Teoretické základy informatiky

M. Nehéz D. Chudá I. Polický M. Čerňanský

september 2006

Obsah

Kapitola 4

Zásobníkové automaty a bezkontextové jazyky

4.1 Nedeterministické zásobníkové automaty a vlastnosti bezkontextových jayzkov

Zásobníkový automat si môžeme zjednodušene predstaviť ako konečný automat, ku ktorému je navyše pridaná abstraktná údajová štruktúra *zásobník*. Pri manipulovaní so zásobníkom budeme používať nasledujúce operácie:

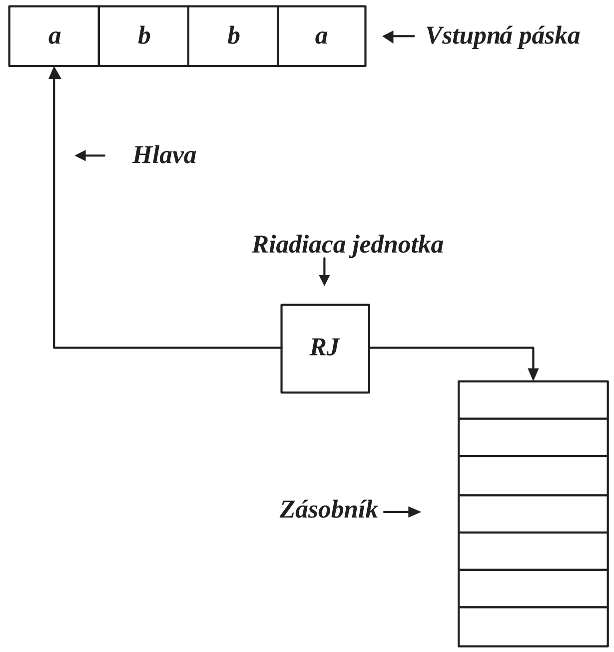
- `push` - pridanie prvku na vrchol zásobníka,
- `pop` - odobratie prvku z vrchola zásobníka,
- `top` - vráti znak, ktorý sa nachádza na vrchole zásobníka (prečítaný znak ostáva na svojom mieste),
- `skip` - prázdna operácia (obsah zásobníka ostáva nezmenený)
- `empty` - test na prázdnosť.

Operácie `top` a `empty` sú funkcie (t.j. ich výsledkom je vrátená hodnota). Funkcia `top` vráti znak, ktorý sa nachádza na vrchole zásobníka. Funkcia `empty` vráti vždy logickú hodnotu, pričom platí:

$$\texttt{empty} = \begin{cases} \mathit{true}, & \text{ak je zásobník prázdny,} \\ \mathit{false}, & \text{ak zásobník obsahuje aspoň jeden prvok.} \end{cases}$$

Schéma zásobníkového automatu je znázornená na obrázku 4.1.

Definícia 4.1.1 Nedeterministický zásobníkový automat



Obrázok 4.1: Schéma zásobníkového automatu.

Nedeterministický zásobníkový automat *(NPDA - nondeterministic push-down automaton) je sedemtica* $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, *kde:*

K *je konečná množina stavov,*

Σ *je vstupná abeceda,*

Γ *je zásobníková abeceda,*

δ *je prechodov, zobrazenie, pričom platí*

$$\delta : K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{K \times \Gamma^*}_{KON}$$

(Symbol $2^{K \times \Gamma^*}_{KON}$ *označuje konečné podmnožiny množiny* $K \times \Gamma^*$*.)*

$q_0 \in K$ *je počiatočný stav,*

$Z_0 \in \Gamma$ *je počiatočný zásobníkový symbol,*

$F \subseteq K$ *je množina koncových stavov.*

Konfigurácia nedeterministického zásobníkového automatu *je trojica*

$$(q, w, \gamma) \ \in \ K \times \Sigma^* \times \Gamma^*,$$

kde:

q *je momentálny stav,*

w *je zvyšok vstupu,*

γ *je aktuálny obsah zásobníka.*

Krok výpočtu nedeterministického zásobníkového automatu *je relácia* $\vdash_A$ *na množine konfigurácií* $K \times \Sigma^* \times \Gamma^*$ *definovaná nasledovne:*

$$(q, aw, Z\gamma) \vdash_A (p, w, \beta\gamma), \qquad ak \qquad (p, \beta) \in \delta(q, a, Z),$$

pričom $p,\, q \in K,\;\; a \in \Sigma \cup \{\varepsilon\},\;\; Z \in \Gamma,\;\; w \in \Sigma^*,\;\; \beta,\, \gamma \in \Gamma^*.$

Jazyk rozpoznávaný nedeterministickým zásobníkovým automatom *(koncovým stavom) je množina*

$$L(A) = \{\; w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_A^* (q, \varepsilon, \gamma),\; q \in F,\; \gamma \in \Gamma^* \;\}.$$

Výpočet nedeterministického zásobníkového automatu A na slove w je postupnosť konfigurácií $(q_0, w, Z_0) \vdash_A \ldots$.

Poznámka 4.1.1 *Definícia 4.1.1 zaručuje, že vstupné slovo je akceptované koncovým stavom, pričom obsah zásobníka môže byť* ľubovoľný, *teda aj neprázdny. Ekvivalentnú verziu nedeterministického zásobníkového automatu predstavuje automat, ktorý akceptuje* prázdnym zásobníkom. *(Jeho definíciu uvádzať nebudeme.)*

Namiesto slovného spojenia "nedeterministický zásobníkový automat" budeme používať aj skrátené označenie $NPDA$. Symbolom $\mathcal{L}(NPDA)$ budeme označovať triedu jazykov rozpoznávaných nedeterministickými zásobníkovými automatmi.

Veta 4.1.1 *Trieda jazykov rozpoznávaných nedeterministickými zásobníkovými automatmi je zhodná s triedou všetkých bezkontextových jazykov. Formálne:*

$$\mathcal{L}(NPDA) = \mathcal{L}_{CF}\ . \tag{4.1}$$

Z uvedenej vety vyplýva, že ku každému jazyku $L_0 \in \mathcal{L}_{CF}$ sa d zostrojiť nedeterministický zásobníkový automat A_0 tak, že $L(A_0) = L_0$. Tiež platí, že každý zásobníkový automat rozpoznáva nejaký bezkontextový jazyk. Dôkaz vety 4.1.1 je vysoko netriviálny.

Uzáverové vlastnosti triedy bezkontextových jazykov sú uvedené v nasledujúcich dvoch tvrdeniach.

Veta 4.1.2 *Trieda* $\mathcal{L}_{CF}$ *je uzavretá vzhľadom na operácie:*

1. *zjednotenie* $(\cup)$,
2. *zreťazenie* $(\cdot)$,
3. *Kleeneho iterácia* $(^*)$,
4. *prienik s regulárnymi jazykmi.*

Veta 4.1.3 *Trieda* $\mathcal{L}_{CF}$ *nie je uzavretá vzhľadom na operácie:*

1. *prienik* $(\cap)$,
2. *doplnok* $(^C)$.

Riešené príklady

Príklad 4.1.1 *Nech G je gramatika v Greibachovej normálnom tvare, pričom $G = (N, T, P, S)$, kde $N = \{S, A\}$, $T = \{a, b\}$,*
$P = \{$

$$\begin{array}{l} S \to aAA \;\mid\; a \\ A \to aSA \;\mid\; bS \;\mid\; b \end{array}$$

$\}$.
Zostrojíme zásobníkový automat A tak, aby platilo $L(A) = L(G)$. Nech $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde:
$K = \{q_0, q_1, q_F\}$,
$\Sigma = \{a, b\}$,
$\Gamma = \{Z_0, S, A\}$,
$F = \{q_F\}$,
$\delta(q_0, \varepsilon, Z_0) = (q_1, SZ_0)$,
$\delta(q_1, a, S) = \{\ (q_1, AA), (q_1, \varepsilon)\ \}$,
$\delta(q_1, a, A) = (q_1, SA)$,
$\delta(q_1, b, A) = \{\ (q_1, S), (q_1, \varepsilon)\ \}$,
$\delta(q_1, \varepsilon, Z_0) = (q_F, Z_0)$.
Pre ľubovoľné slovo $w \in L(A)$ existuje akceptujúci výpočet automatu A na tomto slove. Napríklad pre slovo abba existuje nasledujúce odvodenie v gramatike G.

$$S \Rightarrow aAA \Rightarrow abA \Rightarrow abbS \Rightarrow abba$$

Príslušný (nedeterminisitický) výpočet automatu A pre slovo abba zodpovedá jeho odvodeniu.

$$(q_0, abba, Z_0) \vdash (q_1, abba, SZ_0) \vdash (q_1, bba, AAZ_0) \vdash (q_1, ba, AZ_0) \vdash$$

$$\vdash (q_1, a, SZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_F, \varepsilon, Z_0)$$

Príklad 4.1.2 *Nech $\heartsuit$ je binárna operácia nad jazykmi definovaná nasledovne:*

$$L_1 \heartsuit L_2 = (L_1 \cup L_2) \cdot (L_1)^+ \ .$$

Dokážeme, že trieda bezkontextových jazykov $\mathcal{L}_{CF}$ je uzavretá vzhľadom na túto operáciu. Najprv upravíme vzťah určujúci operáciu $\heartsuit$:

$$L_1 \heartsuit L_2 = (L_1 \cup L_2) \cdot (L_1)^+ = L_1 \cdot L_1^+ \cup L_2 \cdot L_1^+ \ .$$

Nech L_1, $L_2 \in \mathcal{L}_{CF}$. Potom musia existovať bezkontextové gramatiky G_1 a G_2, že platí $L(G_1) = L_1$ a $L(G_2) = L_2$. Nech $G_1 = (N_1, T_1, P_1, S_1)$ a $G_2 = (N_2, T_2, P_2, S_2)$ sú uvedené bezkontextové gramatiky, pričom $N_1 \cap (N_2 \cup T_2) = \emptyset$ a $N_2 \cap (N_1 \cup T_1) = \emptyset$. (V prípade, keby táto podmienka nebola splnená, stačí premenovať jednotlivé neterminálne symboly.) Zostrojíme gramatiku $G_\heartsuit = (N_0, T_0, P_0, S_0)$ tak, že:

- $N_0 = N_1 \cup N_2 \cup \{S_0\}$,
- $T_0 = T_1 \cup T_2$,
- $P_0 = P_1 \cup P_2 \cup \{\ S_0 \to S_1S_1,\ S_0 \to S_2S_1,\ S_0 \to S_0S_1\ \}$.

To znamená, že S_0 je počiatočný neterminálny symbol gramatiky $G_\heartsuit$ a k pôvodným pravidlám gramatík G_1 a G_2 (tie sú zahrnuté ako pravidlá gramatiky $G_\heartsuit$) sú pridané ešte tri nové pravidlá: $S_0 \to S_1S_1$, $S_0 \to S_2S_1$ a $S_0 \to S_0S_1$. Z takejto konštrukcie vyplýva, že $L(G_\heartsuit) = (L(G_1) \cup L(G_2)) \cdot (L(G_1))^+$.

Z predpokladu, že gramatiky G_1 a G_2 sú bezkontextové vyplýva, že aj gramatika $G_\heartsuit$ je bezkontextová, takže platí: $L(G_\heartsuit) \in \mathcal{L}_{CF}$. Tým je dokázané, že trieda $\mathcal{L}_{CF}$ je uzavretá vzhľadom na operáciu $\heartsuit$.

4.2 Deterministické zásobníkové automaty a ich vzťah k nedeterministickým zásobníkovým automatom

Deterministický zásobníkový automat je špeciálnym prípadom nedeterministického zásobníkového automatu. (Jeho schéma zodpovedá obrázku 4.1.)

Definícia 4.2.1 Deterministický zásobníkový automat
Nech $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat v zmysle definície 4.1.1. Automat A sa nazýva deterministický zásobníkový automat, *(DPDA - deterministic push-down automaton) ak $\forall\ q \in K$, $\forall\ a \in \Sigma$, $\forall\ Z \in \Gamma$ platí:*

1. *$\delta(q, a, Z)$ obsahuje najviac jeden prvok,*
2. *$\delta(q, \varepsilon, Z)$ obsahuje najviac jeden prvok,*
3. *ak množina $\delta(q, \varepsilon, Z)$ nie je prázdna, potom množina $\delta(q, a, Z)$ je prázdna $\forall\ a \in \Sigma$.*

Namiesto slovného spojenia "deterministický zásobníkový automat" budeme používať aj skrátené označenie $DPDA$. Symbolom $\mathcal{L}(DPDA)$ budeme označovať triedu jazykov rozpoznávaných deterministickými zásobníkovými automatmi.

Uzáverové vlastnosti triedy $\mathcal{L}(DPDA)$ sú uvedené v nasledujúcom tvrdení.

Veta 4.2.1 *Trieda $\mathcal{L}(DPDA)$ je uzavretá vzhľadom na operácie:*

1. *prienik s regulárnymi jazykmi,*
2. *doplnok (^{C}).*

Rozlišujeme teda dve rôzne triedy jazykov:

- $\mathcal{L}(NPDA)$ - trieda jazykov rozpoznávaných nedeterministickými zásobníkovými automatmi,
- $\mathcal{L}(DPDA)$ - trieda jazykov rozpoznávaných deterministickými zásobníkovými automatmi.

Ich vzájomný vzt'ah určuje nasledujúce tvrdenie.

Veta 4.2.2 *Trieda jazykov rozpoznávaných deterministickými zásobníkovými automatmi je vlastnou podmnožinou triedy jazykov rozpoznávaných nedeterministickými zásobníkovými automatmi. Formálne:*

$$\mathcal{L}(DPDA) \subsetneq \mathcal{L}(NPDA) \ . \tag{4.2}$$

Príklad jazyka, ktorý patrí do triedy $\mathcal{L}(NPDA) \setminus \mathcal{L}(DPDA)$ uvádza nasledujúca lema.

Lema 4.2.1 *Nech* $L_N = \{ww^R \mid w \in \{a,b\}^+\}$.

