá sa urobiť rôzne, napríklad

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

  Čiže

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

- Kostra definície:

$$\begin{aligned}\mathsf{Msplit}(0) &= ?, ?\\ \mathsf{Msplit}(a, 0) &= ?, ?\\ \mathsf{Msplit}(a, b, xs) &= ?, ? \leftarrow \mathsf{Msplit}(xs) = ys, zs\end{aligned}$$

13. Zoznamová reprezentácia konečných množín

- Konečná množina je kontainer – 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

VIII.17 Množiny ako usporiadané zoznamy

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

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

  Rekurzívna definícia – podobne ako Ord
- Na definovanie operácií budeme používať *trichotomickú* diskrimináciu

$$\cdots \leftarrow s < t$$
$$\cdots \leftarrow s = t$$
$$\cdots \leftarrow s > t$$

VIII.18 Príslušnosť a extenzionalita

- Príslušnosť do množiny

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

- Pre neprázdne množiny:

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

- Využijeme pri rekurzívnej definícii predikátu $\in$

~~$a \notin 0$~~
~~$a \notin x, xs \leftarrow a < x$~~
$a \in x, xs \leftarrow a = x$
$a \in x, xs \leftarrow a > x \wedge a \in xs$
~~$a \notin x, xs \leftarrow a > x \wedge a \notin xs$~~

- Platí extenzionalita

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

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

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

- Princíp implementácie je rovnaký ako pri Merge
- Ďalšie operácie ($\subseteq$, $\cap$, $\setminus$, $\triangle$) sa implementujú podobne

X. prednáška
Kombinatorické predikáty a funkcie na zoznamoch

16. apríla 2012

14. Kombinatorické predikáty a funkcie na zoznamoch

X.1 Predikáty (opakovanie)

- *Predikát* je vlastnosť objektu alebo n-tice objektov
- Príklady predikátov na zoznamoch:
  - xs je prázdny zoznam

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

  - zoznam xs je palindróm

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

  - a je prvkom zoznamu xs

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

  - ys je prefixom (začiatočným úsekom) zoznamu xs

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

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

14.1. Súvislé úseky

14.1.1. Sufixy

X.2 Sufix

- Sufix – koncový úsek zoznamu

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

- Napríklad

$$(7, 11, 0) \sqsupset (7, 11, 0) \quad (1)$$
$$(2, 3, 5, 7, 11, 0) \sqsupset (7, 11, 0) \quad (2)$$
$$\neg(0 \sqsupset (7, 11, 0))$$
$$\neg((5, 7, 11, 13, 0) \sqsupset (7, 11, 0))$$

- Všimnite si:

$$\begin{array}{rcl} zs & = & \mid 7, 11, 0 \\ xs & = & \underbrace{2, 3, 5,}_{\text{čokoľvek}} \mid \underbrace{7, 11, 0}_{zs} \end{array}$$

- Ako zadefinujeme rekurzívne?

X.3 Sufix – analýza prípadov

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

$$0 \sqsupset zs \leftrightarrow zs = 0$$

  - Sufixami neprázdneho zoznamu (x, xs) sú
    - zoznam (x, xs) sám a
    - všetky sufixy zoznamu xs,

teda

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

- Klauzálna definícia s analýzou prípadov namiesto disjunkcie:

$$\begin{aligned} 0 \sqsupset zs &\leftarrow zs = 0 \\ (x, xs) \sqsupset zs &\leftarrow zs = (x, xs) \\ (x, xs) \sqsupset zs &\leftarrow zs \neq (x, xs) \wedge xs \sqsupset zs \end{aligned}$$

X.4 Všetky sufixy

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

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

- Špecifikácia:

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

- Všimnite si:

$$a \; \varepsilon \; (x, xs) \leftrightarrow a = x \vee a \; \varepsilon \; xs$$

- Využime analýzu prípadov pre $xs \sqsupset zs$

X.5 Všetky sufixy – odvodenie riešenia

$$\begin{aligned} zs \; \varepsilon \; \text{Suffixes}(xs) &\leftrightarrow xs \sqsupset zs \\ a \; \varepsilon \; (x, xs) &\leftrightarrow a = x \vee a \; \varepsilon \; xs \end{aligned}$$

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

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

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

  teda

$$\begin{aligned} zs \;\varepsilon\; \text{Suffixes}(x, xs) &\leftrightarrow (x, xs) \sqsupset zs \\ &\leftrightarrow zs = (x, xs) \vee xs \sqsupset zs \\ &\leftrightarrow zs = (x, xs) \vee zs \;\varepsilon\; \text{Suffixes}(xs) \\ &\leftrightarrow zs \;\varepsilon\; ((x, xs), \text{Suffixes}(xs)) \end{aligned}$$

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

14.1.2. Prefixy

X.6 Prefix

- Prefix – začiatočný úsek zoznamu

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

- Napríklad

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

- Všimnite si:

$$\begin{aligned} ys &= 2, 3, 5, \mid 0 \\ xs &= \underbrace{2, 3, 5,}_{\text{prvky } ys} \mid \underbrace{7, 11, 0}_{\text{čokoľvek}} \end{aligned}$$

- Ako zadefinujeme rekurzívne?

- $ys \sqsubset xs$ analýzou prípadov pre ys:
  - Prázdny zoznam je prefixom každého zoznamu xs

$$0 \sqsubset xs$$

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

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

- $ys \sqsubset xs$ analýzou prípadov pre xs:
  - Prefixom prázdneho zoznamu je iba prázdny zoznam:

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

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

    teda

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

- Druhá verzia je užitočnejšia

X.8 Všetky prefixy

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

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

- Špecifikácia:

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

- Využime analýzu prípadov pre $ys \sqsubset xs$

X.9 Všetky prefixy – odvodenie riešenia

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

- Prefixom prázdneho zoznamu je iba prázdny zoznam:

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

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

  teda

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

- Potrebujeme pomocnú funkciu ???

X.10 Všetky prefixy – pomocná funkcia

- Špecifikácia pomocnej funkcie pre Prefixes:

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

- Pridá x na začiatok každého zoznamu z vss

14.1.3. Segmenty

X.11 Segment

- Segment – všeobecný súvislý úsek zoznamu

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

- Napríklad

$$(5,7,0) \subset (5,7,11,13,0)$$
$$(5,7,0) \subset (2,3,5,7,11,13,0)$$
$$\neg((5,11,0) \subset (2,3,5,7,11,13,0)$$

- Všimnite si:

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

- Ako zadefinujeme rekurzívne?
- Ktorý z predchádzajúcich predikátov nám pomôže?

14.2. Podpostupnosti

X.12 Podpostupnosť

- Zoznam ys je *vybranou podpostupnosťou* zoznamu $xs = x_1, x_2, \ldots, x_n, 0$, ak

$$ys = x_{i_1}, x_{i_2}, \ldots, x_{i_k}, 0$$

pre nejakú rastúcu postupnosť $0 \leq i_1 < i_2 < \cdots < i_k \leq n$

- Formálnejšie na zoznamoch:

$$\begin{array}{l} \mathsf{Subseq}(ys, xs) \equiv (ys \lhd xs) \leftrightarrow \\ \quad \exists is(\mathsf{Set}(is) \wedge \mathsf{L}(is) = \mathsf{L}(ys) \wedge \\ \quad\quad \forall i(i < \mathsf{L}(is) \rightarrow is[i] < \mathsf{L}(xs) \wedge ys[i] = xs[is[i]])) \end{array}$$

- Neformálne:
  - ys vznikne vynechaním niektorých prvkov zo zoznamu xs
  - vzájomné poradie zostávajúcich prvkov sa nezmení
- Napríklad:

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

X.13 Podpostupnosť – analýza prípadov

Podpostupnosť môžeme zadefinovať rekurzívne využitím:

- Podpostupnosťou prázdneho zoznamu je iba prázdny zoznam:

$$ys \lhd 0 \leftrightarrow ys = 0$$

- 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

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

X.14 Všetky podpostupnosti

- Generovanie všetkých podpostupností zoznamu – funkcia Subseqs(xs)

$$ys \; \varepsilon \; \text{Subseqs}(xs) \leftrightarrow ys \lhd xs$$

- Odvodíme opäť využitím analýzy prípadov pre $ys \lhd xs$
- Potrebujeme tiež:

$$a \; \varepsilon \; xs \oplus ys \leftrightarrow a \; \varepsilon \; xs \vee a \; \varepsilon \; ys$$

1. Jedinou podpostupnosťou prázdneho zoznamu je prázdny zoznam

$$ys \varepsilon \text{Subseqs}(xs) \leftrightarrow ys \lhd 0 \leftrightarrow ys = 0$$

