Prednášky z Úvodu do deklaratívneho programovania

Ján Kľuka

Letný semester 2014/2015

Obsah

I. prednáška
Organizácia kurzu
Deklaratívne programovanie v CL
Explicitné definície

23. februára 2015

1. Organizácia kurzu

1.1. Rozvrh a kontakty

Prednášky pondelok 9:50, poslucháreň A

Cvičenia

streda	14:50	H6	1AIN{1,2}	Kľuka, Komara
streda	16:30	H6	1AIN{3,4}	Kľuka, Komara

Druháci a tretiaci: Prihláste sa na termín podľa vlastného výberu (**povinne do 28. 2.**) (e-mail↓)

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

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

Na test môžete prísť iba na svoj termín.

2INFb 1-INF-465 Deklaratívne programovanie: Spoločné prednášky, samostatné cvičenia — utorok 16:30 H3 s Jánom Komarom

Konzultácie TBA

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

E-mail *udp{z@vináč}lists.dai.fmph.uniba.sk*

1.2. Pravidlá hodnotenia

I.2 Organizácia kurzu — hodnotenie

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

neúčasť $\wedge$ *ospravedlnenie* (do 3 dní [Š. p., čl. 21]) $\Rightarrow$ náhradný termín (nasledujúci týždeň)

Prémiové DÚ Nepravidelne zadávané v cvičeniach

Opravný test Opravenie 1 semestrálneho testu podľa výberu (rovnaká časť učiva) v 1. týždeň skúškového obdobia

Platí skóre z opravného testu, aj keby bolo nižšie!

Podmienka pripustenia ku skúške: zisk ≥ 30 bodov za semester

Skúška test za *40* bodov

Známky
A ≥ 90 bodov
B ≥ 80 bodov
C ≥ 70 bodov
D ≥ 60 bodov
E ≥ 50 bodov
FX < 50 bodov

2. Deklaratívne programovanie

I.3 Deklaratívne programovanie

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

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

I.4 Tvorba správnych programov

Tvorba správneho programu má 3 časti:

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

Implementácia samotný program, podrobný návod na počítanie výstupov

Verifikácia overenie, že program má požadované vlastnosti (spĺňa špecifikáciu)

Deklaratívne programovanie:

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

2.1. Príklad

I.5 Príklad — zadanie a špecifikácia

Neformálne zadanie

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

Špecifikácia

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

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

I.6 Príklad — implementácia

Implementácia

- Program realizuje euklidovský algoritmus
- Využíva vlastnosti deliteľnosti:

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

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

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

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

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 klauzúl

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

Má program požadovanú vlastnosť?

Testovanie: *Výpočtom* overíme pre *niektoré* vstupy

Verifikácia: *Dôkazom* overíme pre *všetky* vstupy

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

3. Jazyk CL

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

3.1. Základy syntaxe

I.9 Základy syntaxe — premenné

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

Príklad 2. x, y, $a3$, $zoznam99$;
~~X~~, ~~$velkePismeno$~~, ~~$\_podciar\_kovnik$~~, ~~$y2k$~~, ~~$p123$~~, ~~$kŕdeľ$~~

I.10 Základy syntaxe — funkčné symboly a ich arita

Definícia 3. *Funkčný symbol* je postupnosť znakov začínajúca veľkým písmenom a pokračujúca ľubovoľnou postupnosťou malých písmen, číslic a podčiarkovníkov.
Každý funkčný symbol má priradený počet argumentov — *aritu*.
Funkčný symbol s aritou n nazývame *n-árny*, špeciálne:
$n = 0$ *nulárny* f. s. — *konštanta*, $n = 2$ *binárny* f. s.,
$n = 1$ *unárny* f. s., $n = 3$ *ternárny* f. s.

Dohoda: F, G, H označujú funkčné symboly.

Príklad 4. F, Perms, Tree2string, F_n1_m1, Q54321;
~~malepismeno~~, ~~VelkePismeno~~, ~~_podciarkovnik~~, ~~Zvyšok~~

I.11 Základy syntaxe — konštanty

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

Príklad 6. 0, 8, 2020, 0123456789876543210

Ďalšie špeciálne druhy konštánt neskôr.

I.12 Termy

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

- premenná,

- konštanta (ľubovoľného druhu),
- *aplikácia niektorej zabudovanej binárnej funkcie*

  $$(\tau + \sigma)\ \ (\tau \dot{-} \sigma)\ \ (\tau{\cdot}\sigma)\ \ (\tau \div \sigma)\ \ (\tau \bmod \sigma)\ \ (\tau, \sigma)\ \ (\tau \oplus \sigma)$$

  ak τ a σ sú termy,
- *aplikácia n-árneho funkčného symbolu F* pre $n \geq 1$

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

  ak τ_1, τ_2, ..., τ_n sú termy.

Nič iné nie je termom.

Dohoda: τ, σ označujú termy.

I.13 Termy — príklady a zjednodušenia

Príklady termov:

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

Nemusíme písať všetky zátvorky:

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

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

Zabudované binárne funkcie:

			priorita	implicitné zátvorkovanie pri rovnakej priorite
$\cdot$	$\div$	mod	najvyššia	vľavo
$+$	$\dot{-}$			vľavo
$\oplus$				vľavo
,			najnižšia	vpravo

3.2. Zabudované funkcie

I.14 Niektoré funkcie zabudované v CL

Binárne aritmetické funkcie *na prirodzených číslach*:

$+$ sčítanie (v zdrojovom kóde „+“)

$\cdot$ násobenie (v zdrojovom kóde „*“)

$\dot{-}$ *modifikované* odčítanie (v zdrojovom kóde „-“)

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

$\div$ delenie (v zdrojovom kóde „/“) a

$\mathbf{mod}$ zvyšok po delení („`mod`“), ktoré majú nasledujúce vlastnosti:

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

Zabudovaných je aj niekoľko ďalších funkcií — neskôr

4. Explicitné definície

I.15 Moduly a definície funkcií

Definícia 8. CL *modul* je postupnosť definícií funkcií

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

$F(x_1, \ldots, x_n) = \tau_1 \leftarrow \varphi_1$
$\ldots$
$F(x_1, \ldots, x_n) = \tau_k \leftarrow \varphi_k$

skonštruovaná určitým spôsobom (postupne sa ho naučíme).

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

I.16 Explicitné definície

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

Príklad 11. Najjednoduchšie explicitné definície majú iba jednu rovnosť:
$F(x_1, \ldots, x_n) = \tau$

- F je nový n-árny funkčný symbol
- τ je term obsahujúci iba premenné $x_1, \ldots, x_n$, konštanty a symboly predtým zadefinovaných funkcií

$\mathrm{Z}(x) = 0$ $\quad \mathrm{Twice}(x) = 2{\cdot}x$ $\quad \mathrm{Square}(x) = x{\cdot}x$
$\mathrm{Binomial1}(a, b) = a{\cdot}a + 2{\cdot}a{\cdot}b + b{\cdot}b$
$\mathrm{Binomial2}(a, b) = \mathrm{Square}(a + b)$

4.1. Explicitné definície s jednou podmienkou

I.17 Explicitné definície s podmienkami

- Zložitejšie definície majú viacero klauzúl

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

- Argumenty funkcie musia byť označené *rovnakými premennými*:

$\mathrm{Min}(x, y) = x \leftarrow x \leq y$
~~$\mathrm{Min}(a, b) = b \leftarrow a > b$~~

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

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

- CL ***nerozpozná výlučnosť hocijakých podmienok***

  Postupne sa naučíme, ktoré sady podmienok rozpoznáva

I.18 Diskriminácie

Rozpoznateľnú sadu podmienok voláme *diskriminácia*

Niektoré diskriminácie (τ a σ sú ľubovoľné termy):

- diskriminácia na rovnosť

  $F(\cdots) = \cdots \leftarrow \tau = \sigma$
  $F(\cdots) = \cdots \leftarrow \tau \neq \sigma$
  $\neg(\tau = \sigma \wedge \tau \neq \sigma)$
  $\tau = \sigma \vee \tau \neq \sigma$

- dichotómia

  $F(\cdots) = \cdots \leftarrow \tau \leq \sigma$
  $F(\cdots) = \cdots \leftarrow \tau > \sigma$
  $\neg(\tau \leq \sigma \wedge \tau > \sigma)$
  $\tau \leq \sigma \vee \tau > \sigma$

Teda

- môžeme napísať napríklad:

  $\mathsf{F}(x) = x \quad \leftarrow x{+}7 = x{\cdot}7$
  $\mathsf{F}(x) = x{+}1 \leftarrow x{+}7 \neq x{\cdot}7$

  $\mathsf{G}(x,y) = x{\cdot}y \quad \leftarrow \mathsf{Square}(y) > x$
  $\mathsf{G}(x,y) = x{+}y \leftarrow \mathsf{Square}(y) \leq x$

- nemôžeme napísať napríklad:

  $\mathsf{F}(x,y) = 0 \quad \leftarrow x \quad = y{\cdot}y$
  $\mathsf{F}(x,y) = 1 \leftarrow y{\cdot}y \neq \quad x$

  $\mathsf{G}(x,y,z) = y{+}z \leftarrow y \leq x$
  $\mathsf{G}(x,y,z) = x{+}z \leftarrow y \geq x$

4.2. Explicitné definície s viacerými podmienkami

I.19 Kombinované podmienky

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

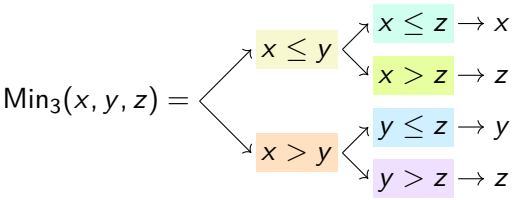
$$\mathrm{Min}_3(x,y,z) = x \leftarrow x \leq y \wedge x \leq z$$
$$\mathrm{Min}_3(x,y,z) = y \leftarrow y \leq z \wedge y \leq x$$
$$\mathrm{Min}_3(x,y,z) = z \leftarrow z \leq x \wedge z \leq y$$

I.20 Ako kombinovať podmienky

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

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

vzniknú vnorením diskriminácií:



I.21 Ako kombinovať podmienky — recept

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

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

$\mathrm{Min}_3(x, y, z) = ? \leftarrow x \leq y$ napíšeme potrebnú podmienku
$\mathrm{Min}_3(x, y, z) = ? \leftarrow x > y$ nezabudneme na ostatné časti jej diskriminácie

$\mathrm{Min}_3(x, y, z) = x \leftarrow x \leq y \wedge x \leq z$ rozvetvíme klauzulu
$\mathrm{Min}_3(x, y, z) = z \leftarrow x \leq y \wedge x > z$ ďalšou potrebnou diskrimináciou
$\mathrm{Min}_3(x, y, z) = ? \leftarrow x > y$

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

4.3. Verifikácia explicitne definovaných funkcií

I.22 Špecifikácia minima troch

Hodnotou funkcie Min_3 je najmenší z jej argumentov, teda hodnota Min_3 sa

- rovná niektorému z argumentov:

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

- nanajvýš rovná každému argumentu:

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

Dokážme — verifikujme, že tieto vlastnosti platia pre našu definíciu

$\mathrm{Min}_3(x, y, z) = x \leftarrow x \leq y \wedge x \leq z$ (1)
$\mathrm{Min}_3(x, y, z) = z \leftarrow x \leq y \wedge x > z$ (2)
$\mathrm{Min}_3(x, y, z) = y \leftarrow x > y \wedge y \leq z$ (3)
$\mathrm{Min}_3(x, y, z) = z \leftarrow x > y \wedge y > z$ (4)

Ako na to?

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

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

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

- Ak $x \le y$, hodnotu Min_3 určí niektorá z klauzúl (1) a (2) na základe porovnania x a z — vnorená analýza prípadov:
  - Ak $x \le z$, podľa (1) je $\mathrm{Min}_3(x,y,z) = x$. Ľahko overíme, že platia všetky konjunkty dokazovanej vlastnosti $(*)$: zrejme platí $x \le x$ (a) a podľa aktuálnych predpokladov $x \le y$ (b) a $x \le z$ (c).
  - Ak $x > z$, podľa (2) je $\mathrm{Min}_3(x,y,z) = z$. Opäť overíme platnosť všetkých konjunktov $(*)$: Podľa aktuálnych predpokladov platí $z < x$ (a), $z < x \le y$ (b) a zrejme $z \le z$ (c).
- Ak $x > y$, hodnotu Min_3 určíme porovnaním y a z. Postup je podobný ako v predchádzajúcom prípade.

II. prednáška
Rekurzia

2. marca 2015

4.4. Opakovanie

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 — rozpoznávané kombinácie podmienok

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

  vzájomne výlučné, pokrývajúce všetky prípady

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

II.2 Skladanie diskriminácií

- Ak potrebujeme viac ako jednu podmienku, musíme správne *skladať diskriminácie*:

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

$\mathrm{Min}_3(x, y, z) = ? \leftarrow x \leq y$ napíšeme potrebnú podmienku
$\mathrm{Min}_3(x, y, z) = ? \leftarrow x > y$ nezabudneme na ostatné časti jej diskriminácie

$\mathrm{Min}_3(x, y, z) = x \leftarrow x \leq y \wedge x \leq z$ rozvetvíme klauzulu
$\mathrm{Min}_3(x, y, z) = z \leftarrow x \leq y \wedge x > z$ ďalšou potrebnou diskrimináciou
$\mathrm{Min}_3(x, y, z) = ? \leftarrow x > y$

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

- Chýba nám *opakovanie*

5. Rekurzívne definície

II.3 Rekurzia — princíp

- Opakovanie v deklaratívnom programovaní — rekurzia
- *Rekurzívna* definícia vyjadruje

  hodnotu funkcie pre nejaké argumenty

  pomocou hodnoty *tej istej funkcie*

  pre *jednoduchšie argumenty*

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

- Rôzne druhy rekurzie — rôzne zjednodušenia rekurzívnych argumentov

5.1. Primitívna rekurzia

II.4 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 $\mathrm{Sum}(n) = \sum_{i=0}^{n} i$ pomocou diskriminácie na rovnosť:

$$\begin{array}{l|l} \sum_{i=0}^{n} i = 0 \leftarrow n = 0 & \mathrm{Sum}(n) = 0 \leftarrow n = 0 \\ \sum_{i=0}^{n} i = n + \sum_{i=0}^{n \dot{-} 1} i \leftarrow n \neq 0 & \mathrm{Sum}(n) = n + \mathrm{Sum}(n \dot{-} 1) \leftarrow n \neq 0 \end{array}$$

Definícia 12. Funkcia F s aritou $n+1$ je definovaná *primitívnou rekurziou* z n-árnej funkcie G a $(n+2)$-árnej funkcie H, ak platí:
$F(\ 0\ , x_1, \ldots, x_n) = G(x_1, \ldots, x_n)$
$F(i+1, x_1, \ldots, x_n) = H(i, F(i, x_1, \ldots, x_n), x_1, \ldots, x_n)$

II.5 Primitívna rekurzia — symbolický výpočet

- Výpočet *prepisovaním výrazov* do základného tvaru — čísla
- Prepisuje sa *najľavejší najvnútornejší* výraz, ktorý nie je v základnom tvare

  $\Rightarrow$ Najprv argumenty, potom aplikácia funkcie
- Aplikácia funkcie F sa prepisuje podľa klauzúl definície F

Príklad 13.

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

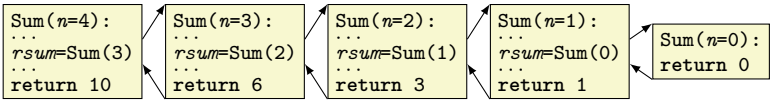
- CL v skutočnosti kompiluje program do *bajtkódu* (strojový kód virtuálneho stroja)
- Čo je potrebné na strojovú realizáciu rekurzívneho výpočtu?