1. *Neexistuje deterministický zásobníkový automat, ktorý by rozpoznával jazyk* L_N.
2. *Existuje nedeterministický zásobníkový automat* A_N*, pre ktorý platí:*
   $L(A_N) = L_N$.

Z toho vyplýva, že $L_N \in \mathcal{L}(NPDA) \setminus \mathcal{L}(DPDA)$.

Rozpoznávanie jazyka L_N je uvedené v príklade 4.2.5.

Riešené príklady

Deterministické zásobníkové automaty

Príklad 4.2.1 *Zásobníkový automat* A_1 *rozpoznáva jazyk* $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$.
Nech $A_1 = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$*, kde:*
$K = \{q_0, q_1, q_2, q_3\}$,
$\Sigma = \{a, b\}$,
$\Gamma = \{Z, Z_0\}$,
$F = \{q_0, q_3\}$,
$\delta(q_0, a, Z_0) = (q_1, ZZ_0)$ `push`(Z)
$\delta(q_1, a, Z) = (q_1, ZZ)$ `push`(Z)
$\delta(q_1, b, Z) = (q_2, \varepsilon)$ `pop`

$\delta(q_2, b, Z) = (q_2, \varepsilon)$ `pop`
$\delta(q_2, \varepsilon, Z_0) = (q_3, Z_0)$ `skip`, *akceptovanie.*

Symbol Z_0 sa používa na identifikáciu dna zásobníka, to znamená, že ho nie je možné zo zásobníka vybrať. Ak zásobník obsahuje iba symbol Z_0, znamená to, že je prázdny. (Napr. postupnosť $AAAZ_0$ reprezentuje zásobník, ktorý obsahuje tri symboly A.) Zásobníkové operácie sa zapisujú ako posledné prvky usporiadaných dvojíc na pravej strane rovnosti v prechodových funkciách. Ak sa napr. na vrchole zásobníka nachádza symbol Z, tak:

- operácia `push`*(A) sa zapíše ako postupnosť AZ,*

- operácia `skip` *sa zapíše symbolom Z,*

- operácia `pop` *sa zapíše symbolom ε.*

Zásobníkový automat A_1 spĺňa definíciu 4.2.1, je to teda deterministický zásobníkový automat. Základná myšlienka činnosti automatu A_1 spočíva v tom, že pri čítaní vstupného (páskového) symbolu a sa vykoná operácia `push`*(Z) a pri čítaní páskového symbolu b sa vykoná operácia* `pop`*. Po prečítaní všetkých symbolov a zo vstupného slova teda zásobník obsahuje rovnaký počet symbolov Z ako bol počet symbolov a. Na každý symbol b sa vykoná operácia* `pop`*. To znamená, že zásobník sa vyprázdni práve v okamihu, keď počet prečítaných symbolov b sa rovná počtu vstupných symbolov a v danom slove. Vtedy je potrebné zabezpečiť, aby automat A_1 prešiel do akceptujúceho stavu.*

Pre každý zásobníkový automat (avšak aj pre ľubovoľný konečný automat!) platí, že:

$$\varepsilon \in L(A) \;\Rightarrow\; q_0 \in F \;. \tag{4.3}$$

V prípade automatu A_1 to znamená, že jeho počiatočný stav q_0 musí byť aj jeho akceptujúcim stavom. Preto 1. riadok prechodovej funkcie zaručuje, že na prvý symbol vstupného slova sa riadiaca jednotka automatu A_1 preklopí do neakceptujúceho stavu, konkrétne q_1. (V neakceptujúcich stavoch potom prebieha celý nasledujúci výpočet automatu.) Uvedomme si, že keby 1. riadok prechodovej funkcie toto preklopenie neobsahoval, bolo by možné akceptovať aj všetky slová typu a^i $(i > 0)$. Výpočet na takomto slove by totiž končil v konfigurácii $(q_0, \varepsilon, Z^i Z_0)$, ktorá by sa považovala za akceptujúcu! Bolo by to preto, lebo zásobníkový automat je definovaný ako akceptujúci svojím koncovým stavom, takže pri akceptovaní nie je potrebné mať zásobník vyprázdnený. V 2. riadku prechodovej funkcie sa vykonáva operáciu `push`*(Z) v stave q_1, ak sa na vstupe vyskytuje symbol a. Na prvý prečítaný symbol b je potrebné vykonať operáciu* `pop` *a súčasne preklopiť riadiacu jednotku do stavu q_2 (3. riadok prechodovej funkcie). Uvedomme si, že absencia preklopenia stavu v tomto riadku by zaručovala aj akceptovanie takých slov, ako napr. aababb, aabaabbb, a pod. (Tieto slová samozrejme nepatria do jazyka L_1.) 4. riadok prechodovej funkcie zaručuje, že v stave q_2 sa vykonáva operácia* `pop`*, ak čítacia hlava číta symbol b. 5. riadok prechodovej funkcie vyjadruje situáciu, že automat sa nachádza v stave q_2, na vstupe už nie je žiaden znak (čítacia hlava číta symbol ε) a zásobník je prázdny. V takomto* *prípade sa vykoná prázdna zásobníková operácia a automat prejde do akceptujúceho stavu* q_0. *Tým je jeho výpočet ukončený.*

Simulovanie jednotlivých výpočtov automatu A_1 *pre vstupné slová* ε, *ab*, *aabb je vyjadrené nasledujúcimi postupnosťami krokov:*

$$(q_0, \varepsilon, Z_0) \ ,$$

$$(q_0, ab, Z_0) \vdash (q_1, b, ZZ_0) \vdash (q_2, \varepsilon, Z_0) \vdash (q_3, \varepsilon, Z_0) \ ,$$

$$(q_0, aabb, Z_0) \vdash (q_1, abb, ZZ_0) \vdash (q_1, bb, ZZZ_0) \vdash (q_2, b, ZZ_0) \vdash (q_2, \varepsilon, Z_0) \vdash (q_3, \varepsilon, Z_0) \ .$$

Všetky uvedené slová sú prvkami jazyka L_1, *preto jednotlivé výpočty končia v jednej z akceptujúcich konfigurácii* (q_0, ε, Z_0) *alebo* (q_3, ε, Z_0).

Dodajme, že napr. pre vstupné slovo aaa $\notin L_1$ *sa výpočet automatu* A_1 *skončí v konfigurácii* $(q_1, \varepsilon, ZZZZ_0)$, *ktorá podľa definície 4.1.1 nie je akceptujúca. Podobne, pre slovo aababb* $\notin L_1$ *automat skončí svoj výpočet v konfigurácii* (q_2, abb, ZZ_0), *ktorá tiež nie je akcepujúca.*

V nasledujúcich príkladoch ukážeme niektoré techniky, ktoré sa používajú pri konštrukcii zásobníkových automatov.

Príklad 4.2.2 *Metóda viacerých zásobníkových symbolov. Deterministický zásobníkový automat* A_2 *rozpoznáva jazyk* $L_2 = \{w0w^R \mid w \in \{1,2\}^*\}$.
Nech $A_2 = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, *kde:*
$K = \{q_0, q_1, q_2\}$,
$\Sigma = \{0, 1, 2\}$,
$\Gamma = \{Z_0, Z_1, Z_2\}$,
$F = \{q_2\}$,

$\delta(q_0, 1, Z) = (q_0, Z_1Z)$,	$Z \in \Gamma$	`push`(Z_1)
$\delta(q_0, 2, Z) = (q_0, Z_2Z)$,	$Z \in \Gamma$	`push`(Z_2)
$\delta(q_0, 0, Z) = (q_1, Z)$,	$Z \in \Gamma$	`skip`
$\delta(q_1, 1, Z_1) = (q_1, \varepsilon)$		`pop`, *ak* `top`$=Z_1$
$\delta(q_1, 2, Z_2) = (q_1, \varepsilon)$		`pop`, *ak* `top`$=Z_2$
$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$		*akceptovanie.*

Simulovanie výpočtu automatu A_2 *na vstupnom slove* $1220221 \in L_2$:

$$(q_0, 1220221, Z_0) \vdash (q_0, 220221, Z_1Z_0) \vdash (q_0, 20221, Z_2Z_1Z_0) \vdash$$

$$\vdash (q_0, 0221, Z_2Z_2Z_1Z_0) \vdash (q_1, 221, Z_2Z_2Z_1Z_0) \vdash (q_1, 21, Z_2Z_1Z_0) \vdash$$

$$\vdash (q_1, 1, Z_1Z_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_2, \varepsilon, Z_0) \ .$$

Zásobníkový symbol Z_1 *môžeme považovať za "obraz" vstupného symbolu* 1 *a symbol* Z_2 *zase za "obraz" vstupného symbolu* 2. *V okamihu, keď sa čítacia hlava nachádza na pozícii páskového symbolu* 0, *obsah zásobníka presne zodpovedá zrkadlovému* *obrazu prvej časti daného vstupného slova (t.j. podslova pred symbolom* 0*). Ak vstupné slovo patrí do jazyka* L_2*, musí sa aktuálny obsah zásobníka zhodovať s podslovom, ktoré nasleduje za symbolom* 0*. Kontrola tejto zhody sa vykonáva v 4. a 5. riadku prechodovej funkcie tak, že operácia* `pop` *sa vykoná vtedy a len vtedy, ak platí:*

- *na vstupe je symbol* $1 \ \wedge$ `top`$= Z_1$*, alebo*
- *na vstupe je symbol* $2 \ \wedge$ `top`$= Z_2$*.*

V 3. riadku prechodovej funkcie je potrebné preklopiť riadiacu jednotku zo stavu q_0 *do stavu* q_1*. (Kvôli lepšiemu oboznámeniu sa s činnosťou zásobníkového automatu odporúčame simulovať jeho výpočet pre rôzne iné vstupné slová.)*

Aplikovanie prázdnej zásobníkovej operácie `skip` má širšie využitie. Ako príklad môže slúžiť DPDA, ktorý rozpoznáva jazyk

$$L_3 = \{\ uv \mid u \in \{a,c\}^*, v \in \{b,c\}^*, \#_a u = \#_b v\ \}\ .$$

Jeho definíciu písať nebudeme, avšak základný princíp konštrukcie spočíva v tom, že na každý vstupný symbol sa musí vykonávať iná zásoníková operácia. Konkrétne:

- ak je na vstupe symbol a, vykonáva sa operácia `push`,
- ak je na vstupe symbol b, vykonáva sa operácia `pop`,
- ak je na vstupe symbol c, vykonáva sa operácia `skip`.

Z definície deterministického zásobníkového automatu je zrejmé, že na jeden krok výpočtu (resp. posun hlavy o jeden symbol) môžeme vykonať iba jednu zásobníkovú operáciu. Znamená to, že nie je možné naraz uložiť do zásobníka viac ako jeden symbol. Podobne, na jeden krok výpočtu je možné zo zásobníka vybrať najviac jeden symbol. V nasledujúcom príklade použijeme operáciu `skip`, aby sme zabránili dvojnásobnému vyberaniu.

Príklad 4.2.3 *Deterministický zásobníkový automat* A_4 *rozpoznáva jazyk*
$L_4 = \{c^{2n}b^n \mid n \in \mathbb{N}^+\}$.
Nech $A_4 = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$*, kde:*
$K = \{q_0, q_1, q_2, q_3\}$,
$\Sigma = \{b, c\}$,
$\Gamma = \{Z_0, Z\}$,
$F = \{q_3\}$,

$\delta(q_0, c, X) = (q_1, X)$,	$X \in \Gamma$	`skip`
$\delta(q_1, c, X) = (q_0, ZX)$,	$X \in \Gamma$	`push`
$\delta(q_0, b, Z) = (q_2, \varepsilon)$		`pop`

$\delta(q_2, b, Z) = (q_2, \varepsilon)$ `pop`
$\delta(q_2, \varepsilon, Z_0) = (q_3, Z_0)$ *akceptovanie.*

Princíp činnosti zásobníkového automatu A_4 spočíva v tom, že operácia `push` *sa vykonáva len pre každý párny načítaný symbol c, zatiaľčo operácia* `pop` *sa vykonáva pri čítaní každého symbolu b. Podslovo c^{2k} sa rozpoznáva v stavoch q_0 a q_1, ako keby to bol konečný automat rozpoznávajúci slová s párnou dĺžkou. Na každý nepárny vstupný symbol c sa vykoná operácia* `skip` *a na každý párny vstupný symbol c sa vykoná operácia* `push`*. Preto platí, že po prečítaní vstupu c^{2k} zásobník obsahuje práve k symbolov Z ($k \in \mathbb{N}$). Potom už len stačí skontrolovať, či sa počet symbolov v zásobníku zhoduje s počtom symbolov b zo vstupu. Poznamenajme ešte, že v tomto prípade $q_0 \notin F$ keďže $\varepsilon \notin L_4$.*

Príklad 4.2.4 *Deterministický zásobníkový automat A_5 rozpoznáva jazyk*
$L_5 = \{w \in \{a, b\}^* \mid \#_a w = \#_b w\}$.
Nech $A_5 = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, *kde:*
$K = \{q_0, q_1\}$,
$\Sigma = \{a, b\}$,
$\Gamma = \{Z_0, A, B\}$,
$F = \{q_0\}$,
$\delta(q_0, a, Z_0) = (q_1, AZ_0)$ `push`(A)
$\delta(q_0, b, Z_0) = (q_1, BZ_0)$ `push`(B)
$\delta(q_1, a, A) = (q_1, AA)$ `push`(A)
$\delta(q_1, b, B) = (q_1, BB)$ `push`(B)
$\delta(q_1, a, B) = (q_1, \varepsilon)$ `pop`
$\delta(q_1, b, A) = (q_1, \varepsilon)$ `pop`
$\delta(q_1, \varepsilon, Z_0) = (q_0, Z_0)$ *akceptovanie.*

Tento zásobníkový automat je zložitejší, akoby sa mohlo zdať na prvý pohľad. Uvedomme si, že do jazyka L_5 patria napríklad nasledujúce slová: aababb, baab, abbaba. Preto nie je možné stanoviť jednoduché pravidlo na pridávanie, resp. vyberanie zo zásobníka. Riešenie spočíva v tom, že okrem symbolu Z_0 použijeme dva ďalšie zásobníkové symboly. Platí, že v zásobníku sa nemôžu súčasné nachádzať aj symboly A aj symboly B.