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

$$ys \varepsilon \text{Subseqs}(x, xs) \leftrightarrow ys \lhd (x, xs)$$
$$\leftrightarrow \exists zs(ys = (x, zs) \wedge zs \lhd xs) \vee ys \lhd xs$$
$$\leftrightarrow \exists zs(ys = (x, zs) \wedge zs \varepsilon \text{Subseqs}(xs)) \vee ys \varepsilon \text{Subseqs}(xs)$$

Príklad:

$$\begin{aligned} \text{Subseqs}(\text{'ak'}) &= \text{'ak'}, \text{'a'}, \text{'k'}, \text{''}, 0 \\ \text{Subseqs}(\text{'tak'}) &= (\text{'tak'}, \text{'ta'}, \text{'tk'}, \text{'t'}, 0) && \text{a)} \\ &\oplus (\text{'ak'}, \text{'a'}, \text{'k'}, \text{''}, 0) && \text{b)} \end{aligned}$$

14.3. Permutácie

X.16 Permutácia

- Permutáciu zoznamu môžeme zadefinovať aj inak ako

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

- Potrebujeme pomocný pojem – vsunutie:

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

- Analýza prípadov pre permutácie:
  - Permutáciou prázdneho zoznamu je iba prázdny zoznam:

$$0 \sim ys \leftrightarrow ys = 0$$

  - Permutáciou zoznamu (x, xs) je ľubovoľné vsunutie prvku x do nejakej permutácie zoznamu xs:

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

X.17 Vsunutie

- Analýza prípadov pre vsunutie:
  - ys je vsunutím a do prázdneho zoznamu, ak ys obsahuje iba a:

$$ys \approx 0[\downarrow a] \leftrightarrow ys = (a, 0)$$

  - ys je vsunutím a do zoznamu (x, xs), ak ys vznikne pridaním a na začiatok (x, xs) alebo ak ys začína prvkom x a pokračuje nejakým vsunutím zs prvku a do zoznamu xs:

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

14.4. Ďalšie úlohy

X.18 Ďalšie kombinatorické úlohy

Premyslite si:

- k-prvkové podpostupnosti,
- variácie (ako permutácie, ale niektoré prvky možno vynechať)
- k-prvkové variácie,
- segmenty[↑14.1.3]

XI. prednáška
Binárne stromy

23. apríla 2012

15. Binárne stromy

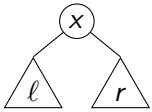
15.1. Kódovanie stromov párovacími konštruktormi

XI.1 Binárne stromy

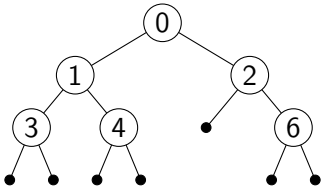
Binárny strom – dátová štruktúra definovaná rekurzívne:

- Prázdny strom

- Vrchol s hodnotou x a dvoma deťmi ℓ a r, ktoré sú binárnymi stromami



Napríklad



XI.2 Párovacie konštruktory

- Binárne stromy by sme v CL mohli zakódovať párovaním:
  - Prázdny strom: 0

– Uzol s hodnotou x a deťmi ℓ a r: trojica (x, ℓ, r)

- Použijeme elegantnejší spôsob – *párovacie konštruktory*

$$C(x_1, x_2, \ldots, x_n) = \underline{k}, x_1, x_2, \ldots, x_n$$

$\underline{k}$ je číselná konštanta – *tag*

– Z hodnoty konštruktora sa dajú jednoznačne dekódovať jeho argumenty

$$C(x_1, \ldots, x_n) = C(y_1, \ldots, y_n) \to x_1 = y_1 \wedge \cdots \wedge x_n = y_n$$

– Žiadna hodnota nemôže byť súčasne výsledkom dvoch konštruktorov s rôznymi tagmi

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

$\Rightarrow$ Konštruktory s rôznymi tagmi sú *diskriminovateľné*

XI.3 Kódovanie binárnych stromov

Binárny strom *zakódujeme* konštruktormi takto:

- Prázdny strom:

$$\mathsf{E} \equiv \bullet = 0, 0$$

- Uzol s hodnotou x a deťmi ℓ a r:

$$\mathsf{Nd}(x, \ell, r) \equiv \frac{x}{\ell \mid r} = 1, x, \ell, r$$

Binárny strom *dekódujeme* diskrimináciou

$$\cdots \leftarrow t = \mathsf{E} \qquad \cdots \leftarrow t = \bullet$$

$$\cdots \leftarrow t = \mathsf{Nd}(x, \ell, r) \qquad \cdots \leftarrow t = \frac{x}{\ell \mid r}$$

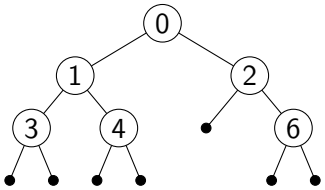
Po dosadení do argumentov:

$$f(\mathsf{E}) = \cdots \qquad f(\bullet) = \cdots$$

$$f(\mathsf{Nd}(x, \ell, r)) = \cdots \qquad f\left(\frac{x}{\ell \mid r}\right) = \cdots$$

XI.4 Kódovanie binárnych stromov – príklad

Napríklad strom



zakódujeme

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

XI.5 Formát binárnych stromov

- Nie každé číslo kóduje binárny strom
- Číslo, ktoré kóduje binárny strom, sa dá zapísať pomocou konštruktorov E, Nd
- Predikát $\mathsf{Bt}(t)$ platí iba pre kódy binárnych stromov:

  $\mathsf{Bt}(\mathsf{E})$
  $\mathsf{Bt}(\mathsf{Nd}(x,\ell,r)) \leftarrow \mathsf{N}(x) \wedge \mathsf{Bt}(\ell) \wedge \mathsf{Bt}(r)$
  $\equiv$
  $\mathsf{Bt}(\bullet)$
  $\mathsf{Bt}\left(\frac{x}{\ell \mid r}\right) \leftarrow \mathsf{N}(x) \wedge \mathsf{Bt}(\ell) \wedge \mathsf{Bt}(r)$

- V CL takéto predikáty slúžia aj na formátovanie výsledkov query

15.2. Operácie na binárnych stromoch

XI.6 Rekurzia na binárnych stromoch

Operácie na binárnych stromoch definujeme *rekurziou na binárnych stromoch*

$$f(\mathsf{E}) = \cdots$$
$$f(\mathsf{Nd}(x, \ell, r)) = \cdots f(\ell) \cdots f(r) \cdots$$

Špeciálny prípad course-of-values rekurzie, kódy detí sú menšie čísla ako kód ich rodičovského uzla

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

XI.7 Dovolené defaulty

- Všimnite si: Klauzálna definícia funkcie $f(t)$

  $$f(\mathsf{E}) = \cdots$$
  $$f(\mathsf{Nd}(x, \ell, r)) = \cdots f(\ell) \cdots f(r) \cdots$$

  neurčuje hodnotu $f(t)$ pre všetky čísla t

- $f(t)$ definujeme za predpokladu $\mathsf{Bt}(t)$, ostatné prípady prenechávame na default

- Všetky funkcie v tejto prednáške definujeme za predpokladu $\mathsf{Bt}(t)$

- **Pravidlá pre použitie defaultov:**
  - Ak platia predpoklady špecifikácie, napr. $\mathsf{Bt}(t)$, *musíme* hodnotu $f(t)$ uviesť explicitne, *aj keď je ňou* 0!
  - Ak predpoklady špecifikácie neplatia, napr. $\neg\mathsf{Bt}(t)$, môžeme hodnotu $f(t)$ prenechať na default

XI.8 Operácie na binárnych stromoch

Veľkosť (počet uzlov) binárneho stromu $\mathrm{Sz}(t) \equiv |t|_b$:

$$\begin{aligned} \mathrm{Sz}(\mathrm{E}) = 0 &\equiv |\bullet|_b = 0 \\ \mathrm{Sz}\,\mathrm{Nd}(x, \ell, r) = 1 + \mathrm{Sz}(\ell) + \mathrm{Sz}(r) &\equiv \left|\tfrac{x}{\ell \mid r}\right|_b = 1 + |\ell|_b + |r|_b \end{aligned}$$

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

$$a\ \varepsilon_b\ \tfrac{x}{\ell \mid r} \leftarrow a = x$$
$$a\ \varepsilon_b\ \tfrac{x}{\ell \mid r} \leftarrow a \neq x \wedge a\ \varepsilon_b\ \ell$$
$$a\ \varepsilon_b\ \tfrac{x}{\ell \mid r} \leftarrow a \neq x \wedge \neg(a\ \varepsilon_b\ \ell) \wedge a\ \varepsilon_b\ r$$

