

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



TIN Teoretická informatika

2. domácí úloha

Obsah

1	Príklad číslo 1	2
1.1	(a)	2
1.2	(b)	2
2	Príklad číslo 2	4
3	Príklad číslo 3	5
3.1	Nerozhodnuteľnosť	5
3.2	Čiastočná rozhodnuteľnosť	6
4	Príklad číslo 4	7
4.1	(a)	7
4.2	(b)	10
5	Literatúra	14

1 Príklad číslo 1

1.1 (a)

Definice 4.29 [1](str. č. 97) Označme ZAV_n pre $n \geq 0$ jazyky setávajúcí ze všech vyvážených řetězců závorek n typů. Tyto jazyky – označované též jako Dyckovy jazyky – jsou generovány gramatikami s pravidly tvaru: $S \rightarrow [^1 S]^1 \mid [^2 S]^2 \mid \dots \mid [^n S]^n \mid SS \mid \varepsilon$

Z hore uvedenej definície pre náš príklad vyplýva, že náš Dyckov jazyk L je generovaný gramatikou G_D , definovanou ako $G_D = (\{S'\}, \{[,]\}, P, S')$ kde množina prepisovacích pravidiel P je daná ako

$$S' \rightarrow \varepsilon \mid S'S' \mid [S']$$

ktorá obsahuje iba jeden typ zátvoriek ktorými sú $[$ a $]$.

Pre každé slovo $w \in L$, pre ktoré platí že $w \neq \varepsilon$, muselo byť aspoň raz použité pravidlo $S' \rightarrow [S']$ v derivácii. S využitím pravidla $S' \rightarrow [S']$ vygenerujeme jeden pár zátvoriek, pričom medzi zátvorkami sa nachádza neterminál S' , ktorým je možné ďalej aplikovať ďalšie pravidlá a generovať reťazec u za pomoci gramatiky G_D – presnejšie reťazec u patriaci do jazyka L . V prípade použitia pravidla $S' \rightarrow S'S'$ pred pravidlom $S' \rightarrow [S']$ vieme generovať ďalší reťazec na pravej strane čo odpovedá reťazcu v generovaným G_D ktorý patrí do L , t.j. vieme generovať za $[u]$ reťazec v patriaci do jazyka L za pomoci G_D . Pri viacnásobnom použití pravidla $S' \rightarrow S'S'$ toto tvrdenie stále platí, pretože za $[u]$ sa vezme najľavejšia derivácia $S' \rightarrow [S']$ a v bude reprezentované pravou časťou od symbolu $]$.

Takže, každé slovo $w \in L$, pre ktoré platí že $w \neq \varepsilon$, vieme zapísať v tvare $[u]v$ kde $u, v \in L$ pretože

$$S' \xRightarrow{G_D} S'S' \xRightarrow{G_D} [S']S' \xRightarrow{G_D}^* [u]v$$

Keďže $S' \xRightarrow{G_D}^* u$ a $S' \xRightarrow{G_D}^* v$ ktoré patria do jazyka L , tak $S' \xRightarrow{G_D}^* [u]v$ tiež patrí do jazyka, keďže $u, v \in L$ a $[,] \in L$. Je zrejmé, že ak $S' \xRightarrow{G_D}^* w$ a $S' \xRightarrow{G_D}^* [u]v$ tak potom platí že $S' \xRightarrow{G_D}^* w = [u]v$ t.j. $w = [u]v$.

1.2 (b)

Báza

Báza je bazový prípad **pre $i = 0$** .

Pre $i = 0$ platí, že počet $[$ a počet $]$ v reťazci w je rovno nule z čoho vyplýva, že reťazec w sa musí rovnať ε . Formálne, pre $i = 0$ platí, že $\#_[(w) = 0 \wedge \#_](w) = 0$ z čoho vyplýva, že $w = \varepsilon$.

Keďže, existuje pravidlo $S' \rightarrow \varepsilon$ v gramatike G_D a taktiež, existuje pravidlo $S \rightarrow \varepsilon$ v gramatike G , potom existujú derivácie $S' \xRightarrow{G_D} \varepsilon$ a $S \xRightarrow{G} \varepsilon$ a tak platí, že $w = \varepsilon \wedge w \in L \wedge w \in L(G)$.

Indukčný predpoklad

Predpokladáme, že tvrdenie platí pre všetky $w \in L$ kde $\#_[(w) < i \wedge \#_](w) < i$ t.j. kde $\#_[(w) \leq j \wedge \#_](w) \leq j$ kde $j < i$

Pre i

$\#_{\lceil}(u) + \#_{\lceil}(v) \stackrel{?}{=} i$ analogicky pre $\#_{\rceil}(u) + \#_{\rceil}(v) \stackrel{?}{=} i$

1.) $\#_{\lceil}(u) = 0 \Rightarrow \#_{\lceil}(v) = i \quad \vee \quad \#_{\lceil}(v) = 0 \Rightarrow \#_{\lceil}(u) = i$

Ak je buď $\#_{\lceil}(u)$ alebo buď $\#_{\lceil}(v)$ rovné nule, tak ho vieme vygenerovať z pravidla S na základe indukčnej bázi.