Ak sa v zásobníku nachádzajú iba samé symboly A, znamená to, že vstupné slovo sa začínalo symbolom a. V takomto prípade sa na každý páskový symbol a vykonáva operácia `push`(A) *a na každý páskový symbol b sa vykonáva operácia* `pop`*. Duálny prípad nastáva, ak sa vstupné slovo začínalo symbolom b. Vtedy sa do zásobníka pridávajú symboly B, práve vtedy, ak čítacia hlava číta symbol b. Operácia* `pop` *sa teraz použije, ak sa na vstupe nachádza symbol a. Ako príklad uvádzame výpočet na slove bbabaa.*

$$(q_0, bbabaa, Z_0) \vdash (q_1, babaa, BZ_0) \vdash (q_1, abaa, BBZ_0) \vdash (q_1, baa, BZ_0) \vdash$$

$$\vdash (q_1, aa, BBZ_0) \vdash (q_1, a, BZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_0, \varepsilon, Z_0)$$

Najkomplikovanejší prípad nastane, ak sa v priebehu výpočtu zásobník vyprázdni. Vtedy sa môže stať, že je potrebné zmeniť druh zásobníkového symbolu. Takáto situácia nastane napr. pre vstupné slovo abbbaa. Všimnime si, že najprv sa do zásobníka vkladá symbol A, ale už pri treťom vstupnom symbole tohto slova je potrebné mať v zásobníku symbol B. Uvedenú výmenu zásoníkového symbolu rieši posledný riadok prechodovej funkcie. Podľa neho sa "epsilonovým" krokom dokážeme preklopiť zo stavu q_1 do stavu q_0. (Stav q_0 je súčasne akceptačným ale aj počiatočným stavom.) Keďže definícia 4.2.1 povoľuje isté druhy "epsilonových" krokov, je zásobníkový automat A_5 deterministický. Uvádzame výpočet na slove abbbaa.

$$(q_0, abbbaa, Z_0) \vdash (q_1, bbbaa, AZ_0) \vdash (q_1, bbaa, Z_0) \vdash (q_0, bbaa, Z_0) \vdash$$

$$\vdash (q_1, baa, BZ_0) \vdash (q_1, aa, BBZ_0) \vdash (q_1, a, BZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_0, \varepsilon, Z_0)$$

V konfigurácii $(q_1, bbaa, Z_0)$ máme jedinú možnosť, ktorou sa dá pokračovať vo výpočte. Použitím posledného riadku prechodovej funkcie prejdeme deterministicky (!) do konfigurácie $(q_0, bbaa, Z_0)$, pričom čítacia hlava automatu zostane na svojom pôvodnom mieste. Z tejto konfigurácie je ďalej možné pokračovať použitím druhého riadku prechodovej funkcie. Ďalej už výpočet pokračuje priamočiaro.

Poznámka 4.2.1 *Existuje ešte aj iný spôsob, ktorým sa dá rozpoznávať jazyk L_5. Príslušný deterministický zásobníkový automat používa okrem zásobníkového symbolu Z_0 už iba jeden ďalší zásobníkový symbol, avšak riadiaca jednotka tohto automatu musí obsahovať až tri stavy.*

Nedeterministické zásobníkové automaty

Príklad 4.2.5 *Nedeterministický zásobníkový automat A_N rozpoznáva jazyk $L_N = \{ww^R \mid w \in \{a,b\}^+\}$.*
Nech $A_N = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde:
$K = \{q_0, q_1\}$,
$\Sigma = \{a, b\}$,
$\Gamma = \{Z_0, A, B\}$,
$F = \{q_2\}$,

$\delta(q_0, a, Z_0) = (q_0, AZ_0)$	`push`*(A)*
$\delta(q_0, b, Z_0) = (q_0, BZ_0)$	`push`*(B)*
$\delta(q_0, a, A) = \{\ (q_0, AA),\ (q_1, \varepsilon)\ \}$	*nedeterministický krok*
$\delta(q_0, b, B) = \{\ (q_0, BB),\ (q_1, \varepsilon)\ \}$	*nedeterministický krok*
$\delta(q_0, a, B) = (q_0, AB)$	`push`*(A)*
$\delta(q_0, b, A) = (q_0, BA)$	`push`*(B)*
$\delta(q_1, a, A) = (q_1, \varepsilon)$	`pop`
$\delta(q_1, b, B) = (q_1, \varepsilon)$	`pop`
$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$	*akceptovanie.*

Princíp rozpoznávania jazyka L_N je podobný ako pre jazyk
$L_2 = \{w0w^R \mid w \in \{1,2\}^*\}$ z príkladu 4.2.2, je tu však jeden podstatný rozdiel. Nie je

totiž možné jednoduchým spôsobom určit', kde sa nachádza stred slova ww^R*. Netriviálnym spôsobom sa dá dokonca dokázat', že stred takýchto slov sa nedá určit' deterministicky pomocou zásobníkového automatu. Zásobníkový automat* A_N *preto musí byt' nedeterministický. Ak sa čítacia hlava nachádza na pozícii v prvej polovici slova* ww^R*, bude sa vykonávat' operácia* `push`*. Ak sa čítacia hlava nachádza na pozícii v podslove* w^R*, bude sa nedeterministicky vykonávat' operácia* `pop`*, pričom sa kontroluje zhodnost' symbolov zrkadlového obrazu. Zásobníkový automat dokáže nedeterministicky určit' stred slova, nedeterministický výber krokov s operáciami* `push` *alebo* `pop` *obsahujú prechodové funkcie v 3. a 4. riadku. Uvádzame príklad výpočtu automatu* A_N *na vstupnom slove* $aabbaa \in L_N$*.*

$$(q_0, aabbaa, Z_0) \vdash (q_0, abbaa, AZ_0) \vdash (q_0, bbaa, AAZ_0) \vdash (q_0, baa, BAAZ_0) \vdash$$

$$\vdash (q_1, aa, AAZ_0) \vdash (q_1, a, AZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_2, \varepsilon, Z_0)$$

Nedeterministický výber krokov nastáva spolu dvakrát, a to pre konfigurácie $(q_0, abbaa, AZ_0)$ *a* (q_0, baa, BAA, Z_0)*. Dôležitou vlastnost'ou každého nedeterministického automatu (nielen zásobníkového) je skutočnost', že v prípade nesprávneho vstupného slova (t.j. slova nepatriaceho do jazyka rozpoznávaného príslušným nedeterminisitickým automatom), akákol'vek postupnost' krokov vo výpočte daného automatu na tomto slove nesmie končit' v akceptujúcej konfigurácii. Preto akonáhle automat* A_N *nedeterministicky už raz uskutoční operáciu* `pop`*, musí sa preklopit' zo stavu* q_0 *do stavu* q_1*. Tým je zaručené, že vo všetkých nasledujúcich krokoch výpočtu sa môže vykonávat' už iba operácia* `pop`*. (V stave* q_1 *je totiž akákol'vek iná operácia ako* `pop` *neprípustná.) Poznamenajme, že keby spomínané preklopenie stavu automat* A_N *neobsahoval, bolo by možné pomocou neho akceptovat' aj iné slová, napr.* $baabaa \notin L_N$*. L'ahko sa však presvedčíme, že neexistuje žiaden akceptujúci výpočet automatu* A_N *pre slovo* $baabaa$*.*

Poznámka 4.2.2 *Alternatívny a ekvivalentný spôsob pre rozpoznávanie jazyka* L_N *by predstavovalo nahradenie 3. a 4. riadku prechodovej funkcie automatu* A_N *novou prechodovou funkciou:*

$$\delta(q_0, \varepsilon, X) = (q_1, X) \ , \qquad X \in \{A, B\}$$

a doplnenie prechodových funkcií:

$$\delta(q_0, a, A) = (q_0, AA) \ ,$$

$$\delta(q_0, b, B) = (q_0, BB) \ .$$

Overte správnost' takéhoto automatu a podl'a definície dokážte, že nie je deterministický.

Cvičenia

Cvičenie 4.2.1 *Sú dané gramatiky G_1 a G_2 v Greibachovej normálnom tvare. Definujte zásobníkové automaty A_1 a A_2 tak, aby platilo $L(A_1) = L(G_1)$ a $L(A_2) = L(G_2)$. Zistite a dokážte, či sú skonštruované automaty determinisitické alebo nedeterministické.*

a) $G_1 = (N, T, P, S)$, *kde* $N = \{S, A, B, C\}$, $T = \{a, b, +\}$ *a* P *sú:*

$S \to aACA \mid a \mid b$

$A \to aABAS \mid aBB \mid b \mid bB$

$B \to +SC \mid aAA \mid a \mid bB$

$C \to +$

b) $G_2 = (N, T, P, S)$, *kde* $N = \{S, A, B, C\}$, $T = \{3, 6\}$ *a* P *sú:*

$S \to 6 \mid 6ACA \mid 6S$

$A \to 6ABAS \mid 6BB \mid 3 \mid 6A$

$B \to 3CS \mid 3BB \mid 6 \mid 6A$

$C \to 3$

Cvičenie 4.2.2 *Sú dané nasledujúce jazyky.*

$L_1 = \{ww^R \mid w \in \{a, b, c\}^*\}$

$L_2 = \{2^{2n}3^n \mid n \in \mathbb{N}\}$

$L_3 = \{0a^n1b^n2 \mid n \in \mathbb{N}\}$

Napíšte gramatiky, ktoré definujú nasledujúce jazyky.

a) $L_1 \cup L_2$

b) $(L_1 \cup L_3)^2$

c) $(L_2 \cup L_3)^3$

d) $L_1 \cdot L_2$

e) $L_3 \cdot L_1$

f) $L_2 \cdot (L_3)^2$

g) L_1^*

h) L_3^+

Cvičenie 4.2.3 *Dokážte, že trieda $\mathcal{L}_{CF}$ nie je uzavretá vzhľadom na operáciu prieniku. Návod. Nech $L_{CS} = \{a^nb^nc^n \mid n \in \mathbb{N}\}$. Platí, že $L_{CS} \in \mathcal{L}_{CS}$. Zostrojte bezkontextové jazyky L_1, L_2 tak, aby $L_1 \cap L_2 = L_{CS}$.*

Cvičenie 4.2.4 *Zistite a zdôvodnite, či je trieda $\mathcal{L}_{CF}$ uzavretá vzhľadom na operácie:*

- *kladného uzáveru* $(^+)$,

- *zrkadlového obrazu* (^{R}).

Cvičenie 4.2.5 *Je daný deterministický zásobníkový automat* $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, *kde:*
$K = \{q_0, q_1, q_2\}$,
$\Sigma = \{b, d\}$,
$\Gamma = \{Z, Z_0\}$,
$F = \{q_2\}$,
$\delta(q_0, b, Z_0) = (q_0, ZZ_0)$
$\delta(q_0, d, Z) = (q_1, \varepsilon)$
$\delta(q_1, d, Z) = (q_1, \varepsilon)$
$\delta(q_1, \varepsilon, Z_0) = (q_2, Z_0)$.
Zistite, čomu sa rovná $L(A)$.

Cvičenie 4.2.6 *Definujte deterministické zásobníkové automaty, ktoré rozpoznávajú nasledujúce jazyky.*
a) $L_1 = \{c^n d^n \mid n \in \mathbb{N}\}$
b) $L_2 = \{c^n d^n \mid n \in \mathbb{N}^+\}$
c) $L_3 = \{a^n c b^n \mid n \in \mathbb{N}\}$
d) $L_4 = \{a^n c b^n \mid n \in \mathbb{N}^+\}$
e) $L_5 = \{0a^n 1 b^n 2 \mid n \in \mathbb{N}\}$
f) $L_6 = \{0a^n 12 b^n 34 \mid n \in \mathbb{N}\}$

Cvičenie 4.2.7 *Definujte deterministické zásobníkové automaty, ktoré rozpoznávajú nasledujúce jazyky.*
a) $L_1 = \{wcw^R \mid w \in \{a, b\}^*\}$
b) $L_2 = \{wcw^R \mid w \in \{a, b\}^+\}$
c) $L_3 = \{w0w^R \mid w \in \{a, b, c\}^*\}$
d) $L_4 = \{w0w^R \mid w \in \{a, b, c, d\}^*\}$
e) $L_5 = \{1w2w^R3 \mid w \in \{b, c\}^*\}$
f) $L_6 = \{w101w^R \mid w \in \{a, b, c\}^*\}$
g) $L_7 = \{wcw^R \mid w \in \{1, 2, 3, 4\}^*\}$
h) $L_8 = \{awbw^R c \mid w \in \{1, 2, 3, 4, 5, 6\}^*\}$

Cvičenie 4.2.8 *Definujte deterministické zásobníkové automaty, ktoré rozpoznávajú nasledujúce jazyky.*
a) $L_1 = \{uc^n \mid u \in \{a, b\}^*,\ |u| = n,\ n \in \mathbb{N}\}$
b) $L_2 = \{uc^n \mid u \in \{a, b\}^*,\ \#_a u = n,\ n \in \mathbb{N}\}$
c) $L_3 = \{a^n w \mid w \in \{b, c\}^*,\ |w| = n,\ n \in \mathbb{N}\}$

d) $L_4 = \{a^n w \mid w \in \{b,c,d\}^*,\ \#_c u = n,\ n \in \mathbb{N}\}$

e) $L_5 = \{uv \mid u \in \{a,b\}^*, v \in \{c,d\}^*,\ |u| = |v|\}$

f) $L_6 = \{uv \mid u \in \{a,b,c\}^*, v \in \{0,1\}^*,\ |u| = |v|\}$

g) $L_7 = \{uv \mid u \in \{a,b\}^*, v \in \{c,d\}^*,\ \#_a u = \#_b v\}$

h) $L_8 = \{uv \mid u \in \{a,b\}^+, v \in \{1,2,3\}^+,\ \#_a u = \#_3 v\}$

Cvičenie 4.2.9 *Jednou z úloh, ktorú vykonávajú kompilátory programovacích jazykov, je kontrolovanie správnosti výrazov so zátvorkami. V takýchto výrazoch sa vyskytuje rovnaký počet pravých a l'avých zátvoriek, pričom každá l'avá zátvorka sa nachádza pred príslušnou pravou zátvorkou.*
Nasledujúci jazyk je zjednodušeným príkladom výrazov so zátvorkami.

$$L = \{uv \mid u \in \{(,0\}^*, v \in \{),0\}^*,\ \#_(u = \#_) v\}$$

a) Definujte zásobníkový automat, ktorý rozpoznáva uvedený jazyk.

b) Definujte d'alšie jazyky, v ktoré spĺňajú podmienky pre výrazy so zátvorkami.