15.3. Prechody binárnymi stromami

XI.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 – $\mathrm{Inorder}(t)$ (za predpokladu $\mathrm{Bt}(t)$):

$$\mathrm{Inorder}\left(\tfrac{0}{\tfrac{1}{\tfrac{3}{\bullet \mid \bullet} \mid \tfrac{4}{\bullet \mid \bullet}} \mid \tfrac{2}{\bullet \mid \tfrac{6}{\bullet \mid \bullet}}}\right) = 3, 1, 4, 0, 2, 6, 0$$

$$\nearrow\ ?\ \nwarrow$$

$$\mathrm{Inorder}\left(\tfrac{1}{\tfrac{3}{\bullet \mid \bullet} \mid \tfrac{4}{\bullet \mid \bullet}}\right) = 3, 1, 4, 0 \qquad \mathrm{Inorder}\left(\tfrac{2}{\bullet \mid \tfrac{6}{\bullet \mid \bullet}}\right) = 2, 6, 0$$

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

XI.10 Prechod binárnym stromom – výpočet

$$
\begin{aligned}
&\mathsf{Inorder}\left(\begin{array}{c}0\\ \hline \begin{array}{c|c}\begin{array}{c}1\\ \hline \begin{array}{c|c}3 & 4\end{array}\end{array} & \begin{array}{c}2\\ \hline \begin{array}{c|c}\bullet & 6\end{array}\end{array}\end{array}\end{array}\right)\\
&= \mathsf{Inorder}\left(\begin{array}{c}1\\ \hline \begin{array}{c|c}3 & 4\end{array}\end{array}\right) \oplus \left(0, \mathsf{Inorder}\left(\begin{array}{c}2\\ \hline \begin{array}{c|c}\bullet & 6\end{array}\end{array}\right)\right)\\
&= \left(\mathsf{Inorder}\left(\begin{array}{c}3\\ \hline \bullet \mid \bullet\end{array}\right) \oplus \left(1, \mathsf{Inorder}\left(\begin{array}{c}4\\ \hline \bullet \mid \bullet\end{array}\right)\right)\right)\\
&\quad \oplus \left(0, \mathsf{Inorder}(\bullet) \oplus \left(2, \mathsf{Inorder}\left(\begin{array}{c}6\\ \hline \bullet \mid \bullet\end{array}\right)\right)\right)\\
&= \cdots\\
&= ((0 \oplus (3,0)) \oplus (1, 0 \oplus (4,0))) \oplus (0, 0 \oplus (2, 0 \oplus (6,0)))\\
&= ((3,0) \oplus (1,4,0)) \oplus (0,2,6,0)\\
&= (3,1,4,0) \oplus (0,2,6,0)\\
&= 3,1,4,0,2,6,0
\end{aligned}
$$

XI.11 Efektívne prechody binárnymi stromami

- Výpočet Inorder je neefektívny – zreťazenie prechádza znova zoznam $\mathsf{Inorder}(\ell)$
- Zefektívnenie – odstránenie zreťazenia
- Trik: Pomocná premenná (akumulátor) *as* – zoznam, ktorý pripojíme *za* výpis (pod)stromu inorderom
- Prvky pridávame na začiatok akumulátora
- Požadovaná vlastnosť efektívnej (akumulátorovej) funkcie:

$$\mathsf{Inordera}(t, as) = \mathsf{Inorder}(t) \oplus as \qquad (*)$$

- Funkciu Inordera naprogramujeme bez zreťazenia

  Program odvodíme z požadovanej vlastnosti $(*)$

$$\mathsf{Inordera}(t, as) = \mathsf{Inorder}(t) \oplus as \qquad (*)$$

$$\mathsf{Inordera}\left(\frac{0}{\frac{1}{\frac{3}{\bullet\mid\bullet}\mid\frac{4}{\bullet\mid\bullet}}\mid\frac{2}{\bullet\mid\frac{6}{\bullet\mid\bullet}}}, \underline{98, 99, 0}_{as}\right) = 3, 1, 4, 0, 2, 6, \underline{98, 99, 0}_{as}$$

$$\uparrow ?$$

$$\mathsf{Inordera}\left(\frac{1}{\frac{3}{\bullet\mid\bullet}\mid\frac{4}{\bullet\mid\bullet}}, \underline{77, 78, 0}_{as_1}\right) = 3, 1, 4, \underline{77, 78, 0}_{as_1}$$

$$\mathsf{Inordera}\left(\frac{2}{\bullet\mid\frac{6}{\bullet\mid\bullet}}, \underline{88, 89, 0}_{as_2}\right) = 2, 6, \underline{88, 89, 0}_{as_2}$$

$$\mathsf{Inordera}(t, as) = \mathsf{Inorder}(t) \oplus as \qquad (*)$$

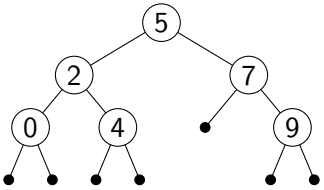
$$\begin{aligned}
&\mathsf{Inordera}(\bullet, as) \overset{(*)}{=} \mathsf{Inorder}(\bullet) \oplus as = 0 \oplus as = as \\
&\mathsf{Inordera}\left(\frac{x}{\ell \mid r}, as\right) \\
&\quad \overset{(*)}{=} \mathsf{Inorder}\left(\frac{x}{\ell \mid r}\right) \oplus as \\
&\quad = \big(\mathsf{Inorder}(\ell)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

15.4. Binárne vyhľadávacie stromy

XI.14 Binárne vyhľadávacie stromy

Binárny vyhľadávací strom

- Binárny strom
- V každom uzle s hodnotou x platí súčasne:
  - x je väčšie ako všetky hodnoty v ľavom dieťati
  - x je menšie ako všetky hodnoty v pravom dieťati
  - obe deti sú binárne vyhľadávacie stromy



XI.15 Binárny vyhľadávací strom – príklad

Vyjadrenie rekurzívnym predikátom:

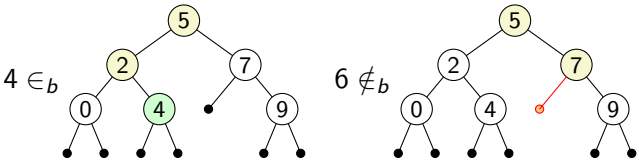
$\mathrm{Bst}(\bullet)$
$\mathrm{Bst}\left(\frac{x}{\ell \mid r}\right) \leftarrow x \succ \ell \wedge x \prec r \wedge \mathrm{Bst}(\ell) \wedge \mathrm{Bst}(r)$

Cvičenie: rekurzívne definície predikátov

$$x \succ t \leftrightarrow \forall y(y \; \varepsilon_b \; t \rightarrow x > y)$$
$$x \prec t \leftrightarrow \forall y(y \; \varepsilon_b \; t \rightarrow x < y)$$

- 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 $t = \frac{x}{\ell \mid r}$:
  - Ak $a < x$, prehľadávame iba ℓ, lebo a nemôže byť v r
  - Ak $a > x$, prehľadávame iba r, lebo a nemôže byť v ℓ

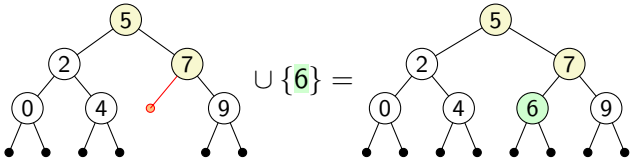
  $\Rightarrow$ Prehľadávame *iba jednu cestu* stromom



Princíp vkladania hodnoty do vyhľadávacieho stromu:

- Pokúsime sa nájsť vkladanú hodnotu
  - a) Hodnota sa v strome nenachádza (na jej mieste je prázdny strom)

    $\Rightarrow$ vyrobíme nový uzol

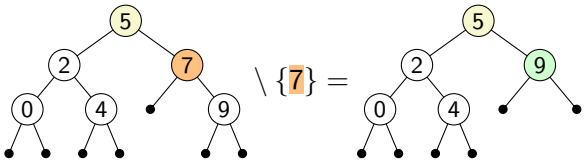


  - b) Hodnota sa v strome nachádza $\Rightarrow$ strom sa nemení
- Cestou naspäť strom zrekonštruujeme

XI.18 Zmazanie z vyhľadávacieho stromu I

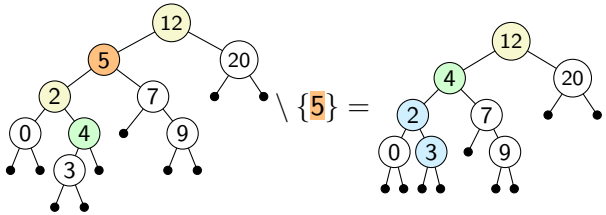
Zmazanie hodnoty je komplikovanejšie ako vloženie