  Zásobník
- V Pythone:

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

$$\mathrm{Sum}(n) = 0 \qquad \leftarrow n = 0$$
$$\mathrm{Sum}(n) = n + \mathrm{Sum}(n \dot{-} 1) \leftarrow n \neq 0$$

- Schematicky:



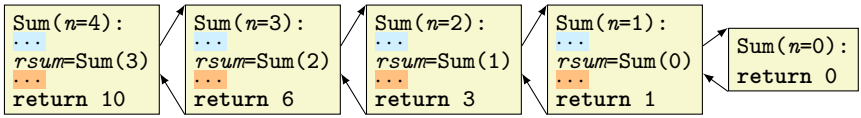
Pozorovanie 14. *Rekurzia je dvojfázový proces:*

1. *príprava rekurzívneho volania*
2. *použitie jeho výsledku*

Príklad 15.

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

$$\mathrm{Sum}(n) = 0 \qquad \leftarrow n = 0$$
$$\mathrm{Sum}(n) = n + \mathrm{Sum}(n \dot{-} 1) \leftarrow n \neq 0$$

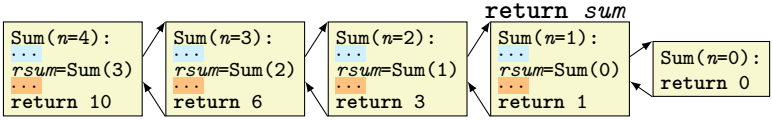


Pozorovanie 16. *Ak sa v definícii funkcie vyskytuje iba jedno rekurzívne volanie a nemenia sa parametre (iné argumenty ako klesajúce rekurzívne), dá sa rekurzia počítať zdola nahor cyklom (iteráciou). Rekurzívne volanie nahradí pomocná premenná.*

Príklad 17.

$$\mathrm{Sum}(n) = 0 \qquad \leftarrow n = 0$$
$$\mathrm{Sum}(n) = n + \mathrm{Sum}(n \dot{-} 1) \leftarrow n \neq 0$$

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



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

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

- 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 $\mathrm{I}^n_k(x_1, \ldots, x_k, \ldots, x_n) = x_k$
- 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. Všeobecná (course-of-values) rekurzia

II.10 Všeobecná (course-of-values) rekurzia

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

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

- Definícia pomocou všeobecnej rekurzie:

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

  Je zaručené zmenšovanie rekurzívneho argumentu?

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

II.11 Všeobecná rekurzia s viacerými rekurzívnymi volaniami

- 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 všeobecnou rekurziou
  - s *dvoma* rekurzívnymi volaniami, a
  - so zloženými podmienkami:

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

5.3. Všeobecná rekurzia so substitúciou v parametri

II.12 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 všeobecnou rekurziou:

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

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

5.4. Verifikácia rekurzívne definovaných funkcií indukciou

II.13 Verifikácia primitívne rekurzívnych funkcií

Funkcia

$$\begin{aligned} \mathrm{Sum}(n) &= 0 && \leftarrow n = 0 \\ \mathrm{Sum}(n) &= n + \mathrm{Sum}(n \dot{-} 1) && \leftarrow n \neq 0 \end{aligned}$$

má dobre známu vlastnosť:

$$\mathrm{Sum}(n) = \frac{n \cdot (n+1)}{2} \qquad (*)$$

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 $\mathrm{Sum}(n) = n + \mathrm{Sum}(n \dot{-} 1)$ 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

  $$\mathrm{Sum}(n \dot{-} 1) = \frac{(n \dot{-} 1) \cdot n}{2},$$

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

  Inak povedané, potrebujeme *indukčný predpoklad*.

Začnime teda odznova:

II.14 Primitívna rekurzia a matematická indukcia

$$\begin{aligned} \mathrm{Sum}(n) &= 0 && \leftarrow n = 0 \\ \mathrm{Sum}(n) &= n + \mathrm{Sum}(n \dot{-} 1) && \leftarrow n \neq 0 \end{aligned} \qquad \mathrm{Sum}(n) = \frac{n \cdot (n+1)}{2} \qquad (*)$$

Dôkaz $(*)$ matematickou indukciou na n:

1. Báza: Dokážme $(*)$ pre 0:

$$\mathsf{Sum}(0) \stackrel{\mathsf{def}}{=} 0 = \tfrac{0\cdot(0+1)}{2}\,.$$

2. Indukčný krok:

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

   Dokážme pre $n+1$:

$$\mathsf{Sum}(n+1) \stackrel{\mathsf{def}}{=} n+1+\mathsf{Sum}(n+1\dot{-}1) \stackrel{\mathsf{IP}}{=}$$
$$\stackrel{\mathsf{IP}}{=} n+1+\tfrac{n\cdot(n+1)}{2} = \tfrac{2\cdot n+2+n\cdot(n+1)}{2} = \tfrac{(n+1)\cdot(n+2)}{2}\,.$$

II.15 Všeobecná rekurzia a . . . ?

Funkcie definované primitívnou rekurziou spravidla verifikujeme matematickou indukciou.
Funkcie definované všeobecnou rekurziou spravidla verifikujeme . . . ?

III. prednáška
Priraďujúce diskriminácie
Chvostová rekurzia

9. marca 2015

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

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

Všeobecná rekurzia (course-of-values)

argument rekurzívneho volania je ľubovoľné číslo ostro menšie ako pôvodný argument

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

rekurzívnych volaní môže byť viac

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

III.2 Opakovanie: Známe diskriminácie

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

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

Vzájomne výlučné, pokrývajúce všetky prípady

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

Obe doterajšie diskriminácie testujú známe hodnoty.

6. Priraďujúce diskriminácie

6.1. Priradenie

III.3 Priradenie

CL umožňuje *priradiť* hodnotu termu *novej* premennej

$$\cdots \leftarrow \cdots \wedge \tau = x$$

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

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

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

Príklady:

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

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

Premenné v *deklaratívnych* jazykoch pomenúvajú *hodnoty*

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

6.2. Monadická diskriminácia

III.4 Monadická diskriminácia — porovnanie so vzorom

Monadická diskriminácia kombinuje test a priradenie

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

x je *nová* premenná, τ je ľubovoľný term

Podmienky sú vzájomne výlučné a pokrývajú všetky prípady:

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

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

Príklad použitia:

$$n! = 1 \quad \leftarrow n = 0 \qquad x^y = 1 \quad \leftarrow y = 0$$
$$n! = n{\cdot}m! \leftarrow n = m+1 \qquad x^y = x{\cdot}x^z \leftarrow y = z+1$$

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 do ľavej strane rovnosti:

$$0! = 1 \qquad x^0 = 1$$
$$(m+1)! = (m+1){\cdot}m! \qquad x^{z+1} = x{\cdot}x^z$$

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

Zovšeobecnená monadická diskriminácia — skratka kaskády:

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

x je *nová* premenná, k je *konštanta*, τ je ľubovoľný term

Príklad bez dosadenia a s dosadením:

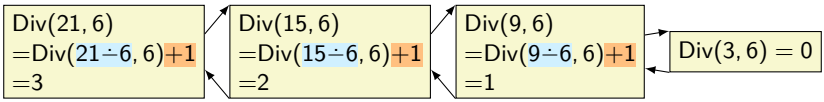
$$\begin{array}{lll}
\mathrm{fib}_n = 0 & \leftarrow n = 0 & \mathrm{fib}_0 = 0 \\
\mathrm{fib}_n = 1 & \leftarrow n = 1 & \mathrm{fib}_1 = 1 \\
\mathrm{fib}_n = \mathrm{fib}_{m+1} + \mathrm{fib}_m & \leftarrow n = m + 2 & \mathrm{fib}_{n+2} = \mathrm{fib}_{n+1} + \mathrm{fib}_n
\end{array}$$

7. Chvostová rekurzia

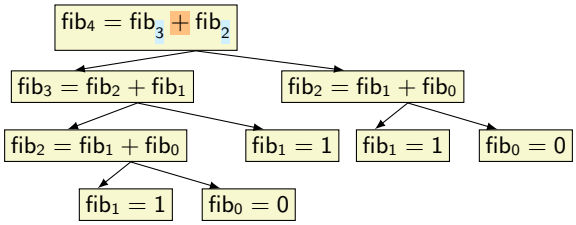
III.7 Rekurzia

Rekurzia je *dvojfázový* proces:

1. príprava rekurzívneho volania
2. použitie jeho výsledku



Výpočet môže byť stromový:



III.8 Rekurzia iteráciou

Jednoduché rekurzívne definície sa dajú počítať imperatívnym cyklom

Príklad 18.

$$n! = 1 \qquad \leftarrow n \leq 0$$
$$n! = n \cdot (n \dot{-} 1)! \leftarrow n > 0$$

```
def Fact(x):
    f = 1
    # f == 0!
    for n in range(1,x+1):
        # f == (n ∸ 1)!
        f = n * f
        # f == n!
    return f
```

Cyklus opakuje — *iteruje* výpočet vo svojom tele
Cyklus = *iterácia*
Namiesto výsledku rekurzívneho volania používa pomocnú premennú

Definícia 19. Funkcia F je definovaná *chvostovou* rekurziou (*tail* recursion), ak každá jej rekurzívna klauzula má tvar:
$F(x_1, \ldots, x_n) = F(\tau_1, \ldots, \tau_n) \leftarrow \varphi$,
kde termy $\tau_1, \ldots, \tau_n$ neobsahujú rekurzívne aplikácie F.

Chvostová rekurzia – obzvlášť jednoduchý typ rekuzie:

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

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

Príklad 20.

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

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

Kompilátory to vedia, naučme sa to aj my!

7.1. Rekurzia pre cyklus `while`

III.10 Rekurzia cyklom `while`

$$\begin{aligned} n! &= 1 && \leftarrow n = 0 \\ n! &= n{\cdot}((n \dot{-} 1)!) && \leftarrow n \neq 0 \end{aligned}$$

```
def Fact(n):
    f = 1
    while n != 0:
        f = n * f
        n = n - 1
    return f
```

III.11 Chvostová rekurzia pre cyklus `while` — rozdelenie

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

      ```
      f = 1
      ```

   b) cyklus a vrátenie výsledku

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

Aký je *matematický* význam tejto časti programu?

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

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

2. Naprogramujeme funkciu While_fact tak, aby počítala to, čo cyklus:
   - Dva argumenty — hodnoty pythonovských premenných n a f
   - Aká bude koncová hodnota f, keď hodnota $n = 0$?
     Rovnaká ako začiatočná hodnota f
   - Aká bude koncová hodnota f, keď hodnota $n \neq 0$?
     a) Vykoná sa telo cyklu $\Rightarrow$ nové hodnoty premenných n a f
     b) Cyklus sa zopakuje s novými hodnotami

     While_fact(nová hodnota n, nová hodnota f)

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

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

Invariant:

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

III.13 Chvost. rek. pre cyklus `while` — inicializácia

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

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

III.14 Chvost. rek. pre cyklus `while` — rekapitulácia

```
def Fact(n):
    f = 1
    while n != 0:
        f = f * n
        n = n - 1
    return f
```

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

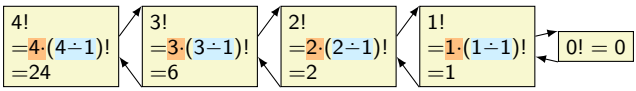
$$\text{Fact}(n) = \text{While\_fact}(n, 1)$$

Stručnejšie a elegantnejšie s monadickou diskrimináciou:

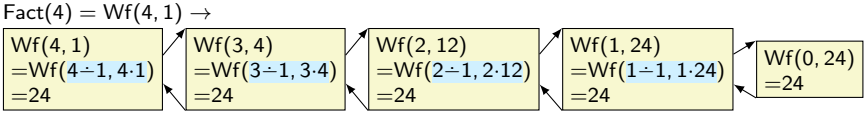
$$\text{While\_fact}(n + 1, f) = \text{While\_fact}(n, f \cdot (n + 1))$$
$$\text{While\_fact}(0, f) = f$$

III.15 *Chvostová* rekurzia pre cyklus `while` — výpočet

Priebeh výpočtu obyčajnej definície $n!$:



Priebeh výpočtu Fact pomocou While_fact (skrátené na Wf):



While_fact: *chvostová* rekurzia

- Žiadne výpočty po návrate z rekurzívneho volania
- Nepotrebuje zásobník

III.16 Chvostová rekurzia — príklad z minulého cvičenia

Chvostovú rekurziu ste použili 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?

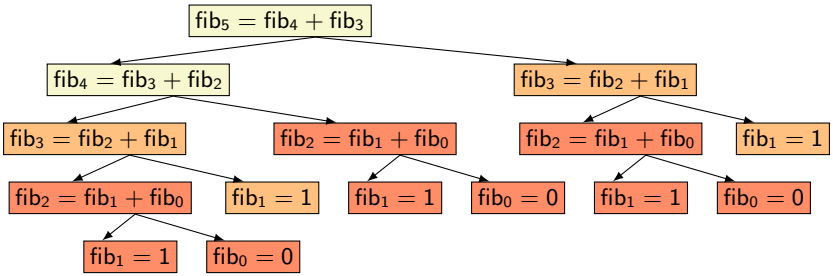
7.2. Efektívny výpočet Fibonacciho postupnosti

III.17 Fibonacciho postupnosť *ne*efektívne

Na pripomenutie: n-tý prvok Fibonacciho postupnosti

$$\mathrm{fib}_0 = 0 \qquad \mathrm{fib}_1 = 1 \qquad \mathrm{fib}_{n+2} = \mathrm{fib}_{n+1} + \mathrm{fib}_n$$

Výpočet obyčajnou rekurziou je veľmi neefektívny:

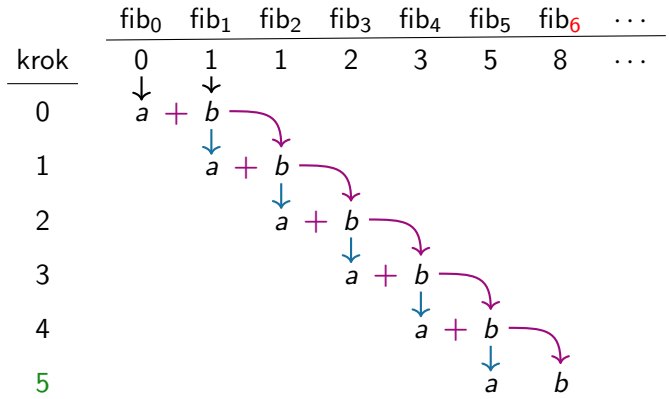


Zhruba fib_{n+2} rekurzívnych volaní
fib_n rastie ako ϕ^n, $\phi = (\sqrt{5} - 1)/2 \doteq 1.618$ — exponenciálne

III.18 Fibonacciho postupnosť efektívne

Efektívny výpočet fib_n $(n > 0)$ – zdola nahor:

- pamätáme si dva za sebou nasledujúce prvky postupnosti (na začiatku prvé dva)
- počítame nasledujúci prvok $\Longrightarrow$ stačí $n \dot{-} 1$ krokov



III.19 Fibonacciho postupnosť efektívne imperatívne

```
def Fib(n):
    if n = 0:
        return 0
    else:
        n = n - 1
        a = 0 # fib_0
        b = 1 # fib_1
        while n != 0:
            # a = fib_i ∧ b = fib_{i+1}
            a1 = b
            b1 = a + b
            a = a1
            b = b1
            # a = fib_{i+1} ∧ b = fib_{i+2}
            n = n - 1
        return b
```

- Chvostová rekurzia pre cyklus `while`:

$$\begin{aligned}\text{While\_fib}(n,a,b) &= \text{While\_fib}(n \dot{-} 1, a_1, b_1) \\ &\quad \leftarrow n \neq 0 \wedge b = a_1 \wedge b + a = b_1 \\ \text{While\_fib}(n,a,b) &= b \quad \leftarrow n = 0\end{aligned}$$

  Stručnejšie:

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

  Invariant:

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

- Úvodné testy, inicializácia a spustenie cyklu:

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

7.3. Rekurzia pre cyklus `for`

III.21 *Spätná* chvostová rekurzia pre cyklus `for`

Výpočet faktoriálu cyklom `for`:

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

Deklaratívne:

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

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

- Inicializácia premenných a štart cyklu:

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

Spätná rekurzia, napríklad

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

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

IV. prednáška
Binárna číselná sústava
Párovanie

16. marca 2015

7.4. Opakovanie

IV.1 Opakovanie: Priraďujúce diskriminácie

Priradenie

Premenná ako skratka za hodnotu termu

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

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

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

IV.2 Opakovanie: Chvostová rekurzia

Chvostová rekurzia

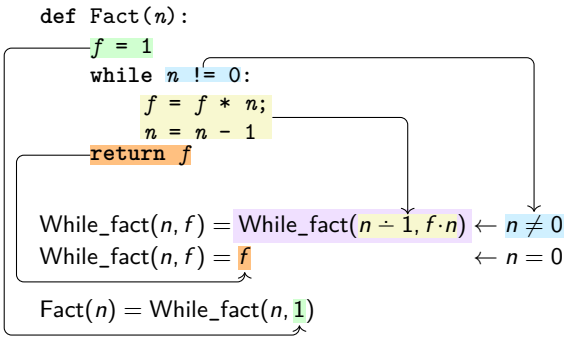
$$\begin{aligned} F(x,y,z) &= \cdots && \leftarrow \cdots \\ F(x,y,z) &= \emptyset F(\,\ldots\,,\,\ldots\,,\,\ldots\,)\emptyset && \leftarrow \cdots \end{aligned}$$

- 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

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

Cyklus `while` ∼ chvostová rekurzia riadená podmienkou cyklu

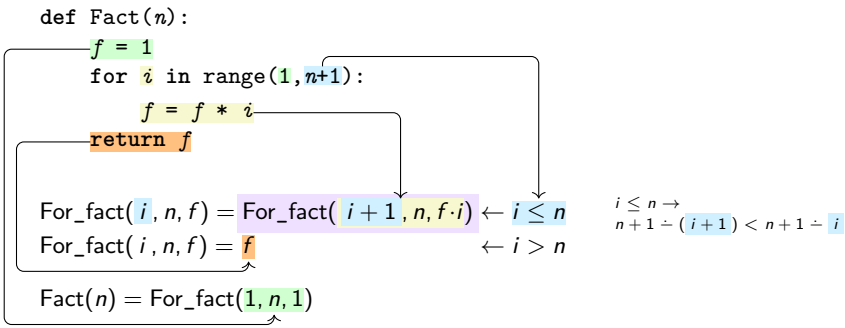
- premenné v cykle ∼ argumenty funkcie
- inicializácia ∼ volanie simulačnej funkcie hlavnou funkciou



IV.4 Opakovanie: Chvostová rekurzia a `for`

Cyklus `for` ∼ spätná chvostová rekurzia

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



8. Binárna číselná sústava

IV.5 Binárna číselná sústava

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

$$13 = \mathbf{1}\cdot 2^3 + \mathbf{1}\cdot 2^2 + \mathbf{0}\cdot 2^1 + \mathbf{1}\cdot 2^0 = (1101)_\mathrm{b}$$

- Vo všeobecnosti

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

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

$$\begin{aligned} (d_k d_{k-1} \ldots d_1 d_0)_\mathrm{b} &= \\ = (0 d_k d_{k-1} \ldots d_1 d_0)_\mathrm{b} &= \\ = (00 d_k d_{k-1} \ldots d_1 d_0)_\mathrm{b} &= \\ = \cdots & \end{aligned}$$

- Algebraické vyjadrenie binárneho zápisu:

$$13 = \mathbf{1}\cdot 2^3 + \mathbf{1}\cdot 2^2 + \mathbf{0}\cdot 2^1 + \mathbf{1}\cdot 2^0$$

  môžeme previesť do Hornerovej schémy:

$$13 = (((0\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{0})\cdot 2 + \mathbf{1}$$

- Všeobecne:

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

8.1. Binárne konštruktory

IV.7 Binárne konštruktory

- Zadefinujme

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

- Potom môžeme zapísať

$$\begin{aligned} 13 &= 0\mathbf{1101} \\ &= (((0\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{0})\cdot 2 + \mathbf{1} \\ &= S_1(((0\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{0}) \\ &= S_1\,S_0((0\cdot 2 + \mathbf{1})\cdot 2 + \mathbf{1}) \\ &= S_1\,S_0\,S_1(0\cdot 2 + \mathbf{1}) \\ &= S_1\,S_0\,S_1\,S_1(0) \end{aligned}$$

$$\begin{aligned} \textstyle\sum_{1\le i\le k} d_i\cdot 2^i &= 0 d_k \ldots d_2 d_1 \\ &= ((\cdots(0\cdot 2 + d_k)\cdots)\cdot 2 + d_2)\cdot 2 + d_1 \\ &= S_{d_1}(\cdots(0\cdot 2 + d_k)\cdots)\cdot 2 + d_2) \\ &= S_{d_1}\,S_{d_2}(\cdots(0\cdot 2 + d_k)\cdots) \\ &= S_{d_1}\,S_{d_2}\cdots S_{d_{k-1}}(0\cdot 2 + d_k) \\ &= S_{d_0}\,S_{d_1}\cdots S_{d_k}(0) \end{aligned}$$

- Takže:

$$S_0(n) = n\mathbf{0} \qquad S_1(n) = n\mathbf{1}$$

- Funkcie S_0 a S_1 nazývame *binárne konštruktory*

8.2. Binárna diskriminácia

IV.8 Testovanie binárneho zápisu

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

1. Ako potom určíme najmenej významnú číslicu čísla n?
   - $\text{Least\_signif\_bit}(n) = d_0 = n \bmod 2$
2. Ako určíme číslo, ktoré vznikne z n odobratím najmenej významnej číslice?
   - $\text{More\_signif\_bits}(n) = 0d_k \dots d_1 = \sum_{i=1}^{k} d_i \cdot 2^i = n \div 2$

Naprogramujme zmenu najmenej významnej binárnej číslice:

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

V CL existuje názornejší spôsob...

IV.9 Binárna diskriminácia

CL pozná *binárnu diskrimináciu*:

- Každé číslo sa dá rozdeliť na najmenej významnú binárnu číslicu a významnejší zvyšok m

$$\dots \leftarrow \tau = m\mathbf{0} \quad \dots \leftarrow \tau = \mathsf{S}_0(m) \quad \dots \leftarrow \tau \div 2 = m \wedge \tau \bmod 2 = 0$$
$$\dots \leftarrow \tau = m\mathbf{1} \quad \dots \leftarrow \tau = \mathsf{S}_1(m) \quad \dots \leftarrow \tau \div 2 = m \wedge \tau \bmod 2 = 1$$

τ je term so známou hodnotou, m je nová premenná

Tieto prípady pokrývajú všetky možnosti

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

a sú vzájomne výlučné:

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

IV.10 Použitie binárnej diskriminácie

v tele klauzuly	v argumente funkcie
$\text{Flip\_lsb}(n) = m\mathbf{1} \leftarrow n = m\mathbf{0}$	$\text{Flip\_lsb}(m\mathbf{0}) = m\mathbf{1}$
$\text{Flip\_lsb}(n) = m\mathbf{0} \leftarrow n = m\mathbf{1}$	$\text{Flip\_lsb}(m\mathbf{1}) = m\mathbf{0}$

Počítajme:

$$\text{Flip\_lsb}(5) = \text{Flip\_lsb}(2\mathbf{1}) = 2\mathbf{0} = 4$$
$$\text{Flip\_lsb}(6) = \text{Flip\_lsb}(3\mathbf{0}) = 3\mathbf{1} = 7$$

8.3. Binárna rekurzia

IV.11 Počet číslic binárneho zápisu

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

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

Počítajme:

$$\begin{aligned}\text{Len}(5) = \text{Len}(2\mathbf{1}) &= 1 + \text{Len}(2) = 1 + \text{Len}(1\mathbf{0}) = \\ &= 1 + 1 + \text{Len}(1) = 1 + 1 + \text{Len}(0\mathbf{1}) = \\ &= 1 + 1 + 1 + \text{Len}(0) = 1 + 1 + 1 + \text{Len}(0\mathbf{0}) = \\ &= 1 + 1 + 1 + 1 + \text{Len}(0) = 1 + 1 + 1 + 1 + \text{Len}(0\mathbf{0}) = \cdots\end{aligned}$$

S nulou opatrne a s nepresnými zadaniami tiež!

IV.12 Binárna rekurzia

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

$$\begin{aligned}F(n,\ldots) &= \cdots && \leftarrow n = m\mathbf{0} \wedge m = 0 \\ F(n,\ldots) &= \cdots F(m,\ldots) \cdots && \leftarrow n = m\mathbf{0} \wedge m > 0 \\ F(n,\ldots) &= \cdots F(m,\ldots) \cdots && \leftarrow n = m\mathbf{1}\end{aligned}$$

alebo s dosadením:

$$
\begin{aligned}
F(m\mathbf{0},\ldots) &= \cdots && \leftarrow m = 0\\
F(m\mathbf{0},\ldots) &= \cdots F(m,\ldots)\cdots && \leftarrow m > 0\\
F(m\mathbf{1},\ldots) &= \cdots F(m,\ldots)\cdots
\end{aligned}
$$

Platí

$$m > 0 \to m < m\mathbf{0} = 2{\cdot}m + 0, \qquad m < m\mathbf{1} = 2{\cdot}m + 1,$$

$\Longrightarrow$ Binárna rekurzia je špeciálny prípad všeobecnej rekurzie

IV.13 Binárna rekurzia — príklad a dĺžka rekurzie

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

$$
\begin{aligned}
\mathrm{Len}(m\mathbf{0}) &= 0 && \leftarrow m = 0\\
\mathrm{Len}(m\mathbf{0}) &= 1 + \mathrm{Len}(m) && \leftarrow m > 0\\
\mathrm{Len}(m\mathbf{1}) &= 1 + \mathrm{Len}(m)
\end{aligned}
$$

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

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

8.4. Aritmetické operácie na binárnom zápise čísel

IV.14 Efektívna aritmetika s veľkými číslami

Neefektívna definícia aritmetickej operácie:

počet krokov = *hodnota* niektorého argumentu

Napríklad násobenie primitívnou rekurziou

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

Efektívna definícia aritmetickej operácie:

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

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

IV.15 Nasledovník na binárnom zápise

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

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

Príklady:

$$\begin{array}{r} 0 \\ +\ 01 \\ \hline 01 \end{array} \qquad \begin{array}{r} 0100 \quad = 4 \\ +\ 01 \\ \hline 0101 \quad = 5 \end{array} \qquad \begin{array}{r} 010\,1 \quad = 5 \\ +\ 0\,1 \\ \hline 010 \\ +\ 01\,0 \\ \hline 011\,0 \quad = 6 \end{array}$$

Zovšeobecnenie:

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

IV.16 Nasledovník na binárnom zápise

Algebraické odvodenie

- predpokladáme, že Succ(n) počíta $n + 1$
- hľadáme vyjadrenie zamýšľaného výsledku *iba* použitím:
  - čísla 0
  - binárnych konštruktorov $\mathrm{S}_0(n) = n0$ a $\mathrm{S}_1(n) = n1$
  - rekurzie so zvyšnými číslicami binárneho zápisu

$$\begin{aligned} \mathrm{Succ}(\,x0\,) &= x0 + 1 = (2{\cdot}x + 0) + 1 = 2{\cdot}x + 1 = x1 \\ \mathrm{Succ}(\,x1\,) &= x1 + 1 = (2{\cdot}x + 1) + 1 = 2{\cdot}(x + 1) = (x + 1)0 \\ &= (\mathrm{Succ}(\,x\,))0 \end{aligned}$$

Klauzálna definícia použitím binárnej rekurzie:

$$\begin{array}{ll|l} \mathrm{Succ}(x) = v1 & \leftarrow x = v0 & \mathrm{Succ}(v0) = v1 \\ \mathrm{Succ}(x) = (\mathrm{Succ}(v))0 & \leftarrow x = v1 & \mathrm{Succ}(v1) = (\mathrm{Succ}(v))0 \end{array}$$

IV.17 Sčítanie na binárnom zápise

Príklady sčítania:

$$\begin{array}{r} 0 \\ +\ 010 \\ \hline 010 \end{array} \qquad \begin{array}{r} 011 \\ +\ 0110 \\ \hline 01001 \end{array} \qquad \begin{array}{r} 011 \\ +\ 011 \\ \hline 0110 \end{array}$$

Zovšeobecnenie:

$$x > 0 \rightarrow \quad \begin{array}{r} 0\mathbf{0} \\ +\ y \\ \hline y \end{array} \qquad \begin{array}{r} x\mathbf{0} \\ +\ 0\mathbf{0} \\ \hline x\mathbf{0} \end{array} \qquad \begin{array}{r} x\mathbf{1} \\ +\ 0\mathbf{0} \\ \hline x\mathbf{1} \end{array}$$

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

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

IV.18 Sčítanie na binárnom zápise

Klauzálna definícia binárnou rekurziou s dvojitou binárnou diskrimináciou

Odvodíme podľa príkladu z predchádzajúcej snímky alebo algebraicky použitím:

- čísla 0, binárnych konštruktorov $S_0(n) = n\mathbf{0}$ a $S_1(n) = n\mathbf{1}$
- nasledovníka
- rekurzie so zvyšnými číslicami binárneho zápisu

$$\begin{aligned}
\mathrm{Add}(x\mathbf{0}, y\mathbf{0}) &= \ 0 + y\mathbf{0} = y\mathbf{0} && \leftarrow x = 0\\
\mathrm{Add}(x\mathbf{0}, y\mathbf{1}) &= \ 0 + y\mathbf{1} = y\mathbf{1} && \leftarrow x = 0\\
\mathrm{Add}(x\mathbf{0}, y\mathbf{0}) &= x\mathbf{0} + y\mathbf{0} = x\mathbf{0} && \leftarrow x > 0 \wedge y = 0\\
\mathrm{Add}(x\mathbf{0}, y\mathbf{0}) &= x\mathbf{0} + y\mathbf{0} = (2{\cdot}x + 0) + (2{\cdot}y + 0) = 2{\cdot}(x + y) =\\
&= (\,x + y\,)\mathbf{0} = (\,\mathrm{Add}(x, y)\,)\mathbf{0} && \leftarrow x > 0 \wedge y > 0\\
\mathrm{Add}(x\mathbf{0}, y\mathbf{1}) &= x\mathbf{1} + y\mathbf{0} = (2{\cdot}x + 0) + (2{\cdot}y + 1) = 2{\cdot}(x + y) + 1 =\\
&= (\,x + y\,)\mathbf{1} = (\,\mathrm{Add}(x, y)\,)\mathbf{1} && \leftarrow x > 0\\
\mathrm{Add}(x\mathbf{1}, y\mathbf{0}) &= x\mathbf{1} + \ 0 = \cdots && \leftarrow y = 0\\
\mathrm{Add}(x\mathbf{1}, y\mathbf{0}) &= x\mathbf{1} + y\mathbf{0} = \cdots \mathrm{Add}(x, y) \cdots && \leftarrow y > 0\\
\mathrm{Add}(x\mathbf{1}, y\mathbf{1}) &= x\mathbf{1} + y\mathbf{1} = \cdots \mathrm{Add}(x, y) \cdots
\end{aligned}$$

IV.19 Násobenie na binárnom zápise

Algebraické odvodenie:

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

9. Párovanie

9.1. Kódovanie dátových štruktúr

IV.20 Dátové štruktúry

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

- Ako *zakódujeme* usporiadané dvojice v CL?