Cvičenie 4.2.10 *Definujte deterministické zásobníkové automaty, ktoré rozpoznávajú nasledujúce jazyky.*

a) $L_1 = \{u0v \mid u \in \{1,3\}^*, v \in \{2,3\}^*,\ \#_1 u = \#_2 v\}$

b) $L_2 = \{uv \mid u \in \{a,b,c\}^*, v \in \{0,1\}^*,\ |u| = \#_1 v\}$

c) $L_3 = \{uv \mid u \in \{a,b\}^*, v \in \{c,d\}^*,\ \#_b u = |v|\}$

d) $L_4 = \{uv \mid u \in \{a,b,c\}^*, v \in \{1,2,3\}^*,\ \#_a u + \#_c u = \#_2 v\}$

e) $L_5 = \{uv \mid u \in \{a,b,c\}^*, v \in \{1,2,3\}^*,\ |u| = \#_1 v + \#_3 v\}$

f) $L_6 = \{u0v \mid u \in \{a,b,c,d\}^*, v \in \{1,2,3,4\}^*,\ \#_b u + \#_d u = \#_1 v + \#_4 v\}$

Cvičenie 4.2.11 *Definujte deterministické zásobníkové automaty, ktoré rozpoznávajú nasledujúce jazyky.*

a) $L_1 = \{b^n c^{2n} \mid n \in \mathbb{N}\}$

b) $L_2 = \{b^n d^{3n} \mid n \in \mathbb{N}^+\}$

c) $L_3 = \{a^{3n} c^n \mid n \in \mathbb{N}\}$

d) $L_4 = \{c^{3n} b^{2n} \mid n \in \mathbb{N}\}$

e) $L_5 = \{(ac)^n b^n \mid n \in \mathbb{N}\}$

f) $L_6 = \{4^{2n+1} 6^{4n+2} \mid n \in \mathbb{N}\}$

g) $L_7 = \{b^{2n} a c^{3n} \mid n \in \mathbb{N}\}$

h) $L_8 = \{0^{4n} ab 1^{3n} \mid n \in \mathbb{N}\}$

Cvičenie 4.2.12 *Nech $\alpha, \beta \in \mathbb{N}^+$ sú parametre, $\alpha \neq \beta$. Definujte deterministické zásobníkové automaty, ktoré rozpoznávajú nasledujúce jazyky.*

a) $L_1 = \{a^{\alpha n}b^{\beta n} \mid n \in \mathbb{N}\}$
b) $L_2 = \{a^{\alpha n}b^{\beta n} \mid n \in \mathbb{N}^+\}$
c) $L_3 = \{a^{\alpha n+2}b^{\beta n+1} \mid n \in \mathbb{N}\} \quad (\alpha > 2,\ \beta > 1)$
d) $L_4 = \{wc^n \mid w \in \{a,b\}^*,\ \alpha \cdot |w| = n,\ n \in \mathbb{N}\}$
e) $L_5 = \{b^n w \mid w \in \{1,2,3\}^*,\ \alpha \cdot n = \beta \cdot |w|,\ n \in \mathbb{N}\}$
f) $L_6 = \{uv \mid u \in \{a,b\}^*, v \in \{c,d\}^*,\ \alpha \cdot |u| = |v|\}$
g) $L_7 = \{uv \mid u \in \{a,b\}^*, v \in \{2,3\}^*,\ |u| = \beta \cdot |v|\}$
h) $L_8 = \{uv \mid u \in \{a,b,c\}^*, v \in \{1,2\}^*,\ \alpha \cdot |u| = \beta \cdot |v|\}$

Cvičenie 4.2.13 *Definujte deterministické zásobníkové automaty, ktoré rozpoznávajú nasledujúce jazyky.*
a) $L_1 = \{w \in \{a,b,c\}^* \mid \#_a w = \#_c w\}$
b) $L_2 = \{w \in \{1,2,3,4,5,6\}^* \mid \#_1 w + \#_3 w = \#_5 w + \#_6 w\}$

Cvičenie 4.2.14 *Nasledujúce jazyky patria do triedy $\mathcal{L}(NPDA) \backslash \mathcal{L}(DPDA)$. Definujte nedeterministické zásobníkové automaty, ktoré rozpoznávajú uvedené jazyky.*
a) $L_1 = \{ww^R \mid w \in \{1,2,3,4\}^+\}$
b) $L_2 = \{uavbw \mid u,v,w \in \{a,b\}^*,\ |v| = |u| + |w|\}$

Cvičenie 4.2.15 *Detreministický konečný automat s párnym celočíselným počítadlom je šestica $A_p = (K, \Sigma, \delta, P, q_0, F)$, kde všetky prvky okrem P sú tie isté ako pri deterministickom konečnom automate a symbol P reprezentuje počítadlo. Na začiatku výpočtu platí, že $P = 0$ a počas výpočtu môže P nadobúdať ľubovoľnú hodnotu párneho celého čísla (teda aj zápornú). Nad počítadlom P sú definované 3 operácie:*

- `ADD2` - *pripočítanie čísla* 2,
- `SUB2` - *odpočítanie čísla* 2,
- `SKIP` - *prázdna operácia.*

Pre automat A_p definujte jeho konfiguráciu, krok výpočtu a jazyk, ktorý rozpoznáva.

Cvičenie 4.2.16 *Detreministický konečný automat s poľom je sedemtica $A_Y = (K, \Sigma, \Gamma, \delta, Y, q_0, F)$, kde všetky prvky okrem Γ a Y sú tie isté ako pri deterministickom konečnom automate. Symbol Γ reprezentuje abecedu symbolov, ktoré sú uchovávané v poli X. Pole sa indexuje jedným indexom $j \in \mathbb{N}$ tak, že môže mať nekonečne veľa prvkov. (T.j. rozsah indexu j nie je potrebné kontrolovať.) Ľavá strana prechodovej funkcie je daná ako (q, a, x, j), pričom x je symbol, ktorý sa nachádza na j-tom mieste poľa Y. Nad poľom Y sú definované 2 operácie:*

- `WRITE`(α, k) - *zápis symbolu α na k-te miesto poľa,*

- `SKIP` - *prázdna operácia.*

Pre automat A_X definujte jeho konfiguráciu, krok výpočtu a jazyk, ktorý rozpoznáva.

Kapitola 5

Vlastnosti jazykov v Chomského hierarchii

Rozpoznávacia sila automatov a Chomského hierarchia

Automat	Determin. verzia		Nedetermin. verzia		Trieda jazykov
konečný	$\mathcal{L}(DFA)$	$=$	$\mathcal{L}(NFA)$	$=$	$\mathcal{R}$
zásobníkový	$\mathcal{L}(DPDA)$	$\subsetneq$	$\mathcal{L}(NPDA)$	$=$	$\mathcal{L}_{CF}$
lin. ohraničený	$\mathcal{L}(DLBA)$	$\subseteq^1$	$\mathcal{L}(NLBA)$	$=$	$\mathcal{L}_{CS}$
Turingov stroj	$\mathcal{L}(DTM)$	$=$	$\mathcal{L}(NTM)$	$=$	$\mathcal{L}_{RE}$

Tabuľka 5.1: Vzťah automatov a jazykov z Chomského hierarachie.

Uzávarové vlastnosti tried jazykov z Chomského hierarachie

Trieda jazykov	$\cup$	$\cdot$	$\cap$	*	$^+$	^{c}	^{R}	h	h bez ϵ
$\mathcal{R}$	A	A	A	A	A	A	A	A	A
$\mathcal{L}_{CF}$	A	A	N	A	A	N	A	A	A
$\mathcal{L}_{CS}$	A	A	A		A	A	A	N	A
$\mathcal{L}_{RE}$	A	A	A	A	A	N	A	A	A

Tabuľka 5.2: Uzáverové vlastnosti.

Kapitola 6

Vypočítateľnosť

Teória vypočítateľnosti skúma otázky algoritmizovateľnosti na rôznych výpočtových modeloch. Pojem vzpočítateľného problému sa používa ako synonimum pre algoritmizovateľný problém. Exaktné vymedzenie samotného pojmu algoritmus je však vysoko netriviálne. Kvôli tomu sa v teórii, ktorá sa zaoberá vypočítateľnosťou skúmajú rôznorodé problémy, z ktorých najdôležitejšie sú nasledujúce:

- algoritmus a jeho vlastnosti,
- problémy, ktoré sa dajú algoritmizovať a tie, ktoré sa algoritmizovať nedajú.

Kľúčom k riešeniu týchto otázok sú rôzne výpočtové modely, ktoré vznikli kvôli tomu, aby čo možno najvernejšie reprezetovali pojmy algoritmus a počítač. Ich vlastnoti a vzájomné vzťahy (ekvivalencia) dokážu v značnej miere odpovedať na uvedené otázky.

V tejto kapitole sa budeme zaoberať tromi vzájomne ekvivaletnými výpočtovými modelmi. Sú to nasledujúce modely:

- Turingov stroj,
- počítadlový stroj,
- stroj RAM.

6.1 Turingove stroje

Dôležitá vlastnosť Turingových strojov je tzv. univerzalita. Znamená to, že sa dajú zostrojiť Turingove stroje, na ktorých je možné simulovať výpočet iných Turingových strojov.

Definícia 6.1.1 Univerzálny Turingov stroj

Nech Σ je abeceda a nech T_x je Turingov stroj, ktorý je repreznetovaný svojim kódom $x \in \Sigma^*$. Univerzálny Turingov stroj *U pre všetky vstupy y, pre ktoré je výstup $T_x(y)$ definovaný, počíta hodnotu*

$$U(x, y) = T_x(y)$$

pre všetky prípustné kódy $x \in \Sigma^*$.

Veta 6.1.1 *Existuje univerzálny Turingov stroj.*

Turingov stroj sa používa aj na formalizáciu pojmu T-vypočítateľná funkcia.

Definícia 6.1.2 T-vypočítateľná funkcia
Nech $k \in \mathbb{N}^+$. Funkcia $f : \mathbb{N}^k \to \mathbb{N}$ sa nazýva T-vypočítateľná funkcia*, ak existuje Turingov stroj A, ktorý rozpoznáva jazyk*

$$L = \{\ 1^{x_1}¥1^{x_2}¥\dots¥1^{x_k}\$1^{f(\overline{x})}\ \}$$

pre všetky $\overline{x} = (x_1, \dots, x_k) \in \mathbb{N}^k$.

Príklady T-vypočítateľných funkcií:

- lineárne funkcie,
- polynomiálne funkcie,
- exponenciálne funkcie,
- logaritmické funkcie,
- lineárne kombinácie T-vypočítetľných funkcí.

6.2 Počítadlové stroje

Počítadlový stroj (Abacus machine, abacus = počítadlo) je jedným z najjednoduchších výpočtových modelov, ktorý je postavený na operáciách inkrementovania a dektrementovania. Syntakticky správny počítadlový stroj je možné skonštruovať na základe rekurzívnej definície 6.2.1. Sémantika jednotlivých konštrukčných krokov, ako aj popis činnosti počítadlového stroja budú popísané v ďalšej sekcii.

Definícia 6.2.1 Rekurzívna definícia počítadlového stroja
Definícia pozostáva z troch častí:

- *a_k a s_k sú počítadlové stroje, pričom $k \in \mathbb{N}$,*

- *ak $M1$, $M2 \dots Mn$ sú počítadlové stroje, tak aj $M1M2 \dots Mn$ je počítadlový stroj,*

- *ak M je počítadlový stroj, tak aj $(M)_k$ je počítadlový stroj, pričom $k \in \mathbb{N}$,*

nič iné nie je počítadlový stroj.

Poznámka 6.2.1 *Definícia 6.2.1 určuje, aký zápis počítadlového stroja je možné považovať za syntakticky správny. Napríklad tieto štyri stroje oddelené čiarkami: s_5 , a_0, $s_4s_8a_{10}$, $(a_3)_2(s_2a_4)_4$ sú zo syntaktického hľadiska správne (sú to počítadlové stroje). Takto skonštruované počítadlové stroje nemusia byť zo sémantického hľadiska zmysluplné, čiže nemusia vykonávať zmysluplné výpočty.*

6.2.1 Neformálny opis výpočtu počítadlového stroja

Počítadlový stroj pracuje s konečným počtom registrov Ri, kde index $i \in \mathbb{N}$, počet registrov je zvolený podľa potreby. V registroch môžu byť uložené ľubovoľne veľké prirodzené čísla (nezáporné celé čísla). Sémantika jednotlivých častí rekurzívnej definície je nasledovná:

- Zápis a_i po sémantickej stránke znamená inkrementovanie registra Ri („a“ ako addition), čiže stroj a_i vykoná operáciu $Ri = Ri + 1$, čím výpočet končí. Zápis s_i dekrementuje registrer Ri („s“ ako subtraction), ak bol tento nenulový. Čiže stroj s_i vykoná $Ri = Ri \ominus 1$ a jeho výpočet skončí. Operácia $x \ominus y$ je definovaná ako:
$$x \ominus y = \begin{cases} x - y & \text{ak } x > y \\ 0 & \text{inak} \end{cases} .$$

- Počítadlový stroj $M1M2 \dots Mn$, vytvorený zreťazením počítadlových strojov $M1$, $M2 \dots Mn$, vykoná opreácie zodpovedajúce stroju $M1$, potom stroju $M2$ a takto ďalej až po stroj Mn, potom výpočet končí.