- Pokúsime sa nájsť zmazávanú hodnotu:
  a) Hodnotu nenájdeme ⇒ strom sa nezmení
  b) Hodnotu nájdeme a niektoré dieťa je prázdne ⇒ nahradíme uzol druhým dieťaťom



XI.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:

$$\mathsf{Bst}(t) \wedge t \neq \bullet \to \mathsf{Maxt}(t)\ \varepsilon_b\ t$$
$$\mathsf{Bst}(t) \wedge t \neq \bullet \wedge x\ \varepsilon_b\ t \to x \leq \mathsf{Maxt}(t)$$

$$\mathsf{Bst}(t) \wedge t \neq \bullet \to \mathsf{Bst}\,\mathsf{Delmaxt}(t)$$
$$\mathsf{Bst}(t) \wedge t \neq \bullet \to x\ \varepsilon_b\ \mathsf{Delmaxt}(t) \leftrightarrow x\ \varepsilon_b\ t \wedge x \neq \mathsf{Maxt}(t)$$

- Programy hľadajú maximum v najpravejšom uzle:

$$\mathsf{Maxt}\left(\tfrac{x}{\ell \mid r}\right) = x \qquad \leftarrow r = \bullet$$
$$\mathsf{Maxt}\left(\tfrac{x}{\ell \mid r}\right) = \mathsf{Maxt}(r) \qquad \leftarrow r \neq \bullet$$

$$\mathsf{Delmaxt}\left(\tfrac{x}{\ell \mid r}\right) = \ell \qquad \leftarrow r = \bullet$$
$$\mathsf{Delmaxt}\left(\tfrac{x}{\ell \mid r}\right) = \tfrac{x}{\ell \mid \mathsf{Delmaxt}(r)} \leftarrow r \neq \bullet$$

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

- Spojená funkcia – extrakcia maxima:

$$\mathsf{Bst}(t) \wedge t \neq \bullet \to \mathsf{Extract\_max}(t) = \mathsf{Maxt}(t), \mathsf{Delmaxt}(t)$$

$$\begin{aligned}&\mathsf{Bst}(t) \wedge t \neq \bullet \to \\ &\quad \exists m \exists t_1 \big(\mathsf{Extract\_max}(t) = m, t_1 \wedge m\ \varepsilon_b\ t \wedge \mathsf{Bst}(t_1) \\ &\qquad \wedge \forall x (x\ \varepsilon_b\ t \to x \leq m) \\ &\qquad \wedge \forall x (x\ \varepsilon_b\ t_1 \leftrightarrow x\ \varepsilon_b\ t \wedge x \neq m)\big)\end{aligned}$$

- Definícia:

$$\mathsf{Extract\_max}\left(\tfrac{x}{\ell \mid r}\right) = ?, ? \leftarrow r = \bullet$$
$$\mathsf{Extract\_max}\left(\tfrac{x}{\ell \mid r}\right) = ?, ? \leftarrow r \neq \bullet \wedge \mathsf{Extract\_max}(?) = ?, ?$$

XII. prednáška
Číslovanie binárnych stromov
Všeobecné stromy

30. apríla 2012

15. Binárne stromy

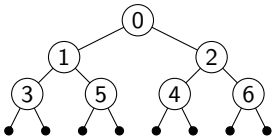
15.5. Číslovanie vrcholov v binárnych stromoch

XII.1 Úplné binárne stromy

- Hĺbka stromu – dĺžka najdlhšej cesty koreň–list:

$$\mathrm{d}(\bullet) = 0$$
$$\mathrm{d}\left(\frac{x}{\ell \mid r}\right) = 1 + \max(\mathrm{d}(\ell), \mathrm{d}(r))$$

- Úplný binárny strom:
  - na každej úrovni okrem poslednej: všetky uzly neprázdne
  - na poslednej úrovni sú iba prázdne stromy

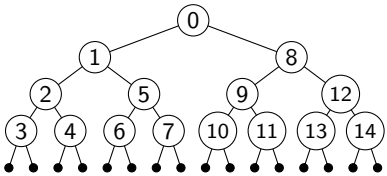


- Vzťah veľkosti a hĺbky úplného stromu:

$$|t|_b + 1 = 2^{\mathrm{d}(t)}$$

- Číslovanie uzlov úplného stromu číslami 0, 1, 2, ... v poradí preorder:

Full_pre(4) =

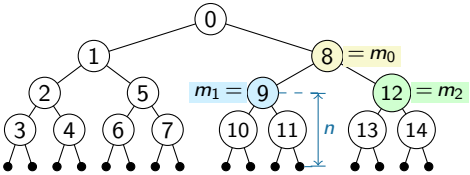


- Na nájdenie rekurzívneho riešenia musíme problém *zovšeobecniť*
- Zovšeobecnenie umožní číslovať podstromy

- Zovšeobecnenie – $\text{Full\_pre}_1(n, m)$:

  Číslovanie uzlov úplného stromu hĺbky n číslami m, $m+1$, $m+2$, ... v poradí preorder

- Schéma riešenia:

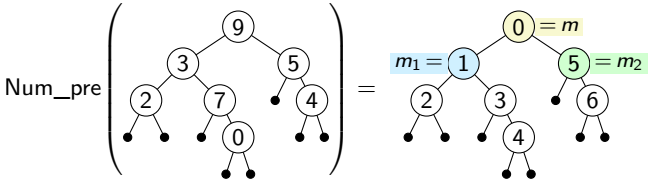


  Číslovanie v poradí *preorder*:

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

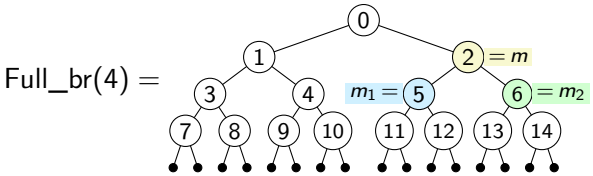
- Aké je s pre úplný strom hĺbky n?
- Ako sa číslovanie zmení pre inorder, postorder?

- Číslovanie uzlov *ľubovoľného* stromu číslami 0, 1, 2, ... v poradí pre-order:



- Rovnaký postup ako pri úplných stromoch:
  - Zovšeobecníme na číslovanie od m – $\text{Num\_pre}_1(t, m)$
  - Aký je rozdiel medzi číslom ľavého a pravého dieťaťa?
  - Rozdiel vieme počítať aj bez pomocnej funkcie – tupling

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



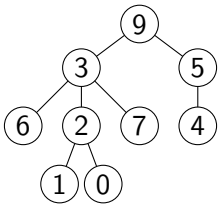
- Problém zovšeobecníme pre podstrom s koreňom s číslom m
- Aký je vzťah medzi číslom uzla m a číslami jeho detí m_1 a m_2?

16. Všeobecné stromy

16.1. Kódovanie

XII.6 Všeobecné stromy – kódovanie

Všeobecný strom – ľubovoľne veľa potomkov každého uzla:



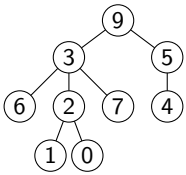
Ako zakódujeme takýto strom v CL?

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

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

  x – hodnota uložená v uzle stromu, ts – zoznam detí
- Ako zakódujeme strom na obrázku?

XII.7 Všeobecné stromy – kódovanie



- Ako zakódujeme strom na obrázku?

$$\begin{aligned}\mathsf{T}_1 = \mathsf{Gnd}(9, (\mathsf{Gnd}(3, (\mathsf{Gnd}(6,0),\\ &\quad \mathsf{Gnd}(2,0),\\ &\quad \mathsf{Gnd}(7, (\mathsf{Gnd}(1,0), \mathsf{Gnd}(0,0),0)),\\ &\quad 0)),\\ &\quad \mathsf{Gnd}(5, (\mathsf{Gnd}(4,0),0),\\ &\quad 0))\\ \equiv \frac{9}{\frac{3}{\frac{6}{0}, \frac{2}{0}, \frac{7}{\frac{1}{0}, \frac{0}{0}, 0}, 0}, \frac{5}{\frac{4}{0}, 0}, 0}\end{aligned}$$

XII.8 Všeobecné stromy – dekódovanie

- Všeobecný strom dekódujeme diskrimináciou

$$\cdots \leftarrow t = \frac{x}{ts} \qquad \equiv \qquad \cdots \leftarrow t = \mathsf{Gnd}(x, ts)$$

- Zoznam všeobecných stromov dekódujeme kombináciou zoznamovej diskriminácie a diskriminácie na všeobecných stromoch:

$$\begin{aligned}\cdots \leftarrow ts = 0 \qquad &\equiv \qquad \cdots \leftarrow ts = 0\\ \cdots \leftarrow ts = \frac{x}{ts_1}, ts_2 \qquad &\equiv \qquad \cdots \leftarrow ts = \mathsf{Gnd}(x, ts_1), ts_2\end{aligned}$$

- Obe diskriminácie môžeme dosadiť do argumentov:

$$\begin{aligned}f\left(\frac{x}{ts}\right) = \cdots \qquad &\equiv \qquad f\ \mathsf{Gnd}(x, ts) = \cdots\\ f(0) = \cdots \qquad &\equiv \qquad f(0) = \cdots\\ f\left(\frac{x}{ts_1}, ts_2\right) = \cdots \qquad &\equiv \qquad f(\mathsf{Gnd}(x, ts_1), ts_2) = \cdots\end{aligned}$$

XII.9 Všeobecné stromy

Aké čísla kódujú všeobecné stromy?