9.2. Párovacie funkcie

IV.21 Kódovanie usporiadaných dvojíc — párov

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

- Každému páru (x, y) *jednoznačne* priradíme číslo $P(x, y)$
  - Potom môžeme z čísla 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 `None`

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

IV.22 Párovacie funkcie

Definícia 21. *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:

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

- 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	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

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

  - 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

IV.24 Párovacia funkcia CL

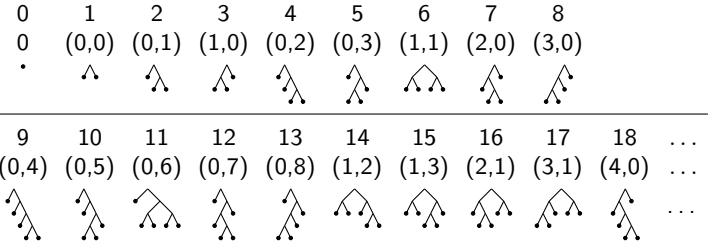
Párovacia funkcia zabudovaná v CL:

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

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

IV.25 Párovacia funkcia CL

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



- Fragment tabuľky párovacej funkcie (,):

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

IV.26 Párovacia funkcia CL

- Syntax operátora (,) v CL:
  – najnižšia priorita: $1 + 2, 3 + 4 \equiv ((1 + 2), (3 + 4))$
  – implicitne sa zátvorkuje doprava: $1, 2, 3, 4 \equiv 1, (2, (3, 4))$
- Výpočet založený na Catalanových číslach, náročný
- Párovacia funkcia sa počíta „lenivo“

  - iba keď je naozaj požadovaná číselná hodnota
  - inak si CL pamätá pôvodnú dvojicu čísel

- Projekčné funkcie:
  - $\mathsf{H}(x)$ 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:

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

IV.27 Usporiadané n-tice

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

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

  Všimnite si implicitné zátvorkovanie doprava

- Usporiadané n-tice:

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

9.2.2. Programovanie s párovacou funkciou

IV.28 Programovanie s párovacou funkciou

- Pár vytvoríme termom „σ, τ“
- Párová diskriminácia:

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

  Druhý prípad priradí zložky páru do nových premenných u a v

- Párová diskriminácia je skratkou za:

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

- Projekcie H a T *nebudeme* používať

IV.29 Programovanie s párovacou funkciou — príklad

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

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

- Prípad v druhej klauzule nemôže nastať

  Môžeme ju vynechať, CL si výsledok 0 doplní *default*om

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

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

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

IV.30 Programovanie s párovacou funkciou — výpočet

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

- Výpočet:

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

- CL nepočíta číselnú hodnotu párovacej funkcie, pokiaľ to nie je nevyhnutné, v pamäti vytvorí pythonovskú dvojicu

9.2.3. Tupling

IV.31 Tupling

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

  Technika sa nazýva *tupling*

- Napríklad celočíselné delenie a zvyšok:

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

  spojíme do jednej funkcie

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

- Spojená funkcia:

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

V. prednáška
Zoznamy.
Chvostová rekurzia a tupling na zoznamoch

23. marca 2015

10. Zoznamy

V.1 Opakovanie

Párovacia funkcia:

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

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

  Druhý prípad priradí zložky páru do nových premenných u a v
- Párovacia funkcia sa počíta „lenivo“
  - iba keď je naozaj požadovaná číselná hodnota
  - inak si CL pamätá pôvodnú dvojicu čísel

10.1. Kódovanie konečných postupností

V.2 Kódovanie konečných postupností

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

V.3 Kódovanie konečných postupností

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

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

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

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

- Problematické: vopred známe maximum, koncové nuly

V.4 Kódovanie konečných postupností

- Skúsme využiť vnorené párovanie:

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

- Dekódujme teraz postupnosť zakódovanú číslom x

  Dĺžku nepoznáme vopred

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

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

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

V.5 Kódovanie konečných postupností

- Nie každé číslo kóduje dvojicu:

  0

- Nie každé číslo kódujúce dvojicu kóduje trojicu:

  $x_1, 0$

  $\vdots$

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

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

V.6 Kódovanie konečných postupností

- 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

$$xs = 8, 2, 0, 4, 6, 0 = 8, (2, (0, (4, (6, 0))))$$
$$ys = 1, 3, 5, 7, 0 \quad = 1, (3, (5, (7, 0)))$$

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

  Premenenné označujúce zoznamy majú príponu -*s*

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

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

10.2. Programovanie so zoznamami

10.2.1. Zreťazenie

V.7 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?

V.8 Programovanie so zoznamami: Zreťazenie

Ako hľadať riešenie?

1. Načrtneme si príklad
   - ▶ $\text{Conc}\big((1, 2, 3, 0), (4, 5, 6, 0)\big) = 1, 2, 3, 4, 5, 6, 0$
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
   - $\mathrm{Conc}(\underbrace{0}_{xs}, \underbrace{(4,5,6,0)}_{ys}) = \underbrace{4,5,6,0}_{ys}$

4. *Výsledok pre zvyšok zs* zoznamu *xs*
   - $\mathrm{Conc}(\underbrace{(2,3,0)}_{zs}, \underbrace{(4,5,6,0)}_{ys}) = 2,3,4,5,6,0$

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

V.9 Programovanie so zoznamami: Zreťazenie

- 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:

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

V.10 Zabudované zreťazenie

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

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

- Vyššia priorita ako párovanie:

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

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

Zreťazovať môžeme *iba zoznamy*!

10.2.2. Porovnanie s Pythonom

V.11 Zoznamy v CL a v Pythone

- Ako by sme naprogramovali zreťazenie

$$\text{Conc}(xs, ys) = ys \qquad \leftarrow xs = 0$$
$$\text{Conc}(xs, ys) = x, \text{Conc}(zs, ys) \leftarrow xs = x, zs$$

v Pythone?

- Napríklad takto:

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

10.2.3. Obrátenie

V.12 Programovanie so zoznamami: Obrátenie

Naprogramujme funkciu 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
   - ▶ $\text{Rev}(\underbrace{0}_{xs}) = 0$
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*
   - ▶ $\text{Rev}(\underbrace{3}_{x}, \underbrace{(1, 0, 2, 4, 0)}_{zs}) = \underbrace{(4, 2, 0, 1, 0)}_{\text{Rev}(zs)} \,??\, \underbrace{3}_{x} = 4, 2, 0, 1, 3, 0$

10.3. Chvostová rekurzia na zoznamoch

10.3.1. Súčet prvkov zoznamu

V.13 Cykly na zoznamoch

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

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

- V deklaratívnom programovaní úlohu cyklov plní chvostová rekurzia

V.14 Chvostová rekurzia na zoznamoch

- Chvostová[↑7] rekurzia simulujúca `while` cyklus[↑7.1] pre súčet prvkov zoznamu:

$$\begin{aligned}\text{While\_suml}(xs, s) &= s && \leftarrow xs = 0\\ \text{While\_suml}(xs, s) &= \text{While\_suml}(zs, s + x) && \leftarrow xs = x, zs\end{aligned}$$

- Inicializácia:

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

- S dosadením do argumentov:

$$\begin{aligned}\text{While\_suml}(\quad 0, s) &= s\\ \text{While\_suml}((x, xs), s) &= \text{While\_suml}(xs, s + x)\end{aligned}$$

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

10.3.2. Obrátenie zoznamu efektívne

V.15 Neefektivita obrátenia zoznamu zreťazovaním

- Pred chvíľou[↑10.2.3] sme obrátenie zoznamu naprogramovali:

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

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

$$\begin{array}{rrr}
\mathsf{Rev}(1,2,3,4,0) = \mathsf{Rev}(2,3,4,0) \oplus (1,0) = (4,3,2,0) \oplus (1,0) & 3 \\
\mathsf{Rev}(2,3,4,0) = \mathsf{Rev}(3,4,0) \oplus (2,0) = (4,3,0) \oplus (2,0) & 2 \\
\mathsf{Rev}(3,4,0) = \mathsf{Rev}(4,0) \oplus (3,0) = (4,0) \oplus (3,0) & 1 \\
\mathsf{Rev}(4,0) = \mathsf{Rev}(0) \oplus (4,0) = (0) \oplus (4,0) & 0 \\
\mathsf{Rev}(0) = 0 & \\
\hline
3{+}2{+}1{+}0 = 6 &
\end{array}$$

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

V.16 Efektívne obrátenie zoznamu

- Zoznam xs možno obrátiť na $\mathsf{L}(xs)$ krokov:

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

- Aký program realizuje načrtnutý postup?

V.17 Efektívne obrátenie zoznamu v CL

- Efektívne obrátenie zoznamu `while` cyklom:

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

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

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

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

10.4. Tupling zoznamových operácií

10.4.1. Split = Take,Drop

V.18 Príklad — L, Take, Drop

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

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

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

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

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

- 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 $\Longrightarrow$ dva prechody zoznamom
- Chceme spojenú jednoprechodovú funkciu — *tupling*[↑9.2.3]

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

- Príklad s náčrtom rekurzie:

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

10.4.2. Zip, Unzip

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

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

- Unzip — rozdelenie zoznamu dvojíc na dva zoznamy

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

- V podstate tuplingová úloha: spájame funkcie

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

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

- Príklad s náčrtom rekurzie:

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

VI. prednáška
Triedenie zoznamov
Zoznamová reprezentácia množín

30. marca 2015

11. Predikáty

VI.1 Predikáty

- *Predikát* je vlastnosť objektu alebo n-tice objektov
  - číslo x je nepárne

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

  - číslo d delí číslo x

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

  - číslo x kóduje trojicu čísel

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

  - xs je prázdny zoznam

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

  - zoznam xs je palindróm

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

  ...

VI.2 Prvok zoznamu

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

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

- Alternatívne: Práve vtedy, keď sa *a* vyskytuje ako $(i + 1)$-ý prvok zoznamu *xs* pre nejaké *i* menšie ako dĺžka zoznamu (teda *i* je platným indexom do *xs*):

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

VI.3 Charakteristické funkcie

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

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

Príklad 22.
$\text{Palindrome}_*(xs) = 1 \leftarrow x = \text{Rev}(xs)$
$\text{Palindrome}_*(xs) = 0 \leftarrow x \neq \text{Rev}(xs)$

Príklad 23.
$\text{Member}_*(a, \quad 0) = 0$
$\text{Member}_*(a, (x, xs)) = 1 \quad \leftarrow a = x$
$\text{Member}_*(a, (x, xs)) = \text{Member}_*(a, xs) \leftarrow a \neq x$

VI.4 Definovanie predikátov v CL

V CL môžeme predikáty definovať priamo

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

Príklad 24.
$\text{Palindrome}(x) \leftarrow x = \text{Rev}(x)$
~~$\neg\text{Palindrome}(x) \leftarrow x \neq \text{Rev}(x)$~~

VI.5 Definovanie predikátov v CL — diskriminácia a rekurzia

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

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

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

Príklad 25.
$\neg\text{Member}(a, 0)$
$\text{Member}(a, (x, xs)) \leftarrow x = a$
$\text{Member}(a, (x, xs)) \leftarrow x \neq a \wedge \text{Member}(a, xs)$
$\neg\text{Member}(a, (x, xs)) \leftarrow x \neq a \wedge \neg\text{Member}(y, z)$

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

12. Triedenie zoznamov

12.1. Špecifikácia triedenia

VI.6 Špecifikácia triedenia

Definícia 26. Funkcia Sort(xs) = *ys utriedi* zoznam xs, ak súčasne platí:

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

Formalizácia špecifikácie:

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

VI.7 Permutácia zoznamu

Ako zistíme, či je zoznam xs permutáciou zoznamu ys?

- Každý prvok sa v xs nachádza rovnaký početkrát ako v ys
- Potrebujeme počet výskytov prvku a v zozname xs
  - Funkcia $\text{Count}(a, xs) \equiv \#_a(xs)$
- Potom máme

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

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

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

alternatívne

$$xs \sim ys \leftrightarrow \begin{pmatrix} \forall a(a \varepsilon xs \rightarrow \#_a(xs) = \#_a(ys)) \\ \wedge \forall a(a \varepsilon ys \rightarrow \#_a(xs) = \#_a(ys)) \end{pmatrix}$$

alternatívne

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

VI.8 Permutácia zoznamu — rekurzívny predikát

Ako vypočítame $xs \sim ys$ rekurziou?

1. Rekurzívny výpočet $\#_a(xs)$

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

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

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

Naprogramujeme zoznamovou rekurziou na zs

$\mathsf{Eq\_counts}(\quad 0, \quad xs, ys)$
$\mathsf{Eq\_counts}((z, zs), xs, ys) \leftarrow ?$

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

   $xs \sim ys \leftarrow ?$

VI.9 Usporiadaný zoznam

Ako zistíme, či je zoznam xs usporiadaný?

- Každý prvok a v zozname xs je menší alebo rovnaký ako každý nasledujúci prvok b v zozname xs
- Vyjadrenie indexovaním:

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

- Vyjadrenie zreťazením:

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

- Stačí porovnávať susediace prvky:

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

VI.10 Usporiadaný zoznam — rekurzívny predikát

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

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

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

VI.11 Špecifikácia triedenia — rekapitulácia

Funkcia Sort(xs) utriedi zoznam xs, ak spĺňa

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

pričom

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

VI.12 Algoritmy triedenia zoznamov

- Všetky algoritmy triedenia zoznamov spĺňajú rovnakú špecifikáciu
- Líšia sa 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

VI.13 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

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

Napríklad

$$\mathsf{Isort}(\underbrace{7, 2, 0, 5, 0}_{xs}) = 0, 2, 5, 7, 0$$
$$\mathsf{Isort}(\underbrace{3}_{x}, \underbrace{7, 2, 0, 5, 0}_{xs}) = \underbrace{0, 2, \underbrace{3}_{?}, 5, 7, 0}_{?(x,\mathsf{Isort}(xs))}$$

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

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

VI.14 Triedenie vsúvaním: vsunutie

- Funkcia Ins vsunie prvok do utriedeného zoznamu

$$\mathsf{Ins}(a, xs) \sim a, xs$$
$$\mathsf{Ord}(xs) \to \mathsf{Ord}\,\mathsf{Ins}(a, xs)$$

- 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:

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

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

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

VI.16 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

VI.17 Zlúčenie utriedených zoznamov efektívne

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

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

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

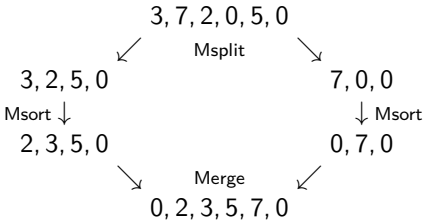
$$\begin{aligned}
&\mathsf{Merge}(xs, ys) = ? \leftarrow xs = 0\\
&\mathsf{Merge}(xs, ys) = ? \leftarrow xs = x, us \wedge ys = 0\\
&\mathsf{Merge}(xs, ys) = ? \leftarrow xs = x, us \wedge ys = y, vs \wedge x \leq y\\
&\mathsf{Merge}(xs, ys) = ? \leftarrow xs = x, us \wedge ys = y, vs \wedge x > y
\end{aligned}$$

12.4. Triedenie zlučovaním

VI.18 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!):

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

VI.19 Triedenie zlučovaním — rozdelenie

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

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

- 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

VI.20 Konečné množiny

- 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

VI.21 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

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

VI.22 Príslušnosť a extenzionalita

- Príslušnosť do množiny

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

- Pre neprázdne množiny:

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

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

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

- Platí extenzionalita

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

VI.23 Operácie na množinách

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

$$\mathrm{Set}(xs) \wedge \mathrm{Set}(ys) \to \mathrm{Set}(xs \cup ys)$$
$$\mathrm{Set}(xs) \wedge \mathrm{Set}(ys) \to a \in xs \cup ys \leftrightarrow a \in xs \vee a \in ys$$

- Princíp implementácie je rovnaký ako pri Merge

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

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

VII. prednáška
Kombinatorické funkcie na zoznamoch

13. apríla 2015

VII.1 Predikáty (opakovanie)

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

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

  - zoznam xs je palindróm

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

  - a je prvkom zoznamu xs

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

  - ys je prefixom (začiatočným úsekom) zoznamu xs

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

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

14. Kombinatorické funkcie na zoznamoch

VII.2 Kombinatorické funkcie na zoznamoch

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

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

alebo ekvivalentne stručnejšie:

$$ys \; \varepsilon \; Rs(xs) \leftrightarrow R(xs, ys)$$

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

$$Rs(0) = \cdots$$
$$Rs(x, xs) = \cdots Rs(xs) \cdots$$

14.1. Súvislé úseky

14.1.1. Sufixy

VII.3 Sufix

- Sufix — koncový úsek zoznamu

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

- 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:

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

- Analyzujme prípady pre xs, vyjadrime $\sqsupset$ rekurzívne

VII.4 Sufix — analýza prípadov

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

- $xs \sqsupset zs$ 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

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

VII.5 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 $xs \sqsupset zs$ na zadefinovanie Suffixes

$$zs \in \text{Suffixes}(xs) \leftrightarrow xs \sqsupset zs \qquad 0 \sqsupset zs \leftrightarrow zs = 0$$
$$a \in (x, xs) \leftrightarrow a = x \vee a \in xs \qquad (x, xs) \sqsupset zs \leftrightarrow zs = (x, xs) \vee xs \sqsupset zs$$

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

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

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

    teda

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

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

14.1.2. Prefixy

VII.7 Prefix

- Prefix — začiatočný úsek zoznamu

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

- 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:

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

- Rozoberme prípady pre xs

VII.8 Prefix — analýza prípadov

$ys \sqsubset xs$ 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

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

VII.9 Všetky prefixy

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

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

- Špecifikácia:

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

- Využime analýzu prípadov pre $ys \sqsubset xs$

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

- Prefixom prázdneho zoznamu je iba prázdny zoznam:

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

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

  teda

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

- Potrebujeme pomocnú funkciu ???