- Zápisom $(M)_k$ sa realizuje cyklus. Najskôr sa otestuje register Rk, a ak je nenulový, vykoná sa operácia zodpovedajúca stroju M. Následne sa opätovne otestuje register Rk, a ak je nenulový znovu sa vykoná stroj M. Stroj M sa cyklicky vykonáva, až kým register Rk nenadobudne nulovú hodnotu, potom výpočet zodpovedajúci stroju $(M)_k$ končí.

Na začiatku výpočtu sú všetky registre nastavené na nulu. Do zvolených registrov sa zapíšu vstupné údaje a počítadlový stroj začne vykonávať operácie tak, ako bolo uvedené v predchádzajúcej časti. Po skončení výpočtu je v zvolených registroch výsledok výpočtu.

6.2.2 Riešené príklady

Atomickými operáciami, modifikujúcimi registre počítadlového stroja, sú operácie inkrementácie a dekrementácie. Vzhľadom na jednoduchosť týchto operácií je aj realizácia funkcií, ako napríklad $f(i,j) = i + j$, $f(i,j) = i \ominus j$, $f(i,j) = i \cdot j$, $f(i,j) = \lceil i/j \rceil$ pre $i, j \in \mathbb{N}$, pomerne netriviálna.

Príklad 6.2.1 *Pomocou počítadlového stroja realizujte funkciu $f(i,j) = i + j$, kde $i, j \in \mathbb{N}$. Ukážeme si dve rôzne riešenia.*
`Riešenie a.`
Vstupné registre: $R1 : a$, $R2 : b$
Výstupné registre: $R3 : f(a,b)$
Pomocné register: žiadny

$$(s_1a_3)_1(s_2a_3)_2$$

Do registra $R1$ a $R2$ sa umiestnia vstupné argumenty a počítadlový stroj začne vykonávať výpočet. Cyklus $(s_1a_3)_1$ sa vykoná $R1$–krát, pretože každé vykonanie počítadlového stroja v cycle (stroja s_1a_3) dekrementuje register $R1$. Zároveň tento stroj inkrementuje register $R3$, a teda po vykonaní stroja $(s_1a_3)_1$ je obsah registera $R3$ zväčšený o obsah registra $R1$ a register $R1$ nadobudol hodnotu 0. Pretože všetky registre okrem vstupných registrov boli vynulované, po vykonaní $(s_1a_3)_1$ je v $R3$ obsah registra $R1$. Stroj $(s_2a_3)_2$ obdobným spôsobom vykoná presun obsahu registra $R2$ do registra $R3$, a teda po vykonaní stroja je v $R3$ výsledok požadovanej operácie a vstupné registre $R1$ a $R2$ nadobudli hodnotu 0.
`Riešenie b.`
Stroje $(s_1a_3)_1$ a $(s_2a_3)_2$ v predchádzajúcom riešení realizovali tzv. „deštruktívne“ kopírovanie obsahu. Ak by sme chceli zachovať obsah vstupných registrov $R1$ a $R2$, je potrebné použiť tzv. „nedeštruktívne“ kopírovanie obsahu pomocou pomocného registra $R0$.
Vstupné registre: $R1 : a$, $R2 : b$
Výstupné registre: $R3 : f(a,b) = a + b$
Pomocné register: $R0$

$$(s_1a_0a_3)_1(s_0a_1)_0(s_2a_0a_3)_2(s_0a_2)_0$$

Princíp riešenia je rovnaký ako v predchádzajúcom riešení. Cyklus z riešenia (a) $(s_1a_3)_1$ je pozmenený na $(s_1s_0a_3)_1$, a teda obsah registra $R1$ je skopírovaný aj do registra $R0$ a môže byť následne presunutý späť do registra $R1$ v cycle $(s_0a_1)_0$. Po nedeštruktívnom presune obsahu registra $R1$ do registra $R3$ strojom $(s_1a_0a_3)_1(s_0a_1)_0$ je obsah registra $R1$ nezmenený. Pomocný register $R0$ je po skončení cyklu vynulovaný, a teda môže byť použitý pri ďalšom výpočte. Nedeštruktívne presunutie obsahu $R2$ do $R3$ je realizované rovnakým spôsobom.

Príklad 6.2.2 *Pomocou počítadlového stroja realizujte funkciu $f(i,j) = i \ominus j$, kde $i, j \in \mathbb{N}$.*
Vstupné registre: $R1 : j$, $R2 : j$

Výstupné registre: $R3 : f(a,b) = i \ominus j$
Pomocné register: $R0$

$$(s_1a_3)_1(s_2s_3)_2$$

Myšlienka riešenia je obdobná ako v príklade 6.2.1, iba namiesto pričítania obsahu registra $R2$ do registra $R3$ je jeho obsah z registra $R3$ odčítaný v cykle $(s_2s_3)_2$.

Príklad 6.2.3 *Pomocou počítadlového stroja realizujte funkciu $f(a,b) = a \cdot b$, kde $a, b \in \mathbb{N}$.*
Vstupné registre: $R1 : i$, $R2 : j$
Výstupné registre: $R3 : f(i,j) = i \cdot j$
Pomocné register: $R0$

$$(s_2(s_1a_0a_3)_1(s_0a_1)_0)_2$$

Stroj vo vnútri cyklu (stroj $(s_1a_0a_3)_1(s_0a_1)_0$) je už popísané nedeštruktívne pričítanie obsahu $R1$ do registra $R3$. Vonkajší cyklus $(s_2...)_2$ sa vykoná $R2$–krát, a teda po skončení výpočtu bude do $R3$ $R2$-krát pričítaný obsah registra $R1$.

Príklad 6.2.4 *Pomocou počítadlového stroja realizujte funkciu $f(a,b) = \lceil a/b \rceil$, kde $a, b \in \mathbb{N}$ a $b \neq 0$.*
Vstupné registre: $R1 : i$, $R2 : j$
Výstupné registre: $R3 : f(i,j) = \lceil i/j \rceil$
Pomocné register: $R0$

$$((s_2a_0s_1)_2(s_0a_2)_0a_3)_1$$

Riešenie spočíva v postupnom znižovaní obsahu registra $R1$ o hodnotu $R2$. Výsledkom stroja je každé aj začaté odpočítavanie, nakoľko nás zaujíma horná celá časť podielu a/b. Čiže pokiaľ je register $R1$ nenulový, a teda je ešte z čoho odpočítavať (vonkajší cyklus $(...)_1$), vykoná sa odpočítanie $R2$ od registra $R1$ $((s_2a_0s_1)_2$, obnoví sa register $R2$ z pomocného $R0$ (stroj $(s_0a_2)_0$) a zvýši sa hodnota doteraz napočítaného podielu (stroj a_3).

Nasledujú ďalšie príklady realizácie funkcií a teraz už so stručnejším popisom: $f(n) = 2^n$, $f(n) = n^2$, $f(n) = \lfloor \log_2 n \rfloor$ a $f(n) = \binom{n}{2}$ pričom $n \in \mathbb{N}$.

Príklad 6.2.5 *Pomocou počítadlového stroja realizujte funkciu $f(n) = 2^i$, kde $n \in \mathbb{N}$.*
Vstupné registre: $R1 : n$
Výstupné registre: $R2 : f(n) = 2^n$
Pomocné register: $R0$

$$a_2(s_1(s_2a_0a_0)_2(s_0a_2)_0)_1$$

Najskôr je do registra $R2$ uložená hodnota 1 a teda výsledok pre 2^0 je už v registri. Pre každú dekrementáciu registra $R1$ sa zdvojnásobí hodnota registra $R2$.

Príklad 6.2.6 *Pomocou počítadlového stroja realizujte funkciu $f(n) = n^2$, kde $n \in \mathbb{N}$.*
Vstupné registre: $R1 : n$
Výstupné registre: $R2 : f(n) = n^2$
Pomocné register: $R0$ a $R3$

$$a_3(s_1(s_3a_2a_0)_3(s_0a_3)_0a_3a_3)_1$$

Výpočet n^2 by bolo možné realizovať jednoduchou modifikáciou stroja z príkladu 6.2.3 a realizovať výpočet $f(a,b) = a \cdot b$ pre $a = n$ a $b = n$. Príklad je ale riešený využitím vzťahu $f(n) = n^2 = \sum_{i=1}^{n} 2i-1$. Register $R3$ je inicializovaný na 1 a postupne nadobúda hodnoty z postupnosti $3, 5, 7, 9\ldots$ a tieto hodnoty sú pripočítavané k výsledku do registra $R2$.

Príklad 6.2.7 *Pomocou počítadlového stroja realizujte funkciu $f(n) = \binom{n}{2}$, kde $n \in \mathbb{N}$.*
Vstupné registre: $R1 : n$
Výstupné registre: $R2 : f(n) = \binom{n}{2}$
Pomocné register: $R0$ a $R3$

$$s_1a_3(s_1(s_3a_2a_0)_3(s_0a_3)_0a_3)_1$$

Aj výpočet $\binom{n}{2}$ by bolo možné realizovať jednoduchou modifikáciou stroja z príkladu 6.2.3 a realizovať výpočet pomocou vztahov $\binom{n}{2} = \frac{n\cdot(n-1)}{2}$. Príklad je riešený podobne ako predchádzajúci využitím vzťahu $f(n) = \binom{n}{2} = \sum_{i=1}^{n-1} i$.

Príklad 6.2.8 *Pomocou počítadlového stroja realizujte funkciu $f(n) = \lfloor \log_2 n \rfloor$, kde $n \in \mathbb{N}^+$.*
Vstupné registre: $R1 : n$
Výstupné registre: $R2 : f(n) = \lfloor \log_2 n \rfloor$
Pomocné register: $R0$

$$s_1((s_1s_1a_0)_1(s_0a_1)_0a_2s_1)_1$$

Stroj $(s_1s_1a_0)_1(s_0a_1)_0$ realizuje delenie dvoma, čiže po jeho vykonaní $R1 \leftarrow \lceil \frac{R1}{2} \rceil$. Dekrementáciou registra $R1$ ešte pred vykonaním cyklu delenia získavame dolnú celú časť delenia a počet „úspešných“ delení je výsledkom výpočtu.

Príklad 6.2.9 *Zostrojte počítadlový stroj, ktorý realizuje ekvivalentný výpočet ako nižšie uvedený fragment programu v jazyku „C“.*

```
if(R1>R2) R3=R1; else R3=R2;
```

Predpokladáme, že v premenných Rx môžu byť uložené iba prirodzené čísla. Hodnoty v registroch musia byť po vykonaní výpočtu na počítadlovom stroji zhodné z premennými po vykonaní výpočtu zodpovedajúceho zápisu v jazyku „C“. Pri riešení je možné použiť ľubovoľný počet pomocných registrov, ktoré sú pred vykonaním výpočtu vynulované a ich obsah po vykonaní výpočtu môže byť ľubovoľný.
Vstupné registre: $R1$, $R2$ *a* $R3$
Výstupné registre: $R1$, $R2$ *a* $R3$
Pomocné register: $R11$, $R12$ *a* $R0$

$$(s_3)_3(s_1a_0a_{11}a_3)_1(s_0a_1)_0(s_2a_0a_{12})_2(s_0a_2)_0(s_{12}s_{11})_{11}(s_{12}a_3)_{12}$$

Cieľom programu je umiestniť do registra $R3$ *väčší z obsahov registrov* $R1$ *a* $R2$. *Stroj* $(s_3)_3$ *vynuluje register* $R3$. *Ďalej sa strojom* $(s_1a_0a_{11}a_3)_1(s_0a_1)_0$ *vykoná nedeštruktívne kopírovanie obsahu registra* $R1$ *do registrov* $R3$ *a* $R11$. *V registri* $R3$ *je teda umiestnený obsah registra* $R1$. *Ostáva už iba do registra* $R3$ *pričítať prípadný rozdiel* $R2 \ominus R1$. *Najskôr stroj* $(s_2a_0a_{12})_2(s_0a_2)_0$ *nedeštruktívne nakopíruje do* $R12$ *obsah registra* $R2$. *Potom stroj* $(s_{12}s_{11})_{11}$ *určí rozdiel medzi registrami* $R2 \ominus R1$ *(rozdiel je v registri* $R12$*) a tento sa strojom* $(s_{12}a_3)_{12}$ *pričíta do* $R3$.

6.3 Stroj RAM

Napriek tomu, že Turingov stroj je najznámejším výpočtovým modelom, je veľmi vzdialený súčasným počítačom. Pamät' je tvorená páskou a prístup do pamäte vyžaduje vyhľadanie požadovaného miesta na páske, pričom sú obsahy jednotlivých políčok pásky čítané v každom kroku. Program pre Turingov stroj je realizovaný prechodovou funkciou, ktorá určuje, za akých podmienok je možné realizovat' zmenu stavu Turingovho stroja za súčasnej modifikácie aktuálneho políčka na páske a posunutia ukazateľa aktuálneho políčka. Prechodová funkcia môže byt' nedeterministická, a teda aj pri rovnakom vstupe môže byt' výpočet (postupnost' krokov výpočtu) rozdielny.

Bežný sekvenčný počítač pracuje deterministicky a jeho program je tvorený inštrukciami, ktoré väčšinou môžu pristupovat' na ľubovoľné miesto v pamäti. Stroj výpočtový RAM (Random Access Machine – stroj s ľubovoľným prístupom do pamäti) je výpočtový model veľmi podobný klasickým sekvenčným počítačom. Obsahuje pamät' dát tvorenú registrami a pamät' programu s inštrukciami. Vstupná jednotka a výstupná jednotka realizujú rozhranie s okolím a aritmeticko-logická a riadiaca jednotka interpretuje program a vykonáva operácie na základe inštrukcií.