- Rozpoznáme ich predikátom Gt:

$$\mathsf{Gt}\left(\frac{x}{ts}\right) \leftarrow \mathsf{N}(x) \wedge ???(ts)$$

- *ts* je zoznam všeobecných stromov (*les*)

  $\mathrm{Lgt}(0)$
  $\mathrm{Lgt}(t, ts) \leftarrow \mathrm{Gt}(t) \wedge \mathrm{Lgt}(ts)$
  $\mathrm{Gt}\left(\frac{x}{ts}\right) \leftarrow \mathrm{N}(x) \wedge \mathrm{Lgt}(ts)$

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

  $\mathrm{Lgt}(0)$
  $\mathrm{Lgt}\left(\frac{x}{ts_1}, ts_2\right) \leftarrow \mathrm{N}(x) \wedge \mathrm{Lgt}(ts_1) \wedge \mathrm{Lgt}(ts_2)$
  $\mathrm{Gt}\left(\frac{x}{ts}\right) \leftarrow \mathrm{N}(x) \wedge \mathrm{Lgt}(ts)$

- Tento postup – rekurziu na zoznamoch všeobecných stromov využijeme aj v ďalších úlohách

XII.10 Odbočka o zovšeobecňovaní

Všimnite si:

- Rozpoznať kód zoznamu stromov je *všeobecnejší problém* ako rozpoznať kód stromu
- Zovšeobecnenie často vedie k riešeniu alebo k efektívnemu riešeniu
- Už sme videli príklady, keď zovšeobecnenie viedlo k efektívnemu riešeniu:
  - funkcia While_fib(n, a, b) je všeobecnejšia ako $\mathrm{Fib}(n+1) =$ While_fib$(n, 0, 1)$
  - While_rev$(xs, rs) = \mathrm{Rev}(xs) \oplus rs$ je všeobecnejšia $\mathrm{Rev}(xs) =$ While_rev$(xs, 0)$
  - $\mathrm{Preordera}(t, as) = \mathrm{Preorder}(t) \oplus as$ je všeobecnejšia ako $\mathrm{Preorder}(t) =$ $\mathrm{Preordera}(t, 0)$

alebo k vôbec nejakému riešeniu:

- $\text{Full\_pre}_1(n, m)$ je všeobecnejšia, ako $\text{Full\_pre}(n) = \text{Full\_pre}_1(n, 0)$
- $\text{Num\_pre}_1(t, m)$ je všeobecnejšia, ako $\text{Num\_pre}(t) = \text{Num\_pre}_1(t, 0)$

16.2. Programovanie so všeobecnými stromami

XII.11 Programovanie so všeobecnými stromami

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

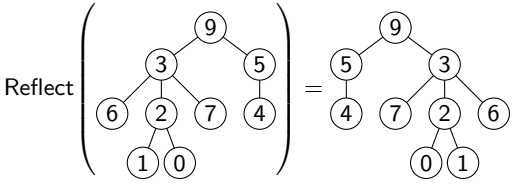
  $f_{\text{L}}(0) = \cdots$
  $f_{\text{L}}\left(\frac{x}{ts_1}, ts_2\right) = \cdots f_{\text{L}}(ts_1) \cdots f_{\text{L}}(ts_2) \cdots$
  $f\left(\frac{x}{ts}\right) = \cdots f_{\text{L}}(ts) \cdots$

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

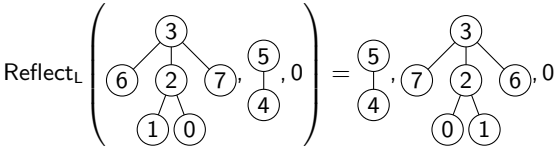
  $|0|_{\text{L}} = 0$
  $\left|\frac{x}{ts_1}, ts_2\right|_{\text{L}} = 1 + |ts_1|_{\text{L}} + |ts_2|_{\text{L}}$
  $\left|\frac{x}{ts}\right|_g = 1 + |ts|_{\text{L}}$

XII.12 Zrkadlový obraz

- Zadefinujme zrkadlový obraz stromu t – funkciu $\text{Reflect}(t)$



- Zovšeobecníme na zoznamy stromov $\mathsf{Reflect_L}(ts)$



XII.13 Zrkadlový obraz

- Skombinujeme obrátenie zoznamu a zrkadlový obraz binárneho stromu

$$\begin{array}{l} \mathsf{Rev}(0) = 0 \\ \mathsf{Rev}(x, xs) = \mathsf{Rev}(xs) \oplus (x, 0) \end{array} \times \begin{array}{l} \mathsf{Reflect}(\bullet) = \bullet \\ \mathsf{Reflect}\left(\frac{x}{\ell \mid r}\right) = \frac{x}{\mathsf{Reflect}(r) \mid \mathsf{Reflect}(\ell)} \end{array}$$

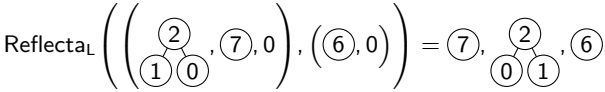
- Výsledná definícia

$$\begin{array}{l} \mathsf{Reflect_L}(0) = 0 \\ \mathsf{Reflect_L}\left(\frac{x}{ts_1}, ts_2\right) = \mathsf{Reflect_L}(ts_2) \oplus \left(\frac{x}{\mathsf{Reflect_L}(ts_1)}, 0\right) \end{array}$$

- Podobne ako Rev, aj $\mathsf{Reflect_L}$ je neefektívna
- Efektívna verzia s akumulátorom a vlastnosťou

$$\mathsf{Reflecta_L}(ts, rts) = \mathsf{Reflect_L}(ts) \oplus rts$$

Príklad:



16.3. Prechody všeobecnými stromami

XII.14 Prechody všeobecnými stromami

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

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

- Opäť zovšeobecníme na zoznamy stromov – funkcia $\text{Preorder}_{\text{L}}(ts)$

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

XII.15 Prechody všeobecnými stromami

- Podobne ako na binárnych stromoch, Preorder na všeobecných stromoch používajúci zreťazenie je neefektívny
- Efektívna verzia s akumulátorom a vlastnosťou

$$\text{Preordera}_{\text{L}}(ts, as) = \text{Preorder}_{\text{L}}(ts) \oplus as$$

- Príklad/návod:

$$\text{Preordera}_{\text{L}}\left(\left(\begin{array}{c}2\\1\quad 0\end{array}, 7, 0\right), 5,4,0\right) = 2,1,0,7,5,4,0$$