- Špecifikácia pomocnej funkcie pre Prefixes:

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

- Pridá x na začiatok každého zoznamu z vss

14.1.3. Segmenty

- 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:

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

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

14.2. Podpostupnosti

VII.13 Podpostupnosť

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

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

VII.14 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

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

VII.15 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:

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

- Pre skúšku správnosti: Koľko je všetkých podpostupností daného zoznamu, v ktorom sa neopakujú prvky?

1. Jedinou podpostupnosťou prázdneho zoznamu je prázdny zoznam

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

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

$$\begin{aligned} ys \,\varepsilon\, \mathrm{Subseqs}(x, xs) &\leftrightarrow ys \lhd (x, xs) \\ &\leftrightarrow \exists zs(ys = (x, zs) \wedge zs \lhd xs) \vee ys \lhd xs \\ &\leftrightarrow \exists zs(ys = (x, zs) \wedge zs \,\varepsilon\, \mathrm{Subseqs}(xs)) \vee ys \,\varepsilon\, \mathrm{Subseqs}(xs) \end{aligned}$$

Príklad:

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

14.3. Permutácie

VII.17 Permutácia

- Permutáciu zoznamu môžeme zadefinovať aj inak ako

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

- Potrebujeme pomocný predikát: zoznam xs vznikne *vsunutím* prvku a do zoznamu ys:

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

- Analýza prípadov pre permutácie:

– Permutáciou prázdneho zoznamu je iba prázdny zoznam:

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

– Permutácia zoznamu (x, xs) vznikne vsunutím prvku x do nejakej permutácie zoznamu xs:

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

VII.18 Vsunutie

- Analýza prípadov pre vsunutie:
  – ys vznikne vsunutím a do prázdneho zoznamu práve vtedy, keď ys obsahuje iba a:

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

  – ys vznikne vsunutím a do zoznamu (x, xs) práve vtedy, keď
    * ys vznikne pridaním a na začiatok (x, xs) alebo
    * ys začína prvkom x a pokračuje nejakým zoznamom zs, ktorý vznikne vsunutím 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

VII.19 Ďalšie kombinatorické úlohy

Premyslite si:

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

VIII. prednáška
Binárne stromy

20. apríla 2015

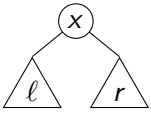
15. Binárne stromy

15.1. Kódovanie stromov párovacími konštruktormi

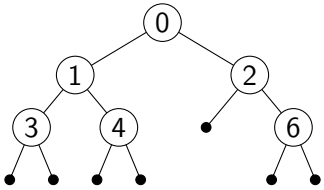
VIII.1 Binárne stromy

Definícia 27. Binárnym stromom je:

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



Príklad 28.



VIII.2 Párovacie konštruktory

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

- Použijeme všeobecnejší 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*

VIII.3 Kódovanie binárnych stromov

Binárny strom *zakódujeme* konštruktormi takto:

- Prázdny strom:

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

- Vrchol s hodnotou x a deťmi ℓ a r:

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

Binárny strom *dekódujeme* diskrimináciou:

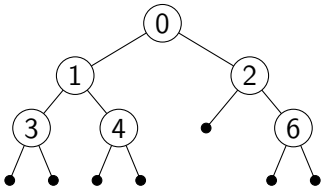
$$\cdots \leftarrow t = \mathsf{E} \qquad \cdots \leftarrow t = \bullet$$
$$\cdots \leftarrow t = \mathsf{Nd}(x, \ell, r) \qquad \cdots \leftarrow t = \tfrac{x}{\ell \mid r}$$

Po dosadení do argumentov:

$$F(\mathsf{E}) = \cdots \qquad F(\bullet) = \cdots$$
$$F(\mathsf{Nd}(x, \ell, r)) = \cdots \qquad F\left(\tfrac{x}{\ell \mid r}\right) = \cdots$$

VIII.4 Kódovanie binárnych stromov — príklad

Napríklad strom



zakódujeme

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

VIII.5 Formát binárnych stromov

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

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

- V CL takéto predikáty slúžia aj na formátovanie výsledkov query

15.2. Operácie na binárnych stromoch

VIII.6 Rekurzia na binárnych stromoch

Operácie na binárnych stromoch definujeme *rekurziou na binárnych stromoch*

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

Špeciálny prípad všeobecnej (course-of-values) rekurzie, kódy detí sú menšie čísla ako kód ich rodičovského vrcholu

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

VIII.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

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

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

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

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

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

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

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

15.3. Prechody binárnymi stromami

Vrcholy 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 $\mathrm{Bt}(t)$):

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

$$\nearrow\ ?\ \nwarrow$$

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

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

VIII.10 Prechod binárnym stromom — výpočet

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

VIII.11 Efektívne prechody binárnymi stromami

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

  Naprogramujme funkciu $\mathsf{Inordera}(t, as)$, ktorá
  - pripojí inorder. výpis stromu *pred* zoznam v akumulátore *as*

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

  - prvky pridáva na začiatok akumulátora párovaním $(,)$, po jednom, *bez* zreťazenia
- Program odvodíme z požadovanej vlastnosti $(*)$

VIII.12 Efektívne prechody binárnymi stromami

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

$$\mathsf{Inordera}\left(\begin{array}{c}0\\ \hline 1 \mid 2\end{array}, \underline{98, 99, 0}_{as}\right) = 3, 1, 4, 0, 2, 6, \underline{98, 99, 0}_{as}$$

$$\uparrow ?$$

$$\mathsf{Inordera}\left(\begin{array}{c}1\\ \hline 3 \mid 4\end{array}, \underline{77, 78, 0}_{as_1}\right) = 3, 1, 4, \underline{77, 78, 0}_{as_1}$$

$$\mathsf{Inordera}\left(\begin{array}{c}2\\ \hline \bullet \mid 6\end{array}, \underline{88, 89, 0}_{as_2}\right) = 2, 6, \underline{88, 89, 0}_{as_2}$$

VIII.13 Efektívne prechody binárnymi stromami

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

$$\mathsf{Inordera}(\bullet, as) \overset{(*)}{=} \mathsf{Inorder}(\bullet) \oplus as = 0 \oplus as = as$$

$$\mathsf{Inordera}\left(\frac{x}{\ell \mid r}, as\right) \overset{(*)}{=} \mathsf{Inorder}\left(\frac{x}{\ell \mid r}\right) \oplus as$$

$$= \big(\mathsf{Inorder}(\ell) \oplus (x, \mathsf{Inorder}(r))\big) \oplus as$$

$$= \mathsf{Inorder}(\ell) \oplus \big((x, \mathsf{Inorder}(r)) \oplus as\big)$$

$$= \mathsf{Inorder}(\ell) \oplus (x, \underline{\mathsf{Inorder}(r) \oplus as})$$

$$\overset{(*)}{=} \underline{\mathsf{Inorder}(\ell) \oplus (x, \underline{\mathsf{Inordera}(r, as)})}$$

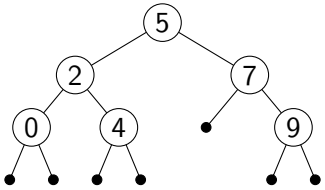
$$\overset{(*)}{=} \underline{\mathsf{Inordera}\big(\ell, (x, \mathsf{Inordera}(r, as))\big)}$$

15.4. Binárne vyhľadávacie stromy

VIII.14 Binárne vyhľadávacie stromy

Definícia 29. Binárny vyhľadávací strom je

- binárny strom;
- v *každom* vrchole s hodnotou x platí súčasne:
  - x je väčšie ako všetky hodnoty v ľavom dieťati,
  - x je menšie ako všetky hodnoty v pravom dieťati,
  - obe deti sú binárne vyhľadávacie stromy.



VIII.15 Binárny vyhľadávací strom — príklad

Vyjadrenie rekurzívnym predikátom:

$\mathrm{Bst}(\bullet)$

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

Cvičenie: rekurzívne definície predikátov

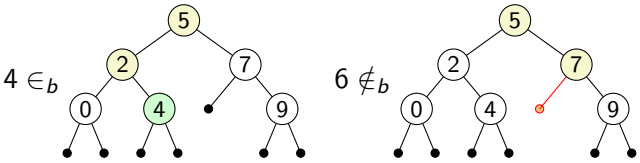
$$x \succ t \leftrightarrow \forall y (y \; \varepsilon_b \; t \rightarrow x > y)$$
$$x \prec t \leftrightarrow \forall y (y \; \varepsilon_b \; t \rightarrow x < y)$$

VIII.16 Využitie vyhľadávacej vlastnosti

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

- Využitie vyhľadávacej vlastnosti pri predikáte „byť prvkom“ $a \in_b t$ pre $t = \frac{x}{\ell \mid r}$:
  - Ak $a < x$, prehľadávame iba ℓ, lebo a nemôže byť v r
  - Ak $a > x$, prehľadávame iba r, lebo a nemôže byť v ℓ

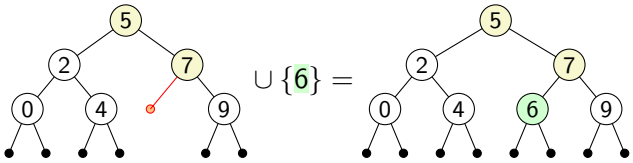
  $\Rightarrow$ Prehľadávame *iba jednu cestu* stromom



VIII.17 Vkladanie do vyhľadávacieho stromu

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 vrchol



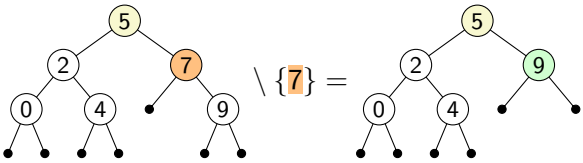
  b) Hodnota sa v strome nachádza $\Rightarrow$ strom sa nemení
- Cestou naspäť strom zrekonštruujeme

VIII.18 Zmazanie z vyhľadávacieho stromu I

Zmazanie hodnoty je komplikovanejšie ako vloženie

- Pokúsime sa nájsť zmazávanú hodnotu:
  a) Hodnotu nenájdeme $\Rightarrow$ strom sa nezmení

b) Hodnotu nájdeme a niektoré dieťa je prázdne ⇒ nahradíme vrchol druhým dieťaťom



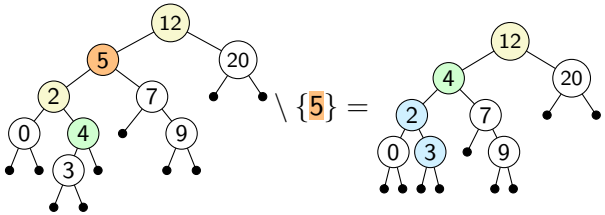
VIII.19 Zmazanie z vyhľadávacieho stromu II

- Pokúsime sa nájsť zmazávanú hodnotu:

  ...

  c) Hodnotu nájdeme a obe deti sú neprázdne:

  1. Nájdeme náhradu za zmazávanú hodnotu: napríklad maximum ľavého podstromu
  2. Odstránime náhradu z pôvodného miesta
  3. Použijeme náhradu namiesto zmazávanej hodnoty



  Kroky 1 a 2 — pomocná tuplingová funkcia (rozoberieme podrobnejšie)

- Cestou naspäť strom zrekonštruujeme

- Nájdenie a zmazanie maxima vyhľadávacieho stromu:

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

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

- Programy hľadajú maximum v najpravejšom vrchole:

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

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

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

VIII.21 Maximum vyhľadávacieho stromu — 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:

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

15.5. Úplné binárne stromy

VIII.22 Úplné binárne stromy

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

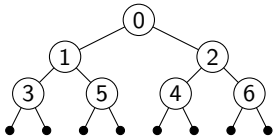
$\mathsf{d}(\bullet) = 0$

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

Definícia 31. Binárny strom nazveme *úplným*, ak

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

Príklad 32.



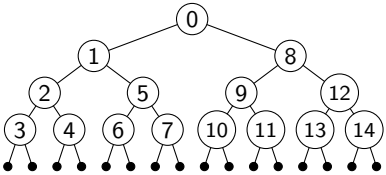
Tvrdenie 33 (Vzťah veľkosti a hĺbky úplného stromu)**.**

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

VIII.23 Číslovanie vrcholov v úplných stromoch I

- Číslovanie vrcholov úplného stromu číslami 0, 1, 2, ... v poradí pre-order:

Full_pre(4) =



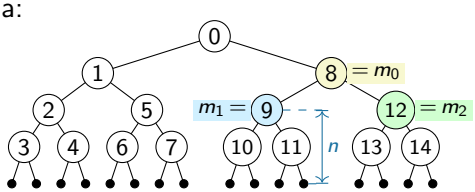
- Na nájdenie rekurzívneho riešenia musíme problém *zovšeobecniť*
- Zovšeobecnenie umožní číslovať podstromy

VIII.24 Číslovanie vrcholov v úplných stromoch II

- Zovšeobecnenie — Full_pre$_1(n, m)$:

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

- Schéma riešenia:



  Číslovanie v poradí *preorder*:

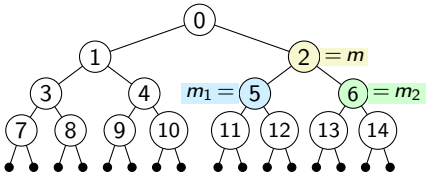
  1. očíslujeme koreň podstromu $m_0 = m$
  2. očíslujeme ľavý podstrom od $m_1 = m_0 + 1$
  3. očíslujeme pravý podstrom od $m_2 = m_1 + s$

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

VIII.25 Číslovanie úplných stromov do šírky

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

Full_br(4) =

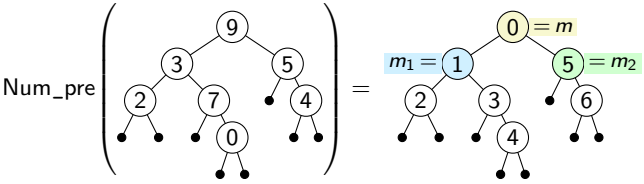


- Problém zovšeobecníme pre podstrom s koreňom s číslom m
- Aký je vzťah medzi číslom vrcholu m a číslami jeho detí m_1 a m_2?

15.6. Číslovanie vrcholov v binárnych stromoch

VIII.26 Číslovanie vrcholov v ľubovoľných stromoch

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



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

IX. prednáška
Aritmetické výrazy

27. apríla 2015

16. Aritmetické výrazy a výrokové formuly

16.1. Aritmetické výrazy

IX.1 Aritmetické výrazy

Definícia 34. *Aritmetický výraz* je

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

Nič iné nie je aritmetický výraz.

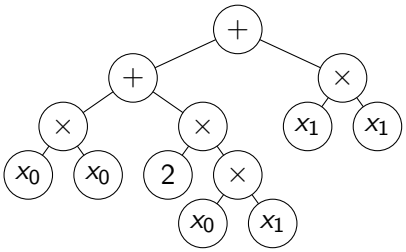
Príklad 35.