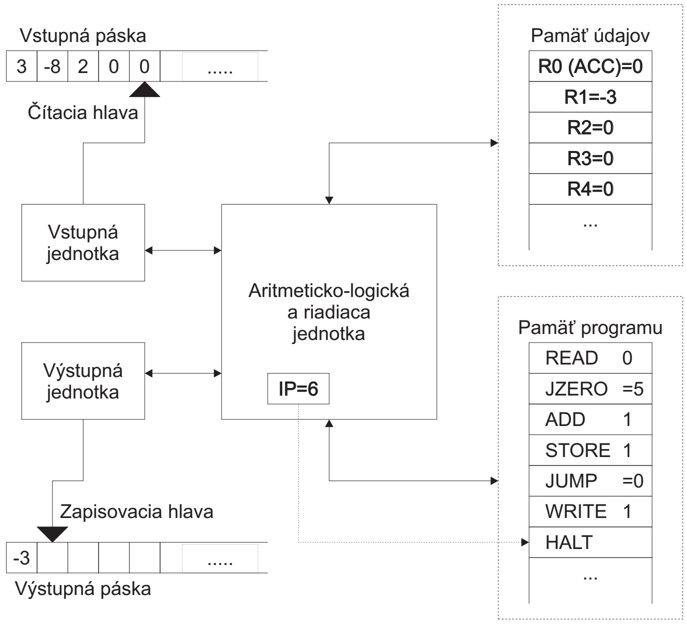
Tieto vlastnosti predurčujú RAM ako vhodný výpočtový model na skúmanie algoritmov napr. z hľadiska výpočtovej či priestorovej zložitosti. Programovanie stroja RAM je podobné programovaniu jednoduchého procesora s veľmi obmedzenou sadou inštrukcií, a teda oboznámenie sa s činnost'ou stroja môže slúžit' ako vhodný úvod do programovania v jazyku symbolických inštrukcií.

6.3.1 Neformálny opis stroja RAM

Schematické znázornenie jednotlivých častí stroja RAM je na obrázku 6.1. V pamäti údajov sú naznačené jednotlivé registre, ako aj ich obsah. V pamäti programu sú naznačené inštrukcie, adresy jednotlivých pamät'ových buniek sú indexované od 0 (indexy nie sú označené). Taktiež sú v obrázku naznačené obsahy vstupnej a výstupnej pásky. Načrtnutý program realizuje sčítanie poľa zo vstupnej pásky. Veľkost' poľa je určená ukončovacou hodnotou 0. Na obrázku je naznačený stav stroja tesne pred ukončením výpočtu inštrukciou `HALT`, ktorú určuje ukazateľ aktuálnej inštrukcie (register IP).

`Pamät' údajov`
Pamät' údajov je tvorená registrami indexovanými od 0. Počet registrov je neobmedzený. Sú jediným miestom, kde je možné počas výpočtu udržiavat' údaje. Do registra je možné uložit' celé číslo ľubovoľnej veľkosti (záporné či kladné celé číslo alebo nulu). Register je adresovaný inštrukciami určením jeho indexu. Nultý register, označený ako ACC, sa nazýva akumulátor a má špeciálny význam - je využívaný mnohými inštrukciami.

`Pamät' programu`



Obrázok 6.1: Schematické znázornenie jednotlivých blokov stroja RAM.

Základná, tu popísaná verzia stroja RAM má program oddelený od údajov, a taktiež nie je možná modifikácia programu za jeho behu. Program je tvorený jednotlivými inštrukciami vykonávanými sekvenčne. Inštrukcie sú usporiadané a na inštrukciu, ktorá sa vykoná v nasledujúcom kroku, ukazuje špeciálny register označený IP (instruction pointer). Po vykonaní aktuálnej inštrukcie sa register IP inkrementuje, a teda v ďalšom kroku bude vykonaná nasledujúca inštrukcia. Špecialným prípadom sú inštrukcie skokov, ktoré môžu tento register zmeniť, a tak riadiť vykonávanie programu. Inštrukcie sú popísané v časti „Inštrukčná sada“

`Vstupná jednotka`

Úlohou vstupnej jednotky je zabezpečiť vstupné údaje pre výpočet. Údaje sú na políčkach vstupnej pásky a na aktuálne políčko ukazuje čítacia hlava. Po načítaní vstupu z aktuálneho políčka inštrukciou READ sa čítacia hlava posunie na nasledujúce políčko, a teda ďalšie vykonanie inštrukcie READ načíta vstup z ďalšieho políčka. Návrat čítacej hlavy na už prečítané políčka nie je možný. Vstupným údajom môže byť samozrejme údaj rovnakého typu a rozsahu ako aj v registri, a to je ľubovoľne veľké celé číslo.

`Výstupná jednotka`

Výstupná jednotka zabezpečuje výstup výsledkov výpočtu a pracuje podobne ako vstupná jednotka. Inštrukcia WRITE zapíše údaj na aktuálne políčko výstupnej pásky, určené zapisovacou hlavou, a zapisovacia hlava sa posunie na nasledujúce políčko. Návrat zapisovacej hlavy nie je možný, a teda už zapísané údaje nemôžu byť prepísané.

`Aritmeticko-logická a riadiaca jednotka`
Táto časť stroja riadi výpočet v závislosti na programe. Na základe registra ukazujúceho na aktuálnu inštrukciu (register IP) je inštrukcia náčítaná z pamäte programu inštrukcia a vykoná sa operácia podľa kódu a operandu inštrukcie. Riadiaca jednotka ovláda vstupnú aj výstupnú jednotku, a taktiež číta a zapisuje údaje z pamäte údajov tvorenej registrami. Po vykonaní operácie prislúchajúcej aktuálnej inštrukcií sa register IP zmení tak, aby ukazoval na inštrukciu, ktorá bude vykonaná v nasledujúcom kroku. Riadiace inštrukcie môžu register IP priamo meniť. Ostatné inštrukcie ho nemenia a register je inkrementovaný riadiacou jednotkou.

`Inštrukčná sada a typy operandov`
Inštrukčná sada stroja RAM je veľmi jednoduchá a obmedzená. Niektoré z inštrukcií musia mať uvedený práve jeden operand, ktorý určuje údaj, s ktorým inštrukcia bude pracovať. Stroj RAM pracuje s troma typmi operandov uvedených v tabuľke 6.1. Ako príklad použitia operandu je uvedená inštrukcia sčítania `ADD`, ktorá k obsahu akumulátora pričíta hodnotu podľa operandu a výsledok uloží opäť do akumulátora.

Operand	**Príklad**	**Popis**
=i	`ADD` =-5	Konštanta *i*. V príklade použitia sa k akumulátoru pričíta číslo 5.
i	`ADD` 4	Priame adresovanie. Operand sa vyhodnotý ako obsah registra *i*. V príklade použitia sa k akumulátoru pričíta obsah registra s indexom 4.
*i	`ADD` *7	Nepriame adresovanie. Obsah registra *i* určí register, ktorého obsah je výsledkom vyhodnotenia operandu. V príklade použitia sa k akumulátoru pričíta obsah registra s indexom daným obsahom registra číslo 7.

Tabuľka 6.1: Typy operandov inštrukcií stroja RAM

Formálne je možné zaviesť operátor c(x), ktorý vráti hodnotu zodpovedajúcu obsahu registra s indexom x. Potom operátor v(op), ktorý vyhodnotí operand op, je možné definovať ako:

- v(=i) = i
- v(i) = c(i)
- v(*i) = c(c(i))

V prípade, ak by pri vyhodnotení operandu nastala nezmyselná situácia, a to pokus o čítanie registra so záporným indexom, sa výpočet stroja RAM zastaví.

Inštrukcia		
Kód	**Operand**	**Popis**
Inštrukcie pre prácu s pamäťou		
`LOAD`	operand	operand sa načíta do akumulátora
`STORE`	operand	obsah akumulátora sa zapíše do pamäte podľa operandu
Aritmetické inštrukcie		
`ADD`	operand	operand sa pričíta do akumulátora (an. addition)
`SUB`	operand	operand sa odčíta od akumulátora (an. subtraction)
`MUL`	operand	akumulátora sa prenásobí operandom (an. multiplication)
`DIV`	operand	akumulátor sa predelý operandom (an. division)
Vstupno výstupné inštrukcie		
`READ`	operand	údaj zo vstupnej pásky sa zapíše do pamäte podľa operandu
`WRITE`	operand	operand sa zapíše na výstupnú pásku
Riadiace inštrukcie		
`JUMP`	návestie	skok na návestie
`JZERO`	návestie	skok na návestie, ak akumulátor nulový (an. jump if zero)
`JGTZ`	návestie	skok, ak akum. väčší ako nula (an. jump if grater than zero)
`HALT`		zastavenie výpočtu

Tabuľka 6.2: Inštrukčná sada stroja RAM

Poznámka 6.3.1 *V tabuľke 6.2 je uvedená inštrukčná sada stroja RAM. Pri inštrukciach* `READ` *a* `STORE` *sa operand nevyhodnocuje za účelom získania hodnoty, tak ako je to v prípade ostatných inštrukcií s operandom a ako bolo vysvetlené v tabuľke 6.1. Obsah pamäťového miesta určenom operandom sa zmení na hodnotu zo vstupu (*`READ`*) alebo na hodnotu z akumulátora (*`STORE`*). V týchto prípadoch nemá zmysel uviesť ako operand konštantu a takýto zápis je možné považovať za neprípustný.*

Poznámka 6.3.2 *Návestie je v tabuľke uvedené ako samostatný typ operandu, ale je vhodné ho považovať za operand typu číselná konštanta. Pamäť s programom je tvorená pamäťovými miestami, kde sú umiestnené kódy inštrukcií. Pamäťové miesta sú očíslované a inštrukcie skokov uvádzajú ako operand práve číslo pamäťového miesta s inštrukciou, ktorá bude v prípade realizovania skoku vykonaná. Toto číslo je v prípade skoku zapísané do registra IP.*

`Výpočet na stroji RAM`

Na začiatku výpočtu je pamäť programu prázdna a tiež sú všetky registre vynulované. Čítacia a aj zapisovacia hlava sú na začiatkoch vstupnej a výstupnej pásky a na vstupnú pásku sú umiestnené vstupné údaje. Program je zavedený do pamäte programu a register IP je nastavený na prvú inštrukciu (keďže je prvá inštrukcia umiestnená v pamäti programu na pamäťovom mieste s indexom 0, bol aj register IP nastavený na 0). Následne začne riadiaca jednotka vykonávať program.

Vykonávanie programu je ukončené inštrukciou HALT, alebo ak nastane niektorá z nezmyselných situácií, ako napríklad pokus o čítanie či zápis do registra so záporným indexom alebo pokus o vykonanie inštrukcie z pamäťového miesta, na ktoré žiadna inštrukcia nebola umiestnená (neuvedenie inštrukcie HALT na konci programu). Výstup z programu je umiestnený na políčkach výstupnej páske až po políčko pred pozíciu zapisovacej hlavy.

Poznámka 6.3.3 *Stroj RAM je možné modifikovať viacerými spôsobmi. Jednoduchšie modifikácie spočívajú v zmene inštrukčnej sady. Stroj bez inštrukcii* MUL *a* DIV *označujeme ako RAM+. Pridaním inštrukcií* ACCEPT *a* REJECT *je možné stroj prispôsobiť úlohe rozpoznávania slov jazyka generovaného gramatikou. Výpočet je ukončený rozpoznaním slova (*ACCEPT*), resp. zamietnutím slova (*REJECT*). Zaujímavým rozšírením stroja RAM je umožniť programu modifikovať sám seba zmenou inštrukcií v pamäti (stroj RASP - Random Access Stored Program).*

6.3.2 Výpočtová zložitosť v modeli RAM

Časová či priestorová zložitosť môže byť významným kritériom hodnotenia kvality programu. Stroj RAM bol navrhnutý tak, aby na ňom simulovaný výpočet zodpovedal reálnym používaným počítačom. Zložitostné miery určené pre program RAM môžu mať významnú výpovednú hodnotu, korelujúcu so zložitosťami reálnych programov.

Časová zložitosť sa vzťahuje na výpočtovú náročnosť programu a udáva množstvo „času“ potrebného na vykonanie programu. Priestorová zložitosť zase určuje pamäťovú náročnosť programu. Očakávanú čiže priemernú zložitosť pre vstupy danej dĺžky je možné určiť, ak vieme odhadnúť proavdepodobnostné rozdelenie vstu-pov danej dĺžky. Horné ohraničenie zložitosti sa počíta pre vstup zodpovedajúci najhoršiemu možnému scenáru, a preto sa väčšinou určuje ľahšie ako očakávaná zložitosť.

Aby bolo možné vypočítať zložitosť, či už časovú alebo priestorovú, je nevyhnutné stanoviť si cenu za vykonanie inštrukcie či cenu za použitie registra. Pre RAM tu zadefinujeme dve zložitostné miery: tzv. jednotkovú a tzv. logaritmickú zložitostnú mieru.

Jednotková zložitostná miera

Pri určovaní časovej jednotkovej zložitosti vykonanie ľubovoľnej inštrukcie stojí 1 časovú jednotku. Pri určovaní pamäťovej zložitosti použitie registra stojí 1 pamäťovú jednotku. Z uvedeného vyplýva, že pre zvolený vstup je časová jednotková zložitosť pre daný program rovná počtu inštrukcií, ktoré vykoná program pri spracovaní zvoleného vstupu. Priestorová jednotková zložitosť je rovná počtu registrov použitých v programe pri spracovaní zvoleného vstupu. Jednotková zložitostná miera neberie do úvahy veľkosť čísel, s ktorými inštrukcie pracujú, a ktoré sú zaznamenané do registrov.

Logaritmická zložitostná miera

Logaritmická zložitostná miera je v niektorých prípadoch realistickejšia, pretože berie do úvahy obmedzenú veľkosť slova v pamäti reálneho počítača. Čím je číslo, s ktorým pracuje inštrukcia väčšie, tým je inštrukcia „drahšia“, čiže dlhšie trvá a tiež viac pamäte je nevyhnutné na uloženie tejto hodnoty. Cena rastie logaritmicky vzhľadom na veľkosť čísla. Náročnosť manipulácie s číslom i je daná nasledujúcou funkciou:

$$l(i) = \begin{cases} \lfloor \log |i| \rfloor + 1 & \text{ak } i \neq 0 \\ 1 & \text{ak } i = 0 \end{cases}$$

Funkčná hodnota $l(i)$ zodpovedá počtu bitov nevyhnutných na uchovanie čísla i. Použitím tejto funkcie je možné definovať časovú náročnosť práce s operandom $t(op)$ (tabuľka 6.3).