XIII. prednáška
Postfixový stroj a numerické výrazy

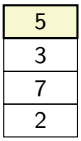
7. mája 2012

17. Postfixový stroj a numerické výrazy

17.1. Postfixový stroj

XIII.1 Postfixový stroj

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



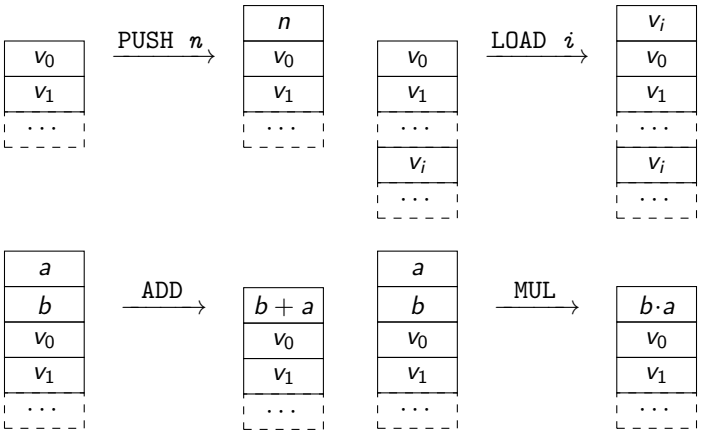
- Program: postupnosť inštrukcií

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

Inštrukcie modifikujú zásobník

Pridávajú/odoberajú čísla na vrch/z vrchu zásobníka

XIII.2 Inštrukcie postfixového stroja

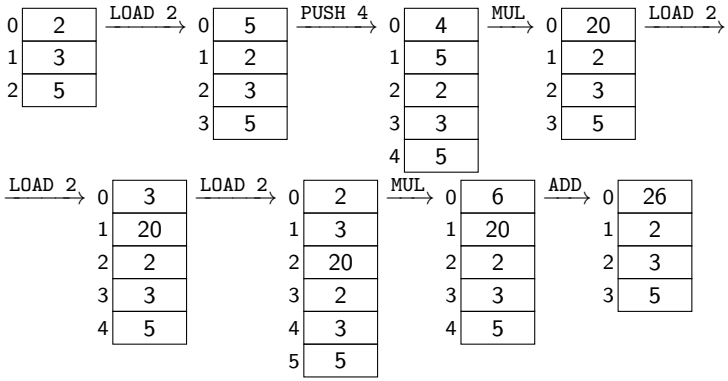


XIII.3 Príklad výpočtu programu

Počítajme program

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

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

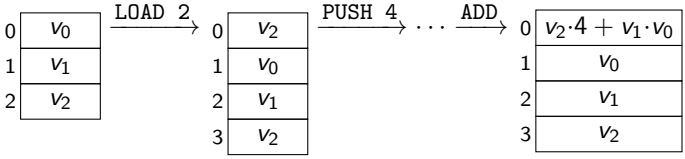


XIII.4 Príklad výpočtu programu – všeobecnejšie

Počítajme program

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

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



XIII.5 Kódovanie postfixových strojov

Postfixové stroje ľahko *zakódujeme* v CL:

- Zásobník: zoznam čísel; vrch zásobníka: prvý prvok
- Inštrukcie: párové konštruktory (podobne ako E a Nd)

  $\texttt{PUSH}(n) \equiv \mathsf{Ci}(n) = 0, n$
  $\texttt{LOAD}(i) \equiv \mathsf{Vi}(i) = 1, i$
  $\texttt{ADD} \equiv \mathsf{Ai} = 2, 0$
  $\texttt{MUL} \equiv \mathsf{Mi} = 3, 0$

  Predikát $\mathsf{Instr}(x)$ platí, ak x kóduje inštrukciu:

  $\mathsf{Instr}\,\texttt{PUSH}(n) \leftarrow \mathsf{N}(n)$
  $\mathsf{Instr}\,\texttt{LOAD}(i) \leftarrow \mathsf{N}(i)$
  $\mathsf{Instr}(\texttt{ADD})$
  $\mathsf{Instr}(\texttt{MUL})$

- Program: zoznam inštrukcií

  $\mathsf{Program}(0)$
  $\mathsf{Program}(i, is) \leftarrow \mathsf{Instr}(i) \wedge \mathsf{Program}(is)$

XIII.6 Simulácia postfixových strojov

Zakódovaný postfixový stroj môžeme *simulovať*:

- Funkcia Run(is, vs)
- Simuluje beh postfixového stroja s programom is a zásobníkom vs
- Vráti vrch zásobníka na konci programu
- Klauzálna definícia:

$$\begin{aligned}
&\mathrm{Run}(0 &&, (v, vs)) &&= v\\
&\mathrm{Run}((\texttt{PUSH}(n), is), vs &&) &&= \mathrm{Run}(is, (n, vs))\\
&\mathrm{Run}((\texttt{LOAD}(i), is), vs &&) &&= \cdots\\
&\mathrm{Run}((\texttt{ADD}, is) &&, (a, b, vs)) &&= \cdots\\
&\mathrm{Run}((\texttt{MUL}, is) &&, (a, b, vs)) &&= \cdots
\end{aligned}$$

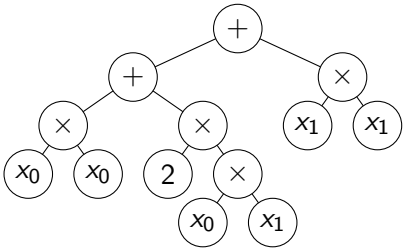
17.2. Numerické výrazy s premennými

XIII.7 Numerické výrazy s premennými

Numerický výraz s premennými je

- číselná konštanta $k \in \mathbb{N}$
  - ▶ 3, 0, 777, ...
- premenná x_i s indexom $i \in \mathbb{N}$
  - ▶ x_2, x_{17}, x_0, ...
- súčtový výraz $(t_1 + t_2)$, kde t_1 a t_2 sú numerické výrazy s premennými
  - ▶ $(x_1 + 5)$, $(x_5 + x_0)$, $(x_3 + (x_0 + x_2))$, ...
- súčinový výraz $(t_1 \times t_2)$, kde t_1 a t_2 sú numerické výrazy s premennými
  - ▶ $(2 \times x_0)$, $(x_4 \times x_1)$, $(((x_1 \times x_1) + (2 \times x_1)) + 1)$, ...

- Numerické výrazy majú stromovú štruktúru
- Napríklad $((x_0 \times x_0) + (2 \times (x_0 \times x_1))) + (x_1 \times x_1)$



XIII.9 Kódovanie numerických výrazov

Numerické výrazy môžeme v CL zakódovať podobne ako binárne stromy:

- Párovacie konštruktory (podobne ako E a Nd)

$$
\begin{array}{lll}
n^\bullet & \equiv \mathsf{Ct}(n) & = 0, n \\
x_i^\bullet & \equiv \mathsf{Vt}(i) & = 1, i \\
t_1 +^\bullet t_2 & \equiv \mathsf{At}(t_1, t_2) & = 2, t_1, t_2 \\
t_1 \times^\bullet t_2 & \equiv \mathsf{Mt}(t_1, t_2) & = 3, t_1, t_2
\end{array}
$$

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

$$
\begin{array}{ll}
\mathsf{Term}(n^\bullet) & \leftarrow \mathsf{N}(n) \\
\mathsf{Term}(x_i^\bullet) & \leftarrow \mathsf{N}(i) \\
\mathsf{Term}(t_1 +^\bullet t_2) & \leftarrow \mathsf{Term}(t_1) \wedge \mathsf{Term}(t_2) \\
\mathsf{Term}(t_1 \times^\bullet t_2) & \leftarrow \mathsf{Term}(t_1) \wedge \mathsf{Term}(t_2)
\end{array}
$$

- Výraz $((x_0 \times x_0) + (2 \times (x_0 \times x_1))) + (x_1 \times x_1)$ zakódujeme ako

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

XIII.11 Vyhodnocovanie numerických výrazov

Ohodnotenie premenných:

- zoznam $vs = v_0, v_1, v_2, \ldots$
- priraďuje premennej x_i hodnotu v_i

Hodnota (*denotácia*) výrazu pre dané ohodnotenie premenných – funkcia $\mathsf{Den}(t, vs) \equiv [\![t]\!]^{vs}$:

- Napríklad:

$$\begin{aligned}&t = ((x_0^\bullet \times^\bullet x_0^\bullet) +^\bullet (2^\bullet \times^\bullet (x_0^\bullet \times^\bullet x_1^\bullet))) +^\bullet (x_1^\bullet \times^\bullet x_1^\bullet)\\ &[\![t]\!]^{3,5,7,0} = ((3{\cdot}3) + (2{\cdot}(3{\cdot}5))) + (5{\cdot}5)\end{aligned}$$

- Všeobecne:

$$\begin{aligned}[\![n^\bullet]\!]^{vs} &= n\\ [\![x_i^\bullet]\!]^{vs} &= \cdots\\ [\![t_1 +^\bullet t_2]\!]^{vs} &= \cdots\\ [\![t_1 \times^\bullet t_2]\!]^{vs} &= \cdots\end{aligned}$$