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

IX.2 Aritmetické výrazy ako stromy

- Aritmetické výrazy majú stromovú štruktúru

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



- Poznáte z programovania pod menom *aritmetický strom*

IX.3 Kódovanie aritmetických výrazov

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

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

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

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

IX.4 Kódovanie aritmetických výrazov — príklad

Príklad 36. Výraz $((x_0 \times x_0) + (2 \times (x_0 \times x_1))) + (x_1 \times x_1)$ zakódujeme ako

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

IX.5 Štrukturálna rekurzia na výrazoch

Definícia 37. Funkcia je definovaná *štrukturálnou rekurziou na výrazoch* (tiež rekurziou na štruktúru výrazu), ak je hodnota F v rekurzívnych prípadoch odvodená od hodnoty F pre podvýrazy.

Typická definícia štrukturálnou rekurziou na výrazoch má tvar:

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

Príklad 38. Veľkosť výrazu $\mathsf{Sz}(t) \equiv |t|$

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

IX.6 Ohodnotenie premenných

Na pripomenutie — funkcia indexovania do zoznamu

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

Definícia 39. *Ohodnotením (valuáciou) premenných* nazveme ľubovoľný zoznam.
Hodnotou premennej x_i pri ohodnotení vs je číslo $vs[i]$.

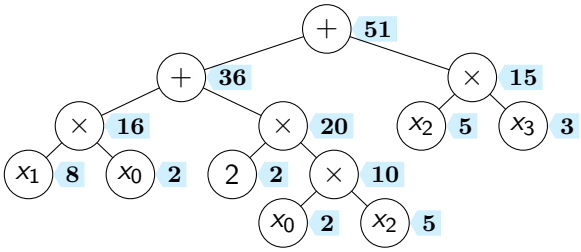
Príklad 40. Pri ohodnotení $2, 8, 5, 3, 0, 12, 9, 0$ majú premenné nasledujúce hodnoty:

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

IX.7 Hodnota výrazu — príklad

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

Príklad 41. Pri ohodnotení $2, 8, 5, 3, 0, 12, 9, 0$ vyhodnotíme výraz $((x_1 \times x_0) + (2 \times (x_0 \times x_2))) + (x_2 \times x_3)$:



IX.8 Hodnota výrazu pri ohodnotení premenných

Hodnota (*denotácia*) výrazu pri danom ohodnotení premenných — $\mathrm{Den}(t, vs) \equiv [\![t]\!]^{vs}$

Zadefinujeme štrukturálnou rekurziou na výrazoch (t.j. zdola nahor):

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

16.2. Transformácia aritmetických výrazov

16.2.1. Súčtový normálny tvar

IX.9 Súčtový normálny tvar

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

  Definícia 42. *Súčinový výraz* je aritmetický výraz tvorený iba konštantami, premennými a násobením.

  $\mathrm{Mterm}(x_i^\bullet)$
  $\mathrm{Mterm}(n^\bullet)$
  $\mathrm{Mterm}(t_1 \times^\bullet t_2) \leftarrow \mathrm{Mterm}(t_1) \wedge \mathrm{Mterm}(t_2)$

  Definícia 43. Výraz je v *súčtovom normálnom tvare*, ak je súčinovým výrazom, alebo súčtom výrazov v súčtovom normálnom tvare. Nič iné nie je výrazom v súčtovom normálnom tvare.

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

- Roznásobenie je prevod do súčtového normálneho tvaru

IX.10 Prevod do súčtového normálneho tvaru

- Špecifikácia funkcie $\mathrm{Anf\_from}(t)$:

$$\mathrm{Term}(t) \rightarrow \mathrm{Term}\,\mathrm{Anf\_from}(t) \wedge \mathrm{Anf}\,\mathrm{Anf\_from}(t) \\ \wedge [\![\mathrm{Anf\_from}(t)]\!]^{vs} = [\![t]\!]^{vs}$$

- Funkciu naprogramujeme štrukturálnou rekurziou na t, teda postupujeme zdola nahor
- Konštanta a premenná už v súčtovom normálnom tvare sú
- Pre súčet $s +^\bullet t$ prevedieme oba podvýrazy do s. n. t. *as* a *at*. Ich súčet $as +^\bullet at$ bude v s. n. t.
- Výraz $s \times^\bullet t \ldots$

IX.11 Prevod súčinu do súčtového normálneho tvaru

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

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

     s_i a t_i budú súčinové výrazy (platí pre ne Mterm)

  2. Potom ich vynásobíme dvoma pomocnými funkciami:

$$\begin{aligned}
&\mathrm{Mul}_{\mathrm{AA}}(as, at) \\
&= \mathrm{Mul}_{\mathrm{MA}}(s_1, at) +^\bullet \cdots +^\bullet \mathrm{Mul}_{\mathrm{MA}}(s_m, at) \\
&= (s_1 \times^\bullet t_1 +^\bullet s_1 \times^\bullet t_2 +^\bullet \cdots +^\bullet s_1 \times^\bullet t_n) +^\bullet \\
&\quad (s_2 \times^\bullet t_1 +^\bullet s_2 \times^\bullet t_2 +^\bullet \cdots +^\bullet s_2 \times^\bullet t_n) +^\bullet \\
&\quad \cdots \qquad\qquad +^\bullet \\
&\quad (s_m \times^\bullet t_1 +^\bullet s_m \times^\bullet t_2 +^\bullet \cdots +^\bullet s_m \times^\bullet t_n)
\end{aligned}$$

IX.12 Násobenie výrazov v súčtovom normálnom tvare

- Pomocné funkcie násobenia výrazov zachovajú súčtový normálny tvar aj hodnotu pôvodného súčinu:

$$\begin{aligned}
&\mathrm{Anf}(as) \wedge \mathrm{Anf}(at) \rightarrow \\
&\quad \mathrm{Anf}\, \mathrm{Mul}_{\mathrm{AA}}(as, at) \wedge [\![\mathrm{Mul}_{\mathrm{AA}}(as, at)]\!]^{vs} = [\![as \times^\bullet at]\!]^{vs} \\
&\mathrm{Mterm}(s_i) \wedge \mathrm{Anf}(at) \rightarrow \\
&\quad \mathrm{Anf}\, \mathrm{Mul}_{\mathrm{MA}}(s_i, at) \wedge [\![\mathrm{Mul}_{\mathrm{MA}}(s_i, at)]\!]^{vs} = [\![s_i \times^\bullet at]\!]^{vs}
\end{aligned}$$

- $\mathrm{Mul}_{\mathrm{AA}}$ a $\mathrm{Mul}_{\mathrm{MA}}$ naprogramujeme štrukturálnou rekurziou na as resp. at.

16.2.2. Zátvorkovanie doľava

IX.13 Prezátvorkovanie sčítania doľava

- Sčítanie je asociatívna operácia: $x + (y + z) = (x + y) + z$
- Zjednoťme uzátvorkovanie sčítaní prezátvorkovaním doľava

  Príklad 44.

  $$\begin{aligned}\mathsf{Lassoc\_from}\big(\quad & (x_0^\bullet +^\bullet x_1^\bullet) +^\bullet (1^\bullet +^\bullet (x_2^\bullet +^\bullet (2^\bullet +^\bullet x_3^\bullet)))\quad\big)\\ &= ((((x_0^\bullet +^\bullet x_1^\bullet) +^\bullet 1^\bullet) +^\bullet x_2^\bullet) +^\bullet 2^\bullet) +^\bullet x_3^\bullet\end{aligned}$$

- Výsledný tvar bude:

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

  $\mathsf{Lassoc}(n^\bullet)$
  $\mathsf{Lassoc}(x_i^\bullet)$
  $\mathsf{Lassoc}(t +^\bullet n^\bullet) \leftarrow \mathsf{Lassoc}(t)$
  $\mathsf{Lassoc}(t +^\bullet x_i^\bullet) \leftarrow \mathsf{Lassoc}(t)$
  $\mathsf{Lassoc}(t +^\bullet (t_1 \times^\bullet t_2)) \leftarrow \mathsf{Lassoc}(t)$
  $\mathsf{Lassoc}(t_1 \times^\bullet t_2)$

IX.14 Prezátvorkovanie sčítania doľava

- Špecifikácia prezátvorkovacej funkcie Lassoc_from

  $$\begin{aligned}\mathsf{Term}(t) \rightarrow \mathsf{Term}\,\mathsf{Lassoc\_from}(t) \wedge [\![\mathsf{Lassoc\_from}(t)]\!]^{vs} &= [\![t]\!]^{vs}\\ \wedge\, \mathsf{Lassoc}\,\mathsf{Lassoc\_from}(t) \wedge |\mathsf{Lassoc\_from}(t)| &= |t|\end{aligned}$$

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

  $\mathsf{Lassoc\_from}(n^\bullet) = n^\bullet$
  $\mathsf{Lassoc\_from}(x_i^\bullet) = x_i^\bullet$
  $\mathsf{Lassoc\_from}(t_1 +^\bullet t_2) = ?$
  $\mathsf{Lassoc\_from}(t_1 \times^\bullet t_2) = t_1 \times^\bullet t_2$

- Táto rekurzia **nie je** štrukturálna!
  Skončí?
  Aká *miera* sa znižuje pri rekurzívnom volaní?

X. prednáška
Výrokové formuly. Postfixový stroj a kompilátor výrazov

4. mája 2015

16 Aritmetické výrazy a výrokové formuly (pokračovanie)

16.3. Výrokové formuly

X.1 Výrokové formuly

Definícia 46. *Výrokovou formulou* (skrátene *formulou*) je

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

Nič iné nie je výroková formula.

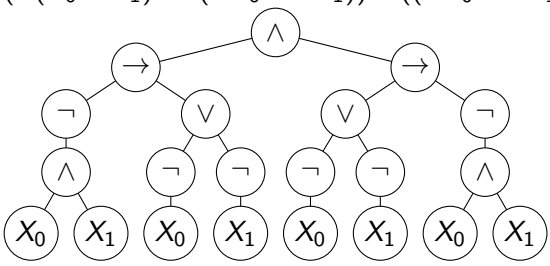
Príklad 47.

- $X_5 \vee (X_5 \to \bot)$
- $(\neg(X_0 \wedge X_1) \to (\neg X_0 \vee \neg X_1)) \wedge ((\neg X_0 \vee \neg X_1) \to \neg(X_0 \wedge X_1))$

- Výrokové formuly majú stromovú štruktúru

  Príklad 48.

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



- Zakódujeme ich párovacími konštruktormi

$$
\begin{array}{lll}
X_i^\bullet \equiv \mathsf{Vf}(i) & \bot^\bullet \equiv \mathsf{Ff} & \top^\bullet \equiv \mathsf{Tf} \\
\neg^\bullet f_1 \equiv \mathsf{Nf}(f_1) & f_1 \vee^\bullet f_2 \equiv \mathsf{Of}(f_1, f_2) & \\
f_1 \wedge^\bullet f_1 \equiv \mathsf{Af}(f_1, f_2) & f_1 \to^\bullet f_2 \equiv \mathsf{If}(f_1, f_2) &
\end{array}
$$

X.3 Kódovanie výrokových formúl

Príklad 49. Formulu $(\neg(X_0 \wedge X_1) \to (\neg X_0 \vee \neg X_1)) \wedge ((\neg X_0 \vee \neg X_1) \to \neg(X_0 \wedge X_1))$ kóduje konštanta

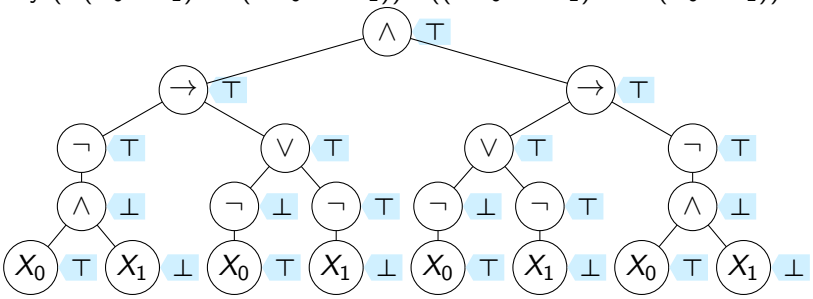
$$
\begin{array}{l}
\mathsf{De\_morgan}_1 = \mathsf{Af}(\mathsf{If}(\mathsf{Nf}\,\mathsf{Af}(\mathsf{Vf}(0), \mathsf{Vf}(1)), \\
\qquad\qquad \mathsf{Of}(\mathsf{Nf}\,\mathsf{Vf}(0), \mathsf{Nf}\,\mathsf{Vf}(1))), \\
\qquad\quad \mathsf{If}(\mathsf{Of}(\mathsf{Nf}\,\mathsf{Vf}(0), \mathsf{Nf}\,\mathsf{Vf}(1)), \\
\qquad\qquad \mathsf{Nf}\,\mathsf{Af}(\mathsf{Vf}(0), \mathsf{Vf}(1))))
\end{array}
$$

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

$$
\begin{array}{ll}
\mathsf{Formula}(X_i^\bullet) & \leftarrow \mathsf{N}(i) \\
\mathsf{Formula}(\bot^\bullet) & \\
\mathsf{Formula}(\top^\bullet) & \\
\mathsf{Formula}(\neg^\bullet f_1) & \leftarrow \mathsf{Formula}(f_1) \\
\mathsf{Formula}(f_1 \vee^\bullet f_2) & \leftarrow \mathsf{Formula}(f_1) \wedge \mathsf{Formula}(f_2) \\
\mathsf{Formula}(f_1 \wedge^\bullet f_2) & \leftarrow \mathsf{Formula}(f_1) \wedge \mathsf{Formula}(f_2) \\
\mathsf{Formula}(f_1 \to^\bullet f_2) & \leftarrow \mathsf{Formula}(f_1) \wedge \mathsf{Formula}(f_2)
\end{array}
$$

Definícia 50. *Ohodnotením (valuáciou) výrokových premenných* nazveme ľubovoľný zoznam obsahujúci iba čísla 0 a 1.
Premenná X_i je pravdivá pri ohodnotení vs práve vtedy, keď $vs[i] = 1$.

Príklad 51. Pri danom ohodnotení $vs = 1, 0, 0$ vyhodnotíme pravdivosť formuly $(\neg(X_0 \wedge X_1) \to (\neg X_0 \vee \neg X_1)) \wedge ((\neg X_0 \vee \neg X_1) \to \neg(X_0 \wedge X_1))$ takto:



Pravdivosť výrokovej formuly pre dané ohodnotenie premenných — *predikát* $\mathrm{True}(vs, f) \equiv (vs \models f)$
Zadefinujeme štrukturálnou rekurziou na formulách:

$$
\begin{array}{ll}
vs \models \top^\bullet & \\
vs \models X_i^\bullet & \leftarrow vs[i] = 1 \\
vs \models (\neg^\bullet f_1) & \leftarrow \neg(vs \models f_1) \\
vs \models (f_1 \wedge^\bullet f_2) & \leftarrow \cdots vs \models f_1 \cdots vs \models f_2 \cdots \\
vs \models (f_1 \vee^\bullet f_2) & \leftarrow \cdots \\
vs \models (f_1 \to^\bullet f_2) & \leftarrow \cdots
\end{array}
$$

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

$$
\begin{array}{ll}
\neg(vs \models \bot^\bullet) & \\
\neg(vs \models X_i^\bullet) & \leftarrow vs[i] \neq 1 \\
\neg(vs \models (\neg^\bullet f_1)) & \leftarrow vs \models f_1 \\
\ldots &
\end{array}
$$

16.4. Transformácia výrokových formúl

X.6 Redukcia množiny spojok

Niektoré logické spojky postačujú na vyjadrenie ostatných, napríklad:

Definícia 52. Výroková formula je *konjunktívna* práve vtedy, keď sa skladá iba z výrokových premenných, negácií a konjunkcií.