Operand op	**Cena** $t(op)$	**Popis**
=i	$l(i)$	Cena daná náročnosťou manipulácie s číslom i.
i	$l(i) + l(c(i))$	Cena daná náročnosťou manipulácie s indexom registra i (čím vyšší index, tým nákladnejší prístup) a obsahom registra i.
*i	$l(i)+l(c(i))+ l(c(c(i)))$	Je nevyhnutné „zaplatiť“ za prístup do registra i, potom za prístup do registra $c(i)$ (register je určený nepriamou adresáciou pomocou registra i) a nakoniec za manipuláciu s číslom $c(c(i))$ v registri $c(i)$.

Tabuľka 6.3: Časovú náročnosť práce s jednotlivými operandami.

Nakoniec je možné definovať ceny inštrukcií vzhľadom na typ a veľkosť operandov (tabuľka 6.4).

Poznámka 6.3.4 *V prípade inštrukcií pracujúcich s akumulátorom je nevyhnutné platiť aj za obsah akumulátora. Prístup do akumulátora ako špecialneho registra určeného na aritmetické a riadiace operácie je okamžitý a nestojí nijakú časovú jednotku. Zaujímavejšia je cena inštrukcie* `READ`*, kde je potrebné zaplatiť aj za veľkosť načítavaného vstupu. Inštrukcia* `WRITE` *už ale za veľkosť zapisovaného výstupu neplatí, táto cena je už započítaná v cene operandu. Inštrukcie* `READ` *a* `STORE` *zapisujú údaj do registra podľa operanda. Veľkosť „prepísaného“ čísla v registri sa pri výpočte časovej zložitosti nezohľadňuje, preto vo výpočte ceny nevystupuje vzťah $t(op)$, ale príspevky za prístup do registra sú rozpísané pre povolené typy operandov.*

Inštrukcia		
Kód	**Operand**	**Cena**
`LOAD`	op	$t(op)$
`STORE`	i	$l(c(0)) + l(i)$
`STORE`	*i	$l(c(0)) + l(i) + l(c(i))$
`ADD`	op	$l(c(0)) + t(op)$
`SUB`	op	$l(c(0)) + t(op)$
`MUL`	op	$l(c(0)) + t(op)$
`DIV`	op	$l(c(0)) + t(op)$
`READ`	i	$l(vstup) + l(i)$
`READ`	*i	$l(vstup) + l(i) + l(c(i))$
`WRITE`	op	$t(op)$
`JUMP`	návestie	1
`JZERO`	návestie	$l(c(0))$
`JGTZ`	návestie	$l(c(0))$
`HALT`		1

Tabuľka 6.4: Časová náročnosť jednotlivých inštrukcií.

Logaritmická časová zložitosť programu je definovaná ako súčet cien všetkých inštrukcií vykonaných programom pri spracovaní zvoleného vstupu. Ak sa inštrukcie nachádzajú v cykle, sú samozrejme ich ceny započítavané do zložitosti toľkokrát, v koľkých iteráciach cyklov sa vykoná. Pri každej iterácií cyklu môže byť cena inštrukcie v cykle rozdielna, a to v závislosti na obsahu registrov, s ktorými inštrukcie pracujú, a ktoré sa v priebehu výpočtu menia.

Logaritmická priestorová zložitosť je definovaná ako súčet hodnôt $l(x_i)$ cez všetky registre (aj akumulátor), kde x_i je číslo s najväčšou absolútnou hodnotou, ktoré bolo v priebehu výpočtu v registri i.

Tabuľka 6.5 sumarizuje výpočtové zložitosti stroja RAM.

	Zložitostná miera	
Typ zložitosti	**Jednotková**	**Logaritmická**
Časová	Jednotková časová	Logaritmická časová
Priestorová	Jednotková priestorová	Logaritmická priestorová

Tabuľka 6.5: Tabuľka so zložitosťami.

6.3.3 Riešené príklady

Príklady

Príklad 6.3.1 *Pomocou stroja RAM realizujte funkciu $f(n) = 2^n$, kde $n \in \mathbb{N}$.*

```
      READ   1
      LOAD   =1
      STORE  2
      LOAD   1
      JZERO  =12
      LOAD   2
      MUL    =2
      STORE  2
      LOAD   1
      SUB    =1
      STORE  1
      JUMP   =3
      WRITE  2
      HALT
```

```
      READ   N
      LOAD   =1
      STORE  RES
next: LOAD   N
      JZERO  end
      LOAD   RES
      MUL    =2
      STORE  RES
      LOAD   N
      SUB    =1
      STORE  N
      JUMP   next
end:  WRITE  RES
      HALT
```

Obrázok 6.2: (a) Program pre stroj RAM realizujuci výpočet funkcie $f(n) = 2^n$. (b) Modifikovaná verzia programu s pomenovanými návestiami a registrami.

Na obrázku 6.2(a) je program, ktorý realizuje výpočet 2^n pre n z množiny prirodzených čísel. Hodnota n je zadaná ako vstup na vstupnej páske a program zapíše výsledok na výstupnú pásku. Aby bol zápis programu čitateľnejší, je vhodné určiť konštanty pre pamäťové miesta v pamäti programu návestiami zapísanými priamo do programu a tieto návestia používať pri inštrukciách skokov. Návestia je vhodné umiestniť pred inštrukciu a slovný reťazec, jednoznačne identifikujúci návestie, ukončiť dvojbodkou. Pri inštrukciách skoku je návestie uvedené bez dvojbodky. Prehľadnosti programu tiež pomôže vhodné pomenovanie registrov. Takto upravený program je uvedený na obrázku 6.2(b).

Príklad 6.3.2 *Pomocou stroja RAM realizujte program, ktorý pole prirodzených čísel zadané na vstupe vráti na výstupe s prvkami v „obrátenom“ poradí.*

Príklad je zadaný pomerne neformálne a je potrebné zvoliť vhodnú formu zadania vstupného poľa a výstupného poľa. Pole a[] je určené najskôr počtom prvkom v poli (1. vstup/výstup), ktorý je nasledovaný prvkami poľa $a[0], a[1], \ldots a[n-1]$, kde n je počet prvkov poľa. Prvky sú najskôr načítané do pamäte, a potom sú postupne od $a[n-1], a[n-2], \ldots$ až po $a[0]$ vypísané na výstup. Aby sa zachoval zvolený formát, prvá vykonaná inštrukcia `WRITE` *vypíše na výstup počet prvkov poľa. Riešenie je na obrázku 6.3.*

```
          READ    0
          STORE   I
          STORE   N
    nxt:  JZERO   rev
          LOAD    I
          ADD     =10
          READ    *0
          LOAD    I
          SUB     =1
          STORE   I
          JUMP    nxt
    rev:  WRITE   N
   nxt2:  LOAD    N
          SUB     I
          JZERO   end
          LOAD    I
          ADD     =1
          STORE   I
          ADD     =10
          WRITE   *0
          JUMP    nxt2
    end:  HALT
```

Obrázok 6.3: Program pre stroj RAM, ktorý „obráti“ prvky pola.

Príklad 6.3.3 *Pre program z príkladu 6.2 určite jednotkovú časovú a priestorovú zložitosť ako aj logaritmickú časovú a priestorovú zložitosť.*

Jednotková časová zložitosť je daná počtom vykonaných inštrukcií. Ako je zrejmé z tabuľky 6.6 zo stĺpca „Počet vykonaní“, časovú jednotkovú zložitosť v závislosti na veľkosti vstupu n je možné spočítať ako $T_1(n) = 7 + 9n$ *čiže* $T_1(n) = O(n)$.

Jednotková priestorová zložitosť je daná počtom použitých registrov v závislosti na veľkosti vstupu. Pri výpočte sa vždy použijú 3 registre, a teda $S_1(n) = 3$, *čiže* $S_1(n) = O(1)$.

Logaritmickú časovú zložitosť je možné určiť pomocou stĺpca „Logaritmická cena“ z tabuľky 6.6. Cena inštrukcie môže závisieť od veľkosti vstupu označeného n. *Cena niektorých inštrukcií sa v priebehu cyklu mení. Závisí od iterácie cyklu s indexom* i, *pričom prvá iterácia cyklu má index* $i = 0$. *Príspevok od prvých troch inštrukcií programu si môžeme vyjadriť ako:*

$$T_l^A(n) = l(n) + 3l(1) + l(2) = O(\log(n))$$

Príspevok od posledných dvoch inštrukcií je môžné vyjadriť ako:

$$T_l^C(n) = 1 + l(2) + l(2^n) \approx \log 2^n = n \log 2 = O(n)$$

Príspevok od inštrukcií v cycle je potrebné si vyjadriť za pomoci iterácie cyklu i.

$$T_l^B(n) = \sum_{i=0}^{n-1} \Big[4l(1)+3l(2)+1+l(n-i-1)+4l(n-i)+2l(n^i)+l(n^{(i+1)})\Big]+l(1)+2l(0).$$

V tomto výpočte inštrukcií bol už zohľadnený fakt, že prvé dve inštrukcie cyklu sa vykonajú $n+1$*–krát. Zaujíma nás asymptotický odhad zložitosti, a teda v tomto smere menej význammné prírastky môžeme zanedbať a logaritmickú časovú zložitosť cyklu* *môžeme vyjadriť ako:*

$$T_l^B(n) \approx \sum_{i=0}^{n-1} \log(n^i) = \log(n) \sum_{i=0}^{n-1} i = \log(n)\frac{(n-1)n}{2} = O(n^2 \log(n))$$

Celková zložitosť je daná súčtom zložitostí pre jednotlivé bloky programu:

$$T_l(n) = T_l^A(n) + T_l^B(n) + T_l^C(n) = O(\log(n)) + O(n^2 \log(n)) + O(n) = O(n^2 \log(n)).$$

*Logaritmickú priestorovú zložitosť je daná súčtom „najdrahších“ obsahov všetkých registrov v priebehu behu programu. Najväčšie číslo zapísané v priebehu programu v registri R*0 *je* 2^n*, v registri R*1 *je n a v registri R*2 *opäť* 2^n*. Logaritmická priestorová zložitosť je teda:*

$$S_l(n) = 2l(2^n) + l(n) \approx n \log 2 = O(n).$$

Inštrukcia			**Počet vykonaní**	**Logaritmická cena**
	`READ`	`1`	1	$l(n) + l(1)$
	`LOAD`	`=1`	1	$l(1)$
	`STORE`	`2`	1	$l(2) + l(1)$
`next:`	`LOAD`	`1`	n+1	$l(1) + l(n-i)$
	`JZERO`	`end`	n+1	$l(n-i)$
	`LOAD`	`2`	n	$l(2) + l(n^i)$
	`MUL`	`=2`	n	$l(n^i) + l(2)$
	`STORE`	`2`	n	$l(n^{(i+1)}) + l(2)$
	`LOAD`	`1`	n	$l(1) + l(n-i)$
	`SUB`	`=1`	n	$l(n-i) + l(1)$
	`STORE`	`1`	n	$l(n-i-1) + l(1)$
	`JUMP`	`next`	n	1
`end:`	`WRITE`	`2`	1	$l(2) + l(2^n)$
	`HALT`		1	1

Tabuľka 6.6: Program s počtom vykonania a logaritmickpu cenou jednotlivých inštrukcií. n označuje vstup a i je iterácia cyklu (začína od 0).

Výsledky sú zhrnuté v tabuľke 6.7. Je zrejmé, že pre rôzne zložitostné miery sú výsledné odhady, či už časových, či pamäťových zložitostí rozdielne.

	Zložitostná miera	
Typ zložitosti	**Jednotková**	**Logaritmická**
Časová	$T_1(n) = O(n)$	$T_l(n) = O(n^2 \log(n))$
Priestorová	$S_1(n) = O(1)$	$S_l(n) = O(n)$

Tabuľka 6.7: Tabuľka s vypočítanými zložitosťami.

6.4 Ekvivalencia výpočtových modelov

Veta 6.4.1 *(O ekvivalencií výpočtových modeloch) Nasledujúce výpočtové modely sú ekvivalentné:*

1. *Turingov stroj*
2. *Počítadlový stroj*
3. *Stroj RAM*

6.4.1 Riešené príklady

Príklad 6.4.1 *Dokážte, že výpočet každého počítadlového stroja sa dá realizovať na stroji RAM.*

Je vhodné postupovať podľa rekurzívnej definície počítadlového stroja. Ku každému počítadlovému stroju je možné zostrojiť program pre stroj RAM realizujúci ekvivalentný výpočet. Registre stroja RAM je možné stotožniť s registrami počítadlového stroja, až na register $R0$ *(akumulátor), ktorý má špeciálnu funkciu. Preto je vhodné namapovať každý register* Ri *počítadlového stroja na register* $R(i+1)$ *stroja RAM. Register* $R0$ *stroja RAM nebude zodpovedať žiadnemu registru počítadlového stroja, a teda ho bude možné používať pri výpočtoch ako akumulátor. V registroch stroja RAM môžu byť uložené čísla* $n \in \mathbb{Z}$*, v registroch počítadlového stroja môžu byť uložené čísla* $n \in \mathbb{N}$*, a teda mapovanie resgistrov počítadlového stroja na registre stroja RAM je možné (*$\mathbb{N} \subset \mathbb{Z}$*).*

Vstupné argumenty pre počítadlový stroj sú umiestnené priamo v registroch počítadlového stroja, ktoré sú označené ako vstupné. Ostatné registre sú vynulované. Stroj RAM má na začiatku výpočtu všetky registre vynulované a vstupné argumenty sú na vstupnej páske. Je zrejmé, že jednoduchou sekvenciou inštrukcií stroja RAM je možné zaviesť vstupné údaje do zodpovedajúcich registrov. Ak Ri je vstupný register, inštrukciou `READ i+1` *sa obsah zapíše do zodpovedajúceho registra stroja RAM (registra R(i+1)). Obdobným spôsobom pomocou inštrukcie* `WRITE` *je možné zrealizovať výstup z výstupných registrov.*

Každý počítadlový stroj je možné zostrojiť aplikovaním krokov podľa rekurzívnej definície. Konštrukcia ekvivalentného programu pre stroj RAM môže byť realizovaná súbežne s konštrukciou počítadlového stroja:

- *a_n a s_n sú počítadlové stroje, pričom $n \in \mathbb{N}$, k nim zodpovedajúce programy pre stroj RAM sú:*

```
LOAD  n+1
ADD   =1
STORE n+1
```

```
      LOAD  n+1
      JZERO skip
      SUB   =1
      STORE n+1
skip:
```

- *ak $M1$, $M2$ ... Mn sú počítadlové stroje, ktorým zodpovedajú RAM programy $P1$, $P2$ ... Pn, potom aj $M1M2 \ldots Mn$ je počítadlový stroj, ktorý realizuje rovnaký výpočet ako RAM program:*

```
P1
P2
...
Pn
```

- *ak M je počítadlový stroj, ktorému zodpovedá RAM program P, tak aj $(M)_k$ je počítadlový stroj, pričom $k \in \mathbb{N}$, a RAM program zodpovedajúci stroju $(M)_k$ je:*

```
next: load  k+1
      jzero end
      P
      jump  next
end:
```

Návestia `skip`*,* `next` *a* `end` *reprezentujú adresy inštrukcií, na ktoré tieto návestia v jednotlivých RAM programov ukazujú. Pri zoskupovaní programov do väčších celkov je samozrejme nevyhnutné zodpovedajúco tieto adresy skokov modifikovať tak, aby stále ukazovali na tie isté inštrukcie.*

Takto skonštruovaný počítadlový stroj a súbežne skonštruovaný RAM program realizujú rovnaký výpočet pre vstupné argumenty z $\mathbb{N}$.