Ak je buď $\#_{\lceil}(u)$ alebo buď $\#_{\lceil}(v)$ rovné i , tak ten prvok prepíšeme pomocou vzorca

$$w' = [u']v' \Rightarrow \#_{\lceil}(u') + \#_{\lceil}(v') = i - 1 = j, \text{ analogicky } \#_{\rceil}(u') + \#_{\rceil}(v') = i - 1 = j$$

Na základe indukčného predpokladu vieme z S vygenerovať u' a v' .

$$S \Rightarrow [S]S \Rightarrow^* [u']v' = w'$$

kde

$$\#_{\lceil}(u') + \#_{\lceil}(v') = i, \text{ analogicky } \#_{\rceil}(u') + \#_{\rceil}(v') = i$$

2.) $\#_{\lceil}(u) \neq 0 \quad \wedge \quad \#_{\lceil}(v) \neq 0 \quad \wedge \quad \#_{\lceil}(u) + \#_{\lceil}(v) \leq i$

Keď $\#_{\lceil}(u)$ a $\#_{\lceil}(v)$ sú nenulové, tak musí platiť že

$$\#_{\lceil}(u) < i \wedge \#_{\lceil}(v) < i$$

t.j.

$$\#_{\lceil}(u) = i - M = j_1 \wedge \#_{\lceil}(v) = i - N = j_2 \text{ kde } M, N \in \mathbb{N} \setminus \{0\}$$

Pre i + 1

$S \Rightarrow^* w$ kde $w \in L$ pre ktoré platí, že

$$\#_{\lceil}(w) = i + 1$$

a z definície Dyckovho jazyka platí, že

$$\#_{\rceil}(w) = i + 1$$

Potom vieme w zapísať ako $w = [u]v$ podľa bodu (a) (kapitola 1.1) $\Rightarrow \#_{\lceil}(u) + \#_{\lceil}(v) = i$, analogicky musí platiť $\#_{\rceil}(u) + \#_{\rceil}(v) = i$.

$S \Rightarrow [S]S \Rightarrow^* [u]v = w$ kde $\#_{\lceil}(u) + \#_{\lceil}(v) = i + 1$, analogicky $\#_{\rceil}(u) + \#_{\rceil}(v) = i + 1$.

2 Príklad číslo 2

Veta 4.19 [1](str. č. 92): Nechť L je bezkontextový jazyk. Pak existuje konstanta $k > 0$ taková že je-li $z \in L$ a $|z| \geq k$, pak lze z napsat ve tvaru:

$$z = uvwxy, vx \neq \varepsilon, |vwx| \leq k$$

a pro všechna $i \geq 0$ je $uv^iwx^iy \in L$.

Nech L_{primes} je bezkontextový jazyk.

Tak existuje celočíselná konstanta $k > 0$ taká, že ak $z \in L$ a $|z| \geq k$, tak

$$z = uvwxy \wedge vx \neq \varepsilon \wedge |vwx| \leq k \wedge uv^iwx^iy \in L \text{ kde } i \geq 0$$

Zvoľme prvočíslo r väčšie ako k t.j. $r \geq k$ kde r je prvočíslo.

Potom platí, že

$$a^r \in L \wedge |a^r| = r \text{ kde } r \geq k \implies a^r = uvwxy \wedge vx \neq \varepsilon \wedge |vwx| \leq k \wedge uv^iwx^iy \in L \text{ pre } i \geq 0$$

Nech

$$\begin{aligned} v = a^m &\Rightarrow |v| = m \\ x = a^n &\Rightarrow |x| = n \\ w = a^o &\Rightarrow |w| = o \end{aligned}$$

Tak musí platiť že $m + n > 0$ pretože $vx \neq \varepsilon$ a $k \geq m + n + o$ pretože $|vwx| \leq k$.

Zvoľme $i = r + 1$, potom

$$uv^{r+1}wx^{r+1}y \in L$$

$$|uv^{r+1}wx^{r+1}y| = |uvwxy| + |v^r| + |x^r| = r + r \cdot m + r \cdot n = r \cdot (1 + m + n) \text{ čo nie je prvočíslo}$$

A z toho vyplýva spor pretože

$$uv^{r+1}wx^{r+1}y \notin L$$

Takže jazyk L_{primes} nie je bezkontextový jazyk.

3 Príklad číslo 3

3.1 Nerozhodnuteľnosť

Problém môžeme charakterizovať jazykom L pre ktorý platí

$$L = \{ \langle M_L \rangle \mid M_L \text{ je } TS : \exists w \in \text{Affine} : w \in L(M_L) \}$$

Problém členstva je charakterizovaný jazykom MP pre ktorý platí

$$MP = \{ \langle M_{MP} \rangle \# w \mid M_{MP} \text{ je } TS \text{ ktorý prijme } w \}$$

Zostavíme redukciu

$$\sigma : \{0, 1, \#\}^* \longrightarrow \{0, 1\}^* \text{ z jazyka } MP \text{ na } L$$

$TS M_\sigma$ implementujúci σ priradí každému vstupu $x \in \{0, 1, \#\}^*$ reťazec $\langle M_x \rangle$, kde M_x je TS , ktorý na vstupe $y \in \{0, 1\}^*$ pracuje nasledovne:

1. M_x zmaže svoj vstup y .
2. Zapiše na pásku reťazec x .
3. M_x posúdi, zda $x = x_1 \# x_2$ pre x_1 , ktorý je kódom TS , a x_2 , ktorý je kódom jeho vstupu. Pokiaľ nie, odmietne.
4. Inak M_x simuluje činnosť TS s kódom x_1 na reťazci s kódom x_2 .
 - Ak x_1 prijme x_2 , tak M_x prijme.
 - Ak x_1 odmietne x_2 , tak M_x odmietne.
 - Inak cyklí.

M_σ je možné implementovať úplným TS . Konečne tento TS