$\mathrm{Cform}(X_i^\bullet)$
$\mathrm{Cform}(\neg^\bullet f_1) \quad \leftarrow \mathrm{Cform}(f_1)$
$\mathrm{Cform}(f_1 \wedge^\bullet f_2) \leftarrow \mathrm{Cform}(f_1) \wedge \mathrm{Cform}(f_2)$

Tvrdenie 53. *Ku každej výrokovej formule existuje konjuktívna formula, ktorá je s ňou ekvivalentná.*

Naprogramujme funkciu Cform_from, ktorá ju zostrojí, teda:

$$\mathrm{Form}(f) \rightarrow \mathrm{Form}\,\mathrm{Cform\_from}(f) \wedge \mathrm{Cform}\,\mathrm{Cform\_from}(f)$$
$$\mathrm{Form}(f) \rightarrow \forall vs(vs \models \mathrm{Cform\_from}(f) \leftrightarrow vs \models f)$$

X.7 Redukcia množiny spojok

Na vyjadrenie iných spojok iba pomocou konjunktívnych formúl využijeme vlastnosti:

$$\begin{aligned} \bot &\iff (A \wedge \neg A) \\ (A \vee B) &\iff \neg(\neg A \wedge \neg B) \\ (A \rightarrow B) &\iff (\neg A \vee B)\,, \end{aligned}$$

ktoré platia pre ľubovoľné výrokové formuly A a B
Transformačnú funkciu potom zadefinujeme jednoduchou štrukturálnou rekurziou:

$$
\begin{aligned}
&\mathsf{Cform\_from}(X_i^\bullet) &&= X_i^\bullet \\
&\mathsf{Cform\_from}(\neg^\bullet f_1) &&= \neg^\bullet \mathsf{Cform\_from}(f_1) \\
&\mathsf{Cform\_from}(f_1 \wedge^\bullet f_2) &&= \mathsf{Cform\_from}(f_1) \wedge^\bullet \mathsf{Cform\_from}(f_2) \\
&\mathsf{Cform\_from}(\bot^\bullet) &&= X_0^\bullet \wedge^\bullet \neg^\bullet X_0^\bullet \\
&\mathsf{Cform\_from}(\top^\bullet) &&= \cdots \\
&\mathsf{Cform\_from}(f_1 \vee^\bullet f_2) &&= \cdots \mathsf{Cform\_from}(f_1) \cdots \mathsf{Cform\_from}(f_2) \cdots \\
&\mathsf{Cform\_from}(f_1 \rightarrow^\bullet f_2) &&= \cdots \mathsf{Cform\_from}(f_1) \cdots \mathsf{Cform\_from}(f_2) \cdots
\end{aligned}
$$

X.8 Negačný normálny tvar

Definícia 54. Výroková formula je v *negačnom normálnom tvare* práve vtedy, keď sa skladá iba z výrokových premenných, negácií výrokových premenných, disjunkcií a konjunkcií.

$$
\begin{aligned}
&\mathsf{Nnf}(X_i^\bullet) \\
&\mathsf{Nnf}(\neg^\bullet X_i^\bullet) \\
&\mathsf{Nnf}(f_1 \vee^\bullet f_2) \leftarrow \mathsf{Nnf}(f_1) \wedge \mathsf{Nnf}(f_2) \\
&\mathsf{Nnf}(f_1 \wedge^\bullet f_2) \leftarrow \mathsf{Nnf}(f_1) \wedge \mathsf{Nnf}(f_2)
\end{aligned}
$$

Tvrdenie 55. *Ku každej výrokovej formule existuje formula v negačnom normálnom tvare, ktorá je s ňou ekvivalentná.*

Naprogramujme funkciu Nnf_from, ktorá ju zostrojí, teda:

$$
\begin{aligned}
&\mathsf{Form}(f) \rightarrow \mathsf{Form}\,\mathsf{Nnf\_from}(f) \wedge \mathsf{Nnf}\,\mathsf{Nnf\_from}(f) \\
&\mathsf{Form}(f) \rightarrow \forall vs(vs \models \mathsf{Nnf\_from}(f) \leftrightarrow vs \models f)
\end{aligned}
$$

X.9 Prevod do negačného normálneho tvaru

Na prevod do negačného normálneho tvaru potrebujeme:

1. Distribuovať negáciu cez všetky spojky až k výrokovým premenným

   Využijeme de Morganove zákony:

$$
\begin{aligned}
\neg(A \vee B) &\iff (\neg A \wedge \neg B) \\
\neg(A \wedge B) &\iff (\neg A \vee \neg B)
\end{aligned}
$$

2. Odstrániť implikáciu a konštantné formuly (pravda, nepravda)

   Využijeme vyjadrenia $\bot$ a implikácie ako pri Cform_from

X.10 Prevod do negačného normálneho tvaru — program

Výsledný program musí pokryť mnoho prípadov:

$$
\begin{aligned}
&\mathrm{Nnf\_from}(X_i^\bullet) && = X_i^\bullet \\
&\mathrm{Nnf\_from}(f_1 \vee^\bullet f_2) && = \mathrm{Nnf\_from}(f_1) \vee^\bullet \mathrm{Nnf\_from}(f_2) \\
&\mathrm{Nnf\_from}(f_1 \wedge^\bullet f_2) && = \mathrm{Nnf\_from}(f_1) \wedge^\bullet \mathrm{Nnf\_from}(f_2) \\
&\mathrm{Nnf\_from}(\bot^\bullet) && = X_0^\bullet \wedge^\bullet \neg^\bullet X_0^\bullet \\
&\mathrm{Nnf\_from}(\top^\bullet) && = \cdots \\
&\mathrm{Nnf\_from}(f_1 \to^\bullet f_2) && = \cdots \\
&\mathrm{Nnf\_from}(\neg^\bullet X_i^\bullet) && = \cdots \\
&\mathrm{Nnf\_from}(\neg^\bullet \top^\bullet) && = \cdots \\
&\mathrm{Nnf\_from}(\neg^\bullet \bot^\bullet) && = \cdots \\
&\mathrm{Nnf\_from}(\neg^\bullet \neg^\bullet f_1) && = \cdots \\
&\mathrm{Nnf\_from}(\neg^\bullet (f_1 \vee^\bullet f_2)) && = \mathrm{Nnf\_from}(\neg^\bullet f_1) \wedge^\bullet \mathrm{Nnf\_from}(\neg^\bullet f_2) \\
&\mathrm{Nnf\_from}(\neg^\bullet (f_1 \wedge^\bullet f_2)) && = \cdots \\
&\mathrm{Nnf\_from}(\neg^\bullet (f_1 \to^\bullet f_2)) && = \cdots
\end{aligned}
$$

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

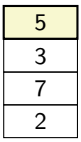
Rekurzia **nie je** štrukturálna, podobne, ako nebola pri Lassoc_from (rekurzívny argument nie je vždy podformulou pôvodného), ale existuje *miera*, v ktorej sú rekurzívne argumenty menšie — *aká?*

17. Postfixový stroj. Kompilátor výrazov

17.1. Postfixový stroj

X.11 Postfixový stroj

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



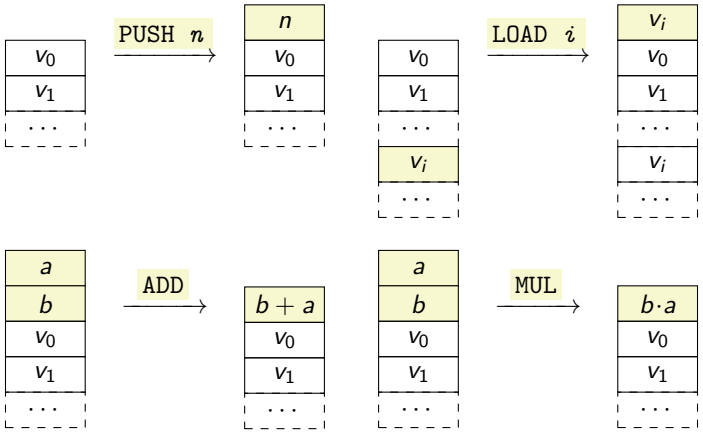
- Program: postupnosť inštrukcií

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

Inštrukcie modifikujú zásobník

Pridávajú/odoberajú čísla na vrch/z vrchu zásobníka

X.12 Inštrukcie postfixového stroja

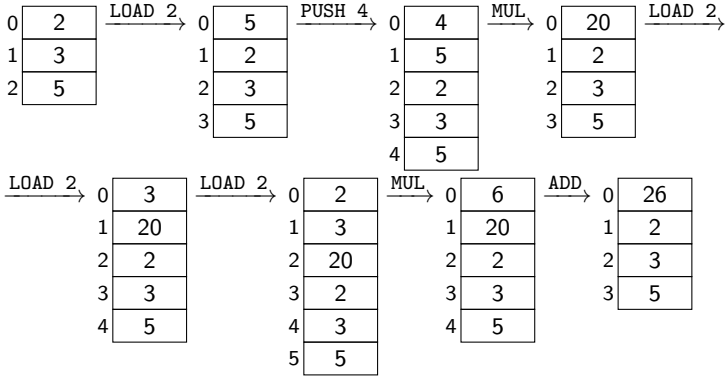


X.13 Príklad výpočtu programu

Počítajme program

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

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

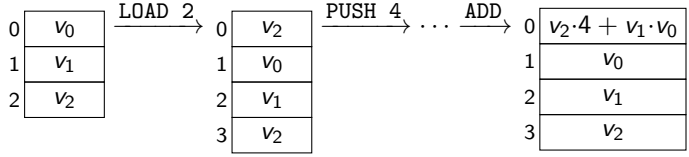


X.14 Príklad výpočtu programu — všeobecnejšie

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:



Symbolické vyhodnotenie

X.15 Kódovanie postfixových strojov

Postfixové stroje ľahko *zakódujeme* v CL:

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

$$\texttt{PUSH}(n) \equiv \text{Pushi}(n) = 0, n \qquad \texttt{ADD} \equiv \text{Addi} = 2, 0$$
$$\texttt{LOAD}(i) \equiv \text{Loadi}(i) = 1, i \qquad \texttt{MUL} \equiv \text{Muli} = 3, 0$$

Predikát 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

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

X.16 Simulácia postfixových strojov

Zakódovaný postfixový stroj môžeme *simulovať*:

- Funkcia $\mathsf{Run}(is, vs)$
- Simuluje beh postfixového stroja s programom *is* a zásobníkom *vs*
- *Interpretuje* program *is*
- Vráti vrch zásobníka na konci programu
- Klauzálna definícia:

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

  Rekurzia bude chvostová

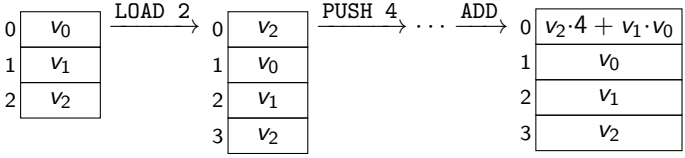
17.2. Kompilácia aritmetických výrazov

X.17 Programy a výrazy — príklad

Program

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

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



vypočíta na vrch zásobníka hodnotu aritmetického výrazu

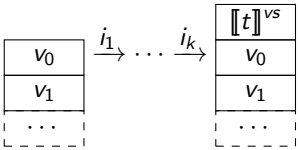
$$(x_2 \times 4) + (x_1 \times x_0)$$

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

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

X.18 Programy a výrazy — kompilácia

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



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

X.19 Kompilácia — príklad

Videli sme, že

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

Chceme, aby kompilátor skompiloval výraz

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

do programu

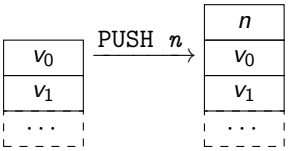
$$\texttt{LOAD}(2), \texttt{PUSH}(4), \texttt{MUL}, \texttt{LOAD}(2), \texttt{LOAD}(2), \texttt{MUL}, \texttt{ADD}, 0$$

teda

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

X.20 Kompilácia — konštanty

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

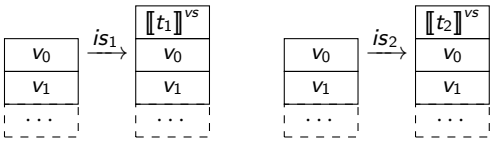


- Klauzula kompilátora:

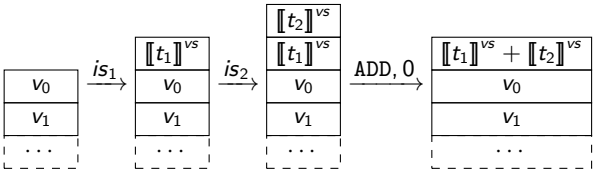
$$\mathsf{Comp}(n^\bullet) = \texttt{PUSH}(n), 0$$

X.21 Kompilácia — sčítanie

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



  2. Zreťazíme ich a pridáme inštrukciu pre sčítanie

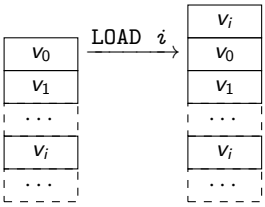


- Klauzula kompilátora:

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

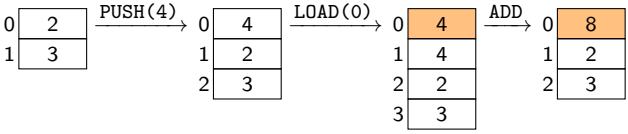
X.22 Kompilácia — premenná

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



- Skompilujme $\text{Comp}(4^\bullet +^\bullet x_0^\bullet) = \texttt{PUSH}(4), \texttt{LOAD}(0), \texttt{ADD}, 0$

- Otestujme s ohodnotením $2, 3, 0$:



Zle! `LOAD(0)` malo načítať 2

X.23 Kompilácia — medzivýsledky

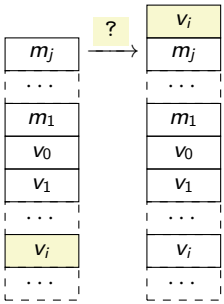
- Kompilátor si musí pamätať počet j *medzivýsledkov* v zásobníku
- Potrebujeme o niečo *všeobecnejšiu* kompilačnú funkciu $\mathrm{Comp}_1(t, j)$:

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

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

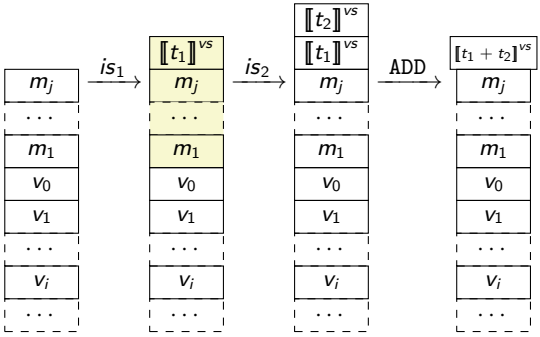
X.24 Kompilácia — medzivýsledky

- Aký program dá na vrch zásobníka hodnotu $vs[i]$, keď je v zásobníku j medzivýsledkov?