6.5 Cvičenia

Cvičenie 6.5.1 *Napíšte gramatiku, ktorá generuje jazyk obsahujúci všetky slová, ktoré zodpovedajú syntakticky správnym počítadlovým strojom.*

Cvičenie 6.5.2 *Zostrojte počítadlové stroje, ktoré realizujú ekvivalentné výpočty ako nižšie uvedené fragmenty programov v jazyku „C“.*

1. `if(R1>(R2-R3)) R4=R1*R1;`
2. `while(R3>R2) { R2 = R2*R2; R3=2*R3 }`
3. `if(R1+R2>R3) R4=R1+R2; else R2=0;`
4. `while(R1<R2) { R2 = R2-R3; }`

Cvičenie 6.5.3 *Zostrojte počítadlové stroje, ktoré realizujú nasledujúce výpočty pre* $i, j, k, n \in \mathbb{N}$*:*

1. $f(i,j,k) = \lceil \log_2(1 + i + j \ominus k) \rceil$
2. $f(n) = 2n + \lceil \log_2 n \rceil$
3. $f(i,j) = \binom{i+j}{2} + 1$
4. $f(i,j,k) = 2i + 3j + 3k \ominus 2$
5. $f_4(i,j,k) = \lfloor i/2 \rfloor + \lfloor j/3 \rfloor + \lfloor k/2 \rfloor$
6. $f(i,j,k) = \lceil \log_2(2i + j + k) \rceil$
7. $f(i,j,k) = \lceil j/3 \rceil + \lceil \log_2(i + k) \rceil$
8. $f(i,j,k) = 3j + \lceil \log_2(i \ominus k) \rceil$
9. $f(n) = n + (2n)^2$
10. $f(n) = (n \ominus 2)^2$
11. $f(n) = n + n^2 \ominus 3$
12. $f(i,j,k) = (i + k)^2 + 3j + 1$
13. $f(i,j,k) = \lceil i/4 \rceil + \lceil j/3 \rceil \ominus k$
14. $f(i,j) = i + \binom{2j}{2}$
15. $f(i,j) = i^2 + 2j$

Cvičenie 6.5.4 *Zostrojte počítadlové stroje, ktoré realizujú nasledujúce výpočty pre* $i, j, k, n \in \mathbb{N}$*:*

1. $f(i,j,k) = \lceil \log_2 \lceil (i + j + k)/3 \rceil \rceil$

2. $f(i,j) = \binom{i \ominus j}{2}$
3. $f(i,j) = 2(i+j)^2$
4. $f(n) = \lceil \log_2(3n) \rceil$
5. $f(i,j) = (i+2j)^2$
6. $f(i,j) = (i \ominus j)^2 + 1$
7. $f(n) = \lceil \log_2(n+2) \rceil$
8. $f(i,j,k) = (i \ominus j + k)^2$
9. $f(n) = \binom{n+3}{2}$
10. $f(i,j,k) = 1 + i + k + \lceil \log_2 j \rceil$
11. $f(i,j,k) = \lceil i/3 \rceil \ominus 3j + \lceil k/2 \rceil + 2$
12. $f(i,j,k) = \lceil \log_2(3i + \lceil j/3 \rceil + k) \rceil$
13. $f(n) = \binom{2n}{2}$
14. $f(n) = 2n + \binom{n}{2}$
15. $f(n) = (n+3)^2 + 1$
16. $f(i,j,k) = \lceil \log_2(j+k+2) \rceil \ominus i$

Cvičenie 6.5.5 *Zostrojte počítadlové stroje, ktoré realizujú nasledujúce výpočty pre* $i,j,k,n \in \mathbb{N}$*:*

1. $f(i,j,k) = 2i + 2^{(j+1 \ominus k)}$
2. $f(i,j,k) = 2^{(1+i+j \ominus k)}$
3. $f(i,j,k) = 2^{(\lceil i/2 \rceil + j + 3k)}$
4. $f(i,j,k) = 1 + 2^{\lceil (i+j+k)/3 \rceil}$
5. $f(i,j,k) = i \ominus k + 2^{2j}$
6. $f(i,j,k) = 3k + 2^{\lceil (1+i+j)/3 \rceil}$
7. $f(i,j,k) = \lceil j/2 \rceil + 2^{(i+2k)}$
8. $f(n) = n!$

Cvičenie 6.5.6 *Zostrojte počítadlové stroje, ktoré realizujú nasledujúce výpočty pre* $i, j, k, n \in \mathbb{N}$*:*

1. $f(i,j,k) = \lfloor (i \cdot j)/(k+2) \rfloor$
2. $f(i) = i^3$
3. $f(i,j,k) = i \cdot j^2 \cdot k^3$
4. $f(i,j) = \lfloor \log_2 i \rfloor + \lceil \log_2 j \rceil$
5. $f(i,j) = \lfloor \log_i j \rfloor$
6. $f(i,j) = i^j$
7. $f(n) = F(n)$, *kde* $F(0) = F(1) = 1$ *a* $F(n) = F(n-1) + F(n-2)$ *pre* $n > 1$.
8. $f(i,j) = \binom{i}{j}$

Cvičenie 6.5.7 *Nápíšte programy pre RAM, ktoré realizujú nasledovné úlohy:*

1. *Vypíšte prvých* N *prirodzených čísel, ktoré nie sú mocninou čísla* 2*, v poradí od najväčšieho po najmenšie.*
2. *Zistite, či kladné celé číslo* N *je mocninou nejakého iného prirodzeného čísla menšieho ako* N*.*
3. *Zistite, či možno vyjadriť zadané kladné celé číslo* N *ako súčet faktoriálov nejakých dvoch prirodzených čísel (napr.* $744 = 6! + 4!$*)*
4. *Dokonalé číslo je také číslo* N*, pre ktoré platí, že súčet jeho deliteľov menších ako* N *je rovný* N *(napr.* $6 = 3 + 2 + 1$*). Preverte, že číslo* 8128 *je dokonalé. Zistite, koľkým dokonalým číslom v poradí je.*
5. *Dvojica kladných celých čísel* A*,* B *sa nazýva spriatelená, ak súčet deliteľov čísla* A *sa rovná číslu* B *a súčet deliteľov čísla* B *sa rovná* A*. Napr. čísla* 220 *a* 284 *sú spriatelené:* $220 = 1 + 2 + 4 + 71 + 142$ *a* $284 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110$*. Vypíšte všetky dvojice spriatelených čísel menších ako* N*.*
6. *Medzi prvočíslami od* 1 *po* N *nájdite dvojicu susedných prvočísel s najväčším rozdielom.*
7. *Polynóm* $f(x)$ *je rovný* $x^2 + 79x + 1601$*. Nájdite najmenšie prirodzené číslo* x*, pre ktoré hodnota* $f(x)$ *nie je prvočíslom.*
8. *Číslo* N *rozložte (ak sa rozložiť dá) na súčet dvoch prvočísel.*

9. *Vypíšte prvočíslo, ktoré v prvočíselnom rozklade čísla N vystupuje s maximálnou mocninou.*

10. *Načítajte kladné celé číslo N; jeho ciferný zápis cyklicky posuňte o 3 miesta doprava a získané číslo vypíšte.*

11. *Načítajte kladné celé číslo N; jeho ciferný zápis obráťte a získané číslo vypíšte.*

12. *N-ciferné číslo sa nazýva Armstrongovo, ak súčet N-tých mocnín jeho cifier je rovný jemu samému (napr.* $153 = 13 + 53 + 33; 407 = 43 + 03 + 73$*). Nájdite a vypíšte všetky dvoj-, troj- a štvorciferné Armstrongove čísla.*

13. *Vypíšte takých prvých N čísel, ktorých posledná cifra sa rovná súčtu všetkých predchádzajúcich cifier.*

14. *Načítajte kladné celé číslo N a prirodzené číslo z $(2 <= z <= 9)$ označujúce číselnú sústavu. Preveďte číslo N do sústavy o základe z.*

15. *Načítajte tri prirodzené čísla M, N, O a vypíšte ich v takom poradí, aby počet núl v ich dvojkových zápisoch neklesal.*

16. *Načítajte kladné celé číslo N a po ňom nasledujúcich N celých čísel. Usporiadajte načítané čísla algoritmom INSERT SORT.*

17. *Načítajte kladné celé číslo N a po ňom nasledujúcich N celých čísel. Usporiadajte načítané čísla algoritmom SELECT SORT.*

18. *Načítajte kladné celé číslo N a po ňom nasledujúcich N celých čísel. Usporiadajte načítané čísla algoritmom BUBBLE SORT.*

19. *Napíšte program, ktorý rozpozná jazyk $L = \{wwRw$ pričom w je slovo nad abecedou $A = \{0, 1, 2, \ldots, 9\}\}$.*

Cvičenie 6.5.8 *Je daná množina registrov $R0$, $R1$, ..., v ktorých je možné reprezentovať nezáporné celé čísla. Je daný počítač WHILE-RAM+$\oplus$, ktorý nad uvedenou množinou registrov používa inštrukcie počítača RAM okrem násobenia, delenia a všetkých skokov. Stroj tiež používa konštrukciu* `while( GZERO ){ ... }` *, čo je cyklus, ktorý vykonáva svoje telo pokiaľ hodnota registra $R0$ je kladná (väčšia ako nula). Inštrukcia* `SUB` *realizujú operáciu odpočítania $\ominus$. Prepíšte zadanú postupnosť WHILE-RAM+$\oplus$ inštrukcií na počítadlový stroj tak, aby vykonával ten istý výpočet. Pri konštrukcií počítačového stroja nezisťujte, aký výpočet zadaný fragment programu pre WHILE-RAM+$\oplus$ realizuje.*

```
while( GZERO )
{
  SUB   =2
```

```
    STORE  1
    LOAD   2
    ADD   =3
    STORE  2
    LOAD   1
  }
  LOAD  2
  ADD   4
```

Cvičenie 6.5.9 *Je daná množina registrov $R0$, $R1$, ..., v ktorých je možné reprezentovať nezáporné celé čísla. Je daný počítač WHILE-RAM+⊕, ktorý nad uvedenou množinou registrov používa nasledujúce inštrukcie:*

`ADD =k`	*pripočítanie konštanty s hodnotou k k obsahu registra $R0$*
`SUB =k`	*odpočítanie konštanty s hodnotou k od obsahu registra $R0$*
`LOAD k`	*načítanie hodnoty registra Rk do registra $R0$*
`STORE k`	*uloženie hodnoty registra $R0$ do registra Rk*

Inštrukcia `SUB` *realizuje operáciu odpočítania ⊖. Stroj tiež používa konštrukciu* `while( GZERO ){ ... }` *, čo je cyklus, ktorý vykonáva svoje telo pokiaľ hodnota registra $R0$ je kladná (väčšia ako nula). Vstupné argumenty pre WHILE-RAM+⊕ sa nachádzajú pred výpočtom v registroch (nemá inštrukcie READ a WRITE). Dokážte, že WHILE-RAM+⊕ je ekvivalentný počítadlovým strojom. Je potrebné dokázať dve implikácie s využitím rekurzívnych definícií počítadlového stroja a stroja WHILE-RAM+⊕.*

Cvičenie 6.5.10 *Je daná množina registrov $R0$, $R1$, ..., v ktorých je možné reprezentovať nezáporné celé čísla. Je daný počítač WHILE-RAM+⊕, ktorý nad uvedenou množinou registrov používa nasledujúce inštrukcie:*

`ADD i, =k`	*pripočítanie konštanty s hodnotou k k obsahu registra Ri*
`ADD i, k`	*pripočítanie hodnoty registra Rk k obsahu registra Ri (Výsledok v registri Ri)*
`SUB i, =k`	*odpočítanie konštanty s hodnotou k od obsahu registra Ri*
`SUB i, k`	*odpočítanie hodnoty registra Rk od obsahu registra Ri (Výsledok v registri Ri)*

Inštrukcie `SUB` *realizujú operáciu odpočítania ⊖. Stroj tiež používa konštrukciu* `while( Ri ){ ... }` *, čo je štandardný while cyklus z C-jazyka, ktorý vykonáva svoje telo pokiaľ hodnota registra Ri je kladná (väčšia ako nula). Vstupné argumenty pre WHILE-RAM+⊕ sa nachádzajú pred výpočtom v registroch (nemá inštrukcie READ a WRITE). Dokážte, že WHILE-RAM+⊕ je ekvivalentný počítadlovým strojom. Je potrebné dokázať dve implikácie s využitím rekurzívnych definícií počítadlového stroja a stroja WHILE-RAM+⊕.*