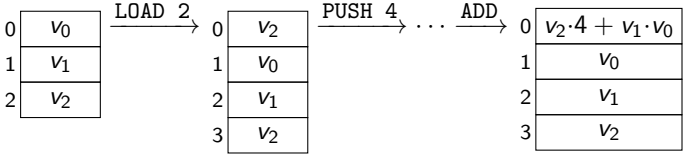
17.3. Kompilácia

XIII.12 Programy a výrazy – príklad

Program

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

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



vypočíta na vrch zásobníka hodnotu numerického výrazu

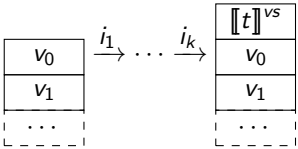
$$(x_2 \times 4) + (x_1 \times x_0)$$

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

$$\begin{aligned}&\mathrm{Run}((\mathtt{LOAD}(2), \mathtt{PUSH}(4), \mathtt{MUL}, \mathtt{LOAD}(2), \mathtt{LOAD}(2), \mathtt{MUL}, \mathtt{ADD}, 0), vs)\\ &\quad = [\![(x_2^\bullet \times^\bullet 4^\bullet) +^\bullet (x_1^\bullet \times^\bullet x_0^\bullet)]\!]^{vs}\end{aligned}$$

XIII.13 Programy a výrazy – kompilácia

- Numerické výrazy – jazyk vyššej úrovne
- Programy postfixového stroja – jazyk nižšej úrovne
- Preklad výrazov do programov – *kompilácia*
- Kompilátor – funkcia Comp(t)
- Vytvorí program $i_1, i_2, \ldots, i_k, 0$, ktorý na vrch zásobníka s hodnotami premenných vypočíta hodnotu výrazu t



$$\mathsf{Term}(t) \to \mathsf{Program}\,\mathsf{Comp}(t)$$
$$\mathsf{Term}(t) \to \mathsf{Run}(\mathsf{Comp}(t), vs) = [\![t]\!]^{vs}$$

XIII.14 Kompilácia – príklad

Videli sme, že

$$\mathsf{Run}((\mathtt{LOAD}(2), \mathtt{PUSH}(4), \mathtt{MUL}, \mathtt{LOAD}(2), \mathtt{LOAD}(2), \mathtt{MUL}, \mathtt{ADD}, 0), vs) = [\![(x_2^\bullet \times^\bullet 4^\bullet) +^\bullet (x_1^\bullet \times^\bullet x_0^\bullet)]\!]^{vs}$$

Chceme, aby kompilátor skompiloval výraz

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

do programu

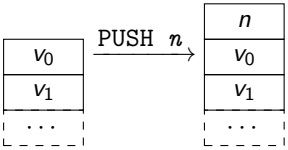
$$\mathtt{LOAD}(2), \mathtt{PUSH}(4), \mathtt{MUL}, \mathtt{LOAD}(2), \mathtt{LOAD}(2), \mathtt{MUL}, \mathtt{ADD}, 0$$

teda

$$\mathsf{Comp}((x_2^\bullet \times^\bullet 4^\bullet) +^\bullet (x_1^\bullet \times^\bullet x_0^\bullet)) = \mathtt{LOAD}(2), \mathtt{PUSH}(4), \mathtt{MUL}, \mathtt{LOAD}(2), \mathtt{LOAD}(2), \mathtt{MUL}, \mathtt{ADD}, 0$$

XIII.15 Kompilácia – konštanty

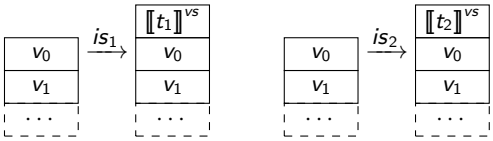
- Hodnotou výrazu $n^\bullet$ je číslo n
- Aký program vypočíta na vrch zásobníka hodnotu n?



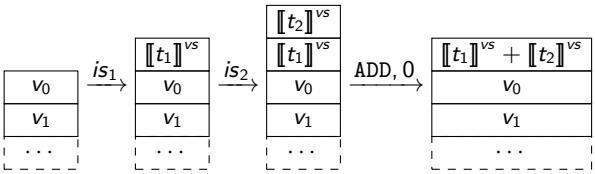
- Klauzula kompilátora:

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

- Hodnotou výrazu $t_1 +^\bullet t_2$ je číslo $[\![t_1]\!]^{vs} + [\![t_2]\!]^{vs}$
- Akým programom ju vypočítame?
  1. Skompilujeme výrazy t_1 a t_2 Dostaneme programy $\mathrm{Comp}(t_1) = is_1$ a $\mathrm{Comp}(t_2) = is_2$



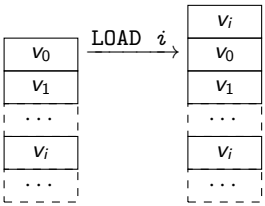
  2. Zreťazíme ich a pridáme inštrukciu pre sčítanie



- Klauzula kompilátora:

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

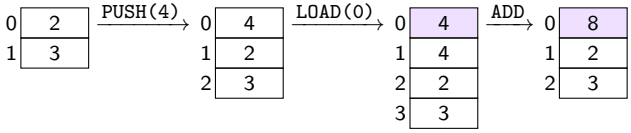
- Hodnotou výrazu $x_i^\bullet$ je číslo $vs[i]$
- Aký program dá na vrch zásobníka hodnotu $vs[i]$?



- Skompilujme $4^\bullet +^\bullet x_0^\bullet$:

$$\texttt{PUSH}(4), \texttt{LOAD}(0), \texttt{ADD}, 0$$

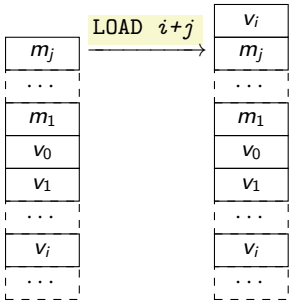
- Otestujme s ohodnotením $2, 3, 0$:



ZLE! `LOAD(0)` malo načítať 2

XIII.18 Kompilácia – medzivýsledky

- Kompilátor si musí pamätať počet j *medzivýsledkov* v zásobníku: $\text{Comp}_1(t, j)$
- $\text{Comp}(t) = \text{Comp}_1(t, 0)$
- Aký program dá na vrch zásobníka hodnotu $vs[i]$? $\text{Comp}_1(x_i^\bullet, j) = ?$



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

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

XIV. prednáška
Generovanie XML/XHTML

14. mája 2012

18. XML/XHTML

18.1. Štruktúra

XIV.1 Čo je XML/XHTML

XML/XHTML je formát na zápis štruktúrovaného textu

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Príklad</title>
  </head>
  <body>
    <p>Príklad
    <abbr title="Extensible Hypertext
          Markup Language">XHTML</abbr><br/>
    dokumentu.</p>
  </body>
</html>
```

XIV.2 Štruktúra XML/XHTML dokumentu

Štruktúra v XML dokumente – *elementy*

- úsek textu medzi otváracím tagom ( ) a príslušným zatváracím tagom ( )

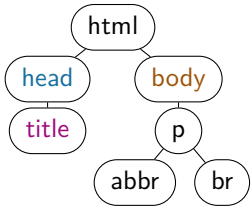
```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Príklad</title>
  </head>
  <body>
```

```
    <p>Príklad
    <abbr title="Extensible Hypertext
          Markup Language">XHTML</abbr><br/>
    dokumentu.</p>
  </body>
</html>
```

XIV.3 Strom elementov

Elementy vytvárajú stromovú štruktúru:

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



Čo v strome chýba?

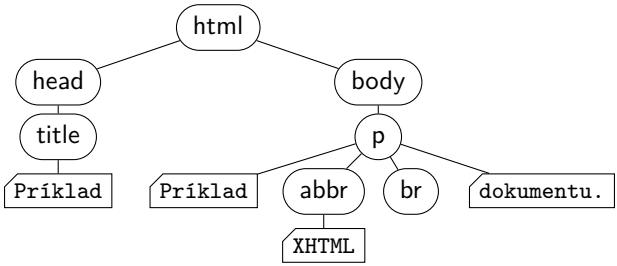
XIV.4 Neštruktúrovaný text

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Príklad</title>
  </head>
  <body>
    <p>Príklad
    <abbr title="Extensible Hypertext
          Markup Language">XHTML</abbr><br/>
    dokumentu.</p>
  </body>
</html>
```

V strome elementov chýba neštruktúrovaný text

XIV.5 Strom elementových a textových uzlov

Pridáme uzly pre neštruktúrovaný text:



Všeobecný strom s dvoma druhmi uzlov:

- (elementové uzly) – ľubovoľne veľa detí (aj 0, br)
- [textové uzly] – listy, žiadne deti

18.2. Kódovanie

XIV.6 Kódovanie stromu dokumentu v CL

- Strom XML dokumentu má 2 druhy uzlov
- Každý druh zakódujeme osobitným konštruktorom (podobne ako E a Nd)
- Textové uzly: Xs(t)

  t – text v uzle – reťazec
- Elementové uzly: Xe(n, as, cs)

  n – meno elementu (html, body, p, ...) – reťazec

  as – atribúty elementu

  cs – deti elementu

XIV.7 Kódovanie atribútov elementu

Xe(n, as, cs)

```
<a href="http://example.com/" class="external">link</a>
```

- Atribút je dvojica *meno*=*hodnota*
- Kódujeme dvojicou reťazcov:

  $\text{Attr}(n, v) \leftarrow \text{Str}(n) \wedge \text{Str}(v)$

- Všetky atribúty elementu zakódujeme zoznamom:

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

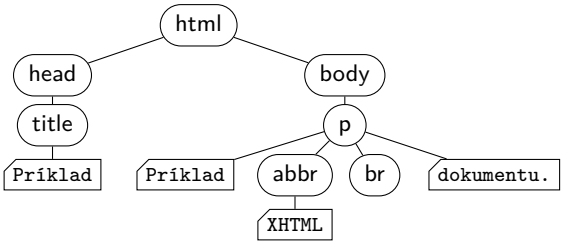
- Napríklad:

  $$(\text{'href'}, \text{'http://example.com/'}), (\text{'class'}, \text{'external'}), 0$$

XIV.8 Kódovanie detí elementu

Xe(n, as, cs)

- Elementové uzly majú rôzny počet detí:



- Deti zakódujeme do *zoznamu* uzlov XML stromu

XIV.9 Kódovanie XML dokumentu – príklad

```
Xe('html', (('xmlns', 'http://www.w3.org/1999/xhtml'), 0),
    Xe('head', 0,
        Xe('title', 0, Xs('Príklad'), 0),
        0),
    Xe('body', 0,
        Xe('p', 0,
            Xs('Príklad '),
            Xe('abbr',(('title','Extensible . . . Language'),0),
                Xs('XHTML'),
                0),
            Xe('br',0,0),
            Xs(' dokumentu.'),
            0),
        0),
    0)
```

XIV.10 Kódovanie XML – predikát

- Štruktúru binárneho stromu sme popísali predikátom:

  $\mathsf{Bt}(\bullet)$
  $\mathsf{Bt}\left(\frac{x}{\ell \mid r}\right) \leftarrow \mathsf{N}(x) \wedge \mathsf{Bt}(\ell) \wedge \mathsf{Bt}(r)$

- Štruktúru všeobecného stromu sme popísali predikátmi:

  $\mathsf{Lgt}(0)$
  $\mathsf{Lgt}\left(\frac{x}{ts_1}, ts_2\right) \leftarrow \mathsf{N}(x) \wedge \mathsf{Lgt}(ts_1) \wedge \mathsf{Lgt}(ts_2)$

  $\mathsf{Gt}\left(\frac{x}{ts}\right) \leftarrow \mathsf{N}(x) \wedge \mathsf{Lgt}(ts)$

- Ako popíšeme štruktúru XML stromu?
  - Predikát $\mathsf{X}(x)$ – x kóduje uzol XML stromu
  - Pomocný predikát $\mathsf{Lx}(xs)$ – xs je zoznam XML uzlov

- Prirodzená definícia *vzájomnou rekurziou*:

  $\mathsf{X}\,\mathsf{Xe}(n, as, cs) \leftarrow \mathsf{Str}(n) \wedge \mathsf{Lattr}(as) \wedge \mathsf{Lx}(cs)$
  $\mathsf{X}\,\mathsf{Xs}(t) \leftarrow \mathsf{Str}(t)$

  $\mathsf{Lx}(0)$
  $\mathsf{Lx}(x, xs) \leftarrow \mathsf{X}(x) \wedge \mathsf{Lx}(xs)$

- CL neumožňuje vzájomnú rekurziu:

  $\mathsf{Lx}(0)$
  $\mathsf{Lx}(\mathsf{Xe}(n, as, cs), xs) \leftarrow \mathsf{Str}(n) \wedge \mathsf{Lattr}(as) \wedge \mathsf{Lx}(cs) \wedge \mathsf{Lx}(xs)$
  $\mathsf{Lx}(\mathsf{Xs}(t), xs) \leftarrow \mathsf{Str}(t) \wedge \mathsf{Lx}(xs)$

  $\mathsf{X}\,\mathsf{Xe}(n, as, cs) \leftarrow \mathsf{Str}(n) \wedge \mathsf{Lattr}(as) \wedge \mathsf{Lx}(cs)$
  $\mathsf{X}\,\mathsf{Xs}(t) \leftarrow \mathsf{Str}(t)$

- Ak v CL zakódujeme strom XML dokumentu, zobrazíme ho ako skutočné XML formátom Xml:

  Query: $\mathsf{Xe}(\text{'pokus'}, 0, \mathsf{Xs}(\text{'text'}), 0), 0 = xs : \mathsf{Xml}$

  Results: $xs =$ <pokus>text</pokus>

- Formátom Xml môžeme zobrazovať *iba* hodnoty xs:
  - jednoprvkové zoznamy uzlov: $\mathsf{Lx}(xs) \wedge \mathsf{L}(xs) = 1$
  - jediným prvkom je Xe
  - ▶ XML dokument musí mať jeden koreňový element

- Skratková funkcia Xhtml(t, cs) – kostra XHTML dokumentu

  t – titulok – reťazec

  cs – obsah tela (body) – Lx

  Formát Xml zobrazí výsledok ako skutočné XHTML

18.3. Neformálne typovanie

XIV.13 Neformálne typovanie

- CL je beztypový jazyk – všetky dáta sú čísla
- Môžeme typovať neformálne predikátmi
- $f :: T_1 \to T_2$ znamená:
  - T_1, T_2 sú predikáty
  - argument funkcie f je typu T_1
  - výsledok je typu T_2
  - skratka za:

$$T_1(x) \to T_2\, f(x)$$

- Napríklad funkcie na binárnych stromoch:

$$\begin{aligned} \text{Preorder} &:: \text{Bt} \to \text{Ln} \\ \text{Insert} &:: (\text{Bt}, \text{N}) \to \text{Bt} \\ \text{Extract\_max} &:: \text{Bt} \to (\text{N}, \text{Bt}) \end{aligned}$$

- Ako otypujeme Xe, Xs, Xhtml?

18.4. Generovanie

XIV.14 Generovanie XML – príklad

Zadefinujme funkciu Mk_ul(*xs*), ktorá zo zoznamu reťazcov *xs* vytvorí nečíslovaný XHTML zoznam (`ul`), ktorého položky (`li`) obsahujú reťazce zo zoznamu *xs*

- Otypujme túto funkciu – vytvorí jeden uzol XML stromu:

$$\text{Mk\_ul} :: \text{Lstr} \to \text{X}$$

- Naprogramujme ju:

– Potrebujeme pomocnú funkciu Mk_lis(xs), ktorá vytvorí zoznam položiek (uzlov XML stromu):

$$\text{Mk\_lis} :: \text{Lstr} \to \text{Lx}$$

$\text{Mk\_lis}(0) = 0$
$\text{Mk\_lis}(x, xs) = \text{Xe}(\text{'li'}, 0, \text{Xs}(x), 0), \text{Mk\_lis}(xs)$

– Vytvorené položky použijeme ako deti elementu `ul`:

$\text{Mk\_ul}(xs) = \text{Xe}(\text{'ul'}, 0, \text{Mk\_lis}(xs))$

19. Organizácia: Opravný test a skúška

XIV.15 Náhradný/opravný test

Termín: pondelok 21. mája o 14:00 v H6

Účel: nahradiť skóre z jedného semestrálneho testu

Materiál: ako nahrádzaný test

Podmienky:

- *záväzné prihlásenie* do 21. mája 0:00 (AIS)
  - účasť bez prihlásenia nie je možná
- výber nahrádzaného testu priamo na termíne
  - neprítomní prihlásení: 2. test
- nové skóre nahradí pôvodné skóre *bez ohľadu na ich vzájomný rozdiel*
  - môžete si pohoršiť
  - skóre neprítomných prihlásených je 0

Termíny: pondelky o 14:00 v H6

riadne: 21. a 28. mája, 4. júna

1. opravné: 4. júna a 18. júna

2. opravné: 18. júna a 25. júna

najviac 25 študentov na termín
(4. jún – riadny termín: 12; opravný termín: 13)

Podmienka: skóre z testov ≥ 30

Materiál: úlohy najmä podľa ex11–ex14

Konzultácie: nahlásiť deň vopred sa e-mailom na udp zavináč lists.dai.fmph.uniba.sk

Termíny:

- piatok 18. mája 9:00–11:30 H6
- štvrtky 24. a 31. mája, 14. a 21. júna vždy o 13:00 I-33b, podľa potreby presun do voľnej počítačovej haly