▶ $\mathrm{Comp}_1(x_i^\bullet, j) = ?$

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



▶ $\mathsf{Comp}_1(t_1 +^\bullet t_2, j) = \mathsf{Comp}_1(t_1, j) \oplus \mathsf{Comp}_1(t_2, ?) \oplus (\mathtt{ADD}, 0)$

17.3. Podmienené výrazy

Rozšírme výrazy o dva ďalšie typy:

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

Zakódujeme ich novými konštruktormi:

$$t_1 \dot{-}^\bullet t_2 \equiv \mathsf{St}(t_1, t_2) = 4, t_1, t_2$$
$$\mathbf{D}^\bullet(t_1, t_2, t_3) \equiv \mathsf{Dt}(t_1, t_2, t_3) = 5, t_1, t_2, t_3$$

Rozšírime predikát Term:

$$\mathsf{Term}(n^\bullet) \leftarrow \mathsf{N}(n)$$
$$\ldots$$
$$\mathsf{Term}(t_1 \dot{-}^\bullet t_2) \leftarrow \mathsf{Term}(t_1) \wedge \mathsf{Term}(t_2)$$
$$\mathsf{Term}(\mathbf{D}^\bullet(t_1, t_2, t_3)) \leftarrow \mathsf{Term}(t_1) \wedge \mathsf{Term}(t_2) \wedge \mathsf{Term}(t_3)$$

a denotačnú funkciu:

$$\ldots$$
$$[\![t_1 \dot{-}^\bullet t_2]\!]^{vs} = [\![t_1]\!]^{vs} \dot{-} [\![t_2]\!]^{vs}$$
$$[\![\mathbf{D}^\bullet(t_1, t_2, t_3)]\!]^{vs} = ? \leftarrow ?$$

X.27 Podmienené výrazy – použitie

- Rozdielovým a podmieneným výrazom vyjadríme napríklad maximum:

  $$\mathrm{Dterm}_1 = \mathbf{D}^\bullet(x_0^\bullet \mathbin{\dot{-}^\bullet} x_1^\bullet, x_0^\bullet, x_1^\bullet)$$

  vďaka vlastnosti

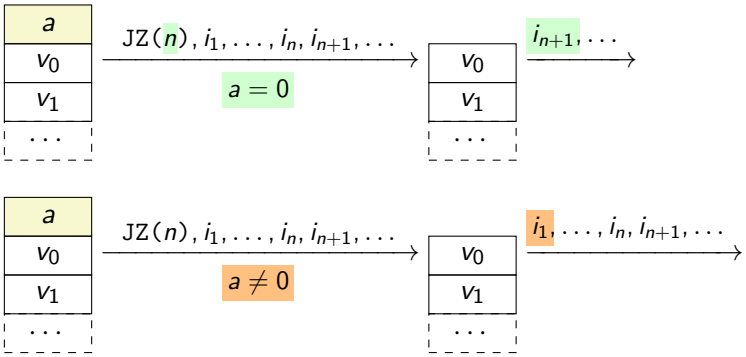
  $$x \mathbin{\dot{-}} y \neq 0 \leftrightarrow x > y$$

- Mohli by sme si zadefinovať aj skratku

  $$\mathrm{Term\_dich}(ta, tb, tgt, tleq) = \mathbf{D}^\bullet(ta \mathbin{\dot{-}^\bullet} tb, tgt, tlet)$$

X.28 Podmienený skok a odčítanie

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



- Potrebujeme aj inštrukciu odčítania `SUB`

  Funguje podobne ako `ADD` a `MUL`

X.29 Kódovanie a simulácia skoku a odčítania

- Zakódujme nové inštrukcie:

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

  $\mathsf{Instr}(\cdots)$
  $\mathsf{Instr}(\texttt{SUB})$
  $\mathsf{Instr}\,\texttt{JZ}(n) \leftarrow \mathsf{N}(n)$

- Rozšírme simulačnú funkciu:

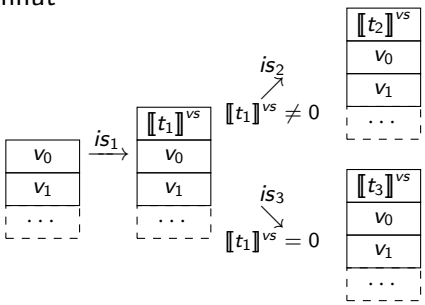
  $\mathsf{Run}(\cdots, \cdots) = \cdots$
  $\mathsf{Run}((\texttt{JZ}(n), is), (a, vs)) = \mathsf{Run}(?, ?) \leftarrow a = 0$
  $\mathsf{Run}((\texttt{JZ}(n), is), (a, vs)) = \mathsf{Run}(?, ?) \leftarrow a \neq 0$

  Potrebujeme preskočiť časť programu:

  $\mathsf{Drop}(0, xs) = xs$
  $\mathsf{Drop}(n+1, 0) = 0$
  $\mathsf{Drop}(n+1, (x, xs)) = \mathsf{Drop}(n, xs)$

X.30 Kompilácia – podmienený výraz

- Hodnotou výrazu $\mathbf{D}^{\bullet}(t_1, t_2, t_3)$ je číslo $[\![t_2]\!]^{vs}$, ak $[\![t_1]\!]^{vs} \neq 0$, a číslo $[\![t_3]\!]^{vs}$, ak $[\![t_1]\!]^{vs} = 0$;
- Akým programom ju vypočítame bez počítania hodnoty nepotrebného výrazu?
  1. Skompilujeme výrazy t_1 až t_3 do programov is_1 až is_3
  2. Chceme dosiahnuť



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

X.31 Kompilácia – podmienený výraz

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

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

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

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

Ako to dosiahneme?

XI. prednáška
Všeobecné stromy

11. mája 2015

Organizácia skúšky a opravného testu

Opravný test

XI.1 Opravný test

Termín: streda 27. mája o 13:30 v H6

Účel: *nahradiť* skóre z jedného semestrálneho testu

Materiál: ako opravovaný test

Podmienky:

- *záväzné prihlásenie* (AIS) najnesôr v piatok 22. mája
  - účasť bez prihlásenia nie je možná
- výber opravovaného testu priamo na termíne
  - neprítomní prihlásení: 2. test
- *nové skóre nahradí pôvodné skóre bez ohľadu na ich vzájomný rozdiel*

  $\Rightarrow$ môžete si pohoršiť
  - skóre neprítomných prihlásených je 0

Skúška

XI.2 Skúška

Termíny:	**Dátum**	**Čas**	**M.**	**Typ** (max. počet študentov)	
	streda 3. júna	13:30	H6	riadny (32)	
	17. júna			riadny (16),	1. opravný (16)
	24. júna			1. opravný (16),	2. opravný (16)
	1. júla			2. opravný (16)	

Podmienky:

- súčet skóre z testov ≥ 30
- záväzné prihlásenie (AIS) najneskôr o 12:00 v deň pred termínom

Materiál: úlohy najmä podľa ex09–ex12

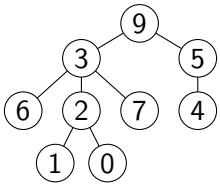
Konzultácie: pondelky 25. mája, 1., 15. a 22. júna od 13:00 do 15:00 v H3

18. Všeobecné stromy

18.1. Kódovanie

XI.3 Všeobecné stromy — kódovanie

Všeobecný strom — ľubovoľne veľa detí každého vrcholu:

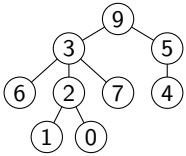


Ako zakódujeme takýto strom v CL?

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

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

  x — hodnota uložená vo vrchole stromu, ts — zoznam detí
- Ako zakódujeme strom na obrázku?



- Ako zakódujeme strom na obrázku?

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

$$\equiv \frac{9}{\frac{3}{\frac{6}{0}, \frac{2}{0}, \frac{7}{\frac{1}{0}, \frac{0}{0}, 0}}, 0}, \frac{5}{\frac{4}{0}, 0}, 0$$

- 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:

$$\begin{aligned}F_1\left(\frac{x}{ts}\right) = \cdots \qquad &\equiv \qquad F_1\,\mathsf{Gnd}(x, ts) = \cdots\\ F_2(0) = \cdots \qquad &\equiv \qquad F_2(0) = \cdots\\ F_2\left(\frac{x}{ts_1}, ts_2\right) = \cdots \qquad &\equiv \qquad F_2(\mathsf{Gnd}(x, ts_1), ts_2) = \cdots\end{aligned}$$

XI.6 Všeobecné stromy

Aké čísla kódujú všeobecné stromy?

- Rozpoznáme ich predikátom Gt:

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

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

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

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

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

Rekurziu na zoznamoch všeobecných stromov využijeme aj ďalej

XI.7 Všeobecné stromy so znakmi vo vrcholoch

- Všeobecné stromy so znakmi vo vrcholoch:

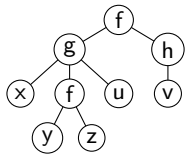
  $$\begin{array}{l}\mathrm{Lgtc}(0)\\ \mathrm{Lgtc}\left(\frac{x}{ts_1}, ts_2\right) \leftarrow \mathrm{Ch}(x) \wedge \mathrm{Lgtc}(ts_1) \wedge \mathrm{Lgtc}(ts_2)\\ \mathrm{Gtc}\left(\frac{x}{ts}\right) \leftarrow \mathrm{Ch}(x) \wedge \mathrm{Lgtc}(ts)\end{array}$$

- Môžeme ich chápať ako výrazy s funkciami ľubovoľných arít (vrátane

konštánt — teda funkcií nulovej arity)

S2gtc('f(g(x,f(y,z),u),h(v))')

$$= \frac{\text{“f”}}{\frac{\text{“g”}}{\frac{\text{“x”}}{0}, \frac{\text{“f”}}{\frac{\text{“y”}}{0}, \frac{\text{“z”}}{0}, 0}, \frac{\text{“u”}}{0}, 0}, \frac{\text{“h”}}{\frac{\text{“v”}}{0}, 0}, 0} =$$



XI.8 Odbočka o zovšeobecňovaní

Pozorovanie 56. *Zovšeobecnenie často vedie k (efektívnemu) riešeniu.*

Príklad 57. Riešenia vďaka zovšeobecneniu:

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

Efektívne riešenia vďaka zovšeobecneniu:

- Funkcia $\text{While\_fib}(n, a, b)$ je všeobecnejšia ako $\text{Fib}(n{+}1) = \text{While\_fib}(n, 0, 1)$
- $\text{While\_rev}(xs, rs) = \text{Rev}(xs) \oplus rs$ je všeobecnejšia $\text{Rev}(xs) = \text{While\_rev}(xs, 0)$
- $\text{Preordera}(t, as) = \text{Preorder}(t) \oplus as$ je všeobecnejšia ako $\text{Preorder}(t) = \text{Preordera}(t, 0)$

18.2. Programovanie so všeobecnými stromami

XI.9 Programovanie so všeobecnými stromami

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

$$F_{\mathsf{L}}(0) = \cdots$$
$$F_{\mathsf{L}}\left(\tfrac{x}{ts_1}, ts_2\right) = \cdots F_{\mathsf{L}}(ts_1) \cdots F_{\mathsf{L}}(ts_2) \cdots$$
$$F\left(\tfrac{x}{ts}\right) = \cdots F_{\mathsf{L}}(ts) \cdots$$

- Príklad: Zadefinujme funkciu $|t|_{\mathsf{g}}$ počítajúcu veľkosť všeobecného stromu t

$$|0|_{\mathsf{L}} = 0$$
$$\left|\tfrac{x}{ts_1}, ts_2\right|_{\mathsf{L}} = 1 + |ts_1|_{\mathsf{L}} + |ts_2|_{\mathsf{L}}$$
$$\left|\tfrac{x}{ts}\right|_{\mathsf{g}} = 1 + |ts|_{\mathsf{L}}$$

XI.10 Zrkadlový obraz

- Zadefinujme zrkadlový obraz stromu t — funkciu $\mathsf{Reflect}(t)$

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

- Zovšeobecníme na zoznamy stromov $\mathsf{Reflect}_{\mathsf{L}}(ts)$

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

XI.11 Zrkadlový obraz

- Skombinujeme obrátenie zoznamu a zrkadlový obraz binárneho stromu

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

- Výsledná definícia

$$\mathsf{Reflect_L}(0) = ?$$
$$\mathsf{Reflect_L}\left(\frac{x}{ts_1}, ts_2\right) = ?$$

- 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:

$$\mathsf{Reflecta_L}\left(\left(\begin{array}{c}2\\1\ 0\end{array}, 7, 0\right), \left(6, 0\right)\right) = 7, \begin{array}{c}2\\0\ 1\end{array}, 6, 0$$

18.3. Prechody všeobecnými stromami

XI.12 Prechody všeobecnými stromami

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

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

- Opäť zovšeobecníme na zoznamy stromov — funkcia $\mathsf{Preorder_L}(ts)$

$$\mathsf{Preorder_L}\left(\begin{array}{c}3\\6\ 2\ 7\\1\ 0\end{array}, \begin{array}{c}5\\4\end{array}, 0\right) = 3, 6, 2, 1, 0, 7, 5, 4, 0$$

- 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:

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

18.4. Číslovanie vrcholov vo všeobecných stromoch

XI.14 Číslovanie vrcholov vo všeobecných stromoch

- Očíslujme vrcholy daného všeobecného stromu v poradí *post*order:

$$\text{Num\_post}\left(\begin{array}{c}f\\g\ \ h\\x\ f\ u\ v\\y\ z\end{array}\right) = \begin{array}{c}8\\5\ \ 7\\0\ 3\ 4\ 6\\1\ 2\end{array}$$

- Zovšeobecníme na číslovanie zoznamu všeobecných stromov od ľubovoľného m:

$$\text{Num\_post}_\text{L}\left(\left(\begin{array}{c}g\\x\ f\ u\\y\ z\end{array}, \begin{array}{c}h\\v\end{array}, 0\right), 2\right) = \begin{array}{c}m_0\ \ 7\ \ m_1\\2\ \ 5\ \ 6\\3\ \ 4\end{array}, \begin{array}{c}m_2\\9\\8\end{array}, 0$$

XII. prednáška
Generovanie XML/XHTML

18. mája 2015

19. XML/XHTML

19.1. Štruktúra

XII.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>
```

XII.2 Štruktúra XML/XHTML dokumentu

Štruktúra v XML dokumente — *elementy*

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

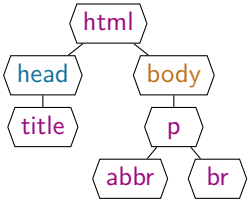
```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Príklad</title>
  </head>
  <body>
```

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

XII.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?

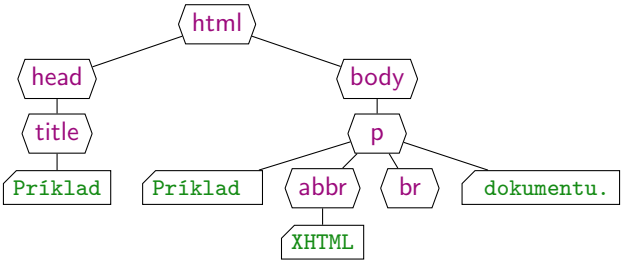
XII.4 Text dokumentu

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

V strome elementov chýba samotný text dokumentu

XII.5 Strom elementových a textových vrcholov

Pridáme vrcholy pre neštruktúrovaný text:



Všeobecný strom s dvoma druhmi vrcholov:

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

19.2. Kódovanie

XII.6 Kódovanie stromu dokumentu v CL

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

XII.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:

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

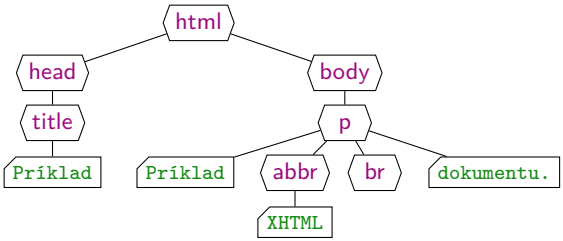
- Napríklad:

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

XII.8 Kódovanie detí elementu

Xe(n, as, cs)

- Elementové vrcholy majú rôzny počet detí:



- Deti zakódujeme do *zoznamu* vrcholov XML stromu

XII.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)
```

XII.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 vrchol XML stromu
  - Pomocný predikát $\mathsf{Lx}(xs)$ — xs je zoznam XML vrcholov

- Prirodzená definícia *vzájomnou rekurziou*:

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

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

- CL neumožňuje vzájomnú rekurziu:

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

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

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

Príklad 58.

Query: .....
$\mathsf{Xe}(\text{'pokus'}, 0, \mathsf{Xs}(\text{'text'}), 0), 0 = xs : \mathsf{Xml}$
Results: .....
$xs =$ <pokus>text</pokus>

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

Príklad 59.

Query: .....

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

Results: .....

xs =

Príklad dokumentu

Nadpis

19.3. Neformálne typovanie

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

$$F :: T_1 \to T_2$$

  - skratka za $T_1(x) \to T_2\big(F(x)\big)$
  - ak argument funkcie F je typu T_1, tak výsledok je typu T_2
- 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?

19.4. Generovanie

XII.15 Generovanie XML

Príklad 60. Zadefinujme funkciu Mk_ul(*items*), ktorá

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

Otypujme: Funkcia vytvorí jeden vrchol XML stromu:

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

Naprogramujme:

- Potrebujeme pomocnú funkciu Mk_lis(*items*), ktorá vytvorí zoznam vrcholov XML stromu (elementov `li`):

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

  $\text{Mk\_lis}(0) = 0$
  $\text{Mk\_lis}(\mathit{item}, \mathit{items}) = \text{Xe}(\text{'li'}, 0, \text{Xs}(\mathit{item}), 0), \text{Mk\_lis}(\mathit{items})$

- Vytvorené položky použijeme ako deti elementu `ul`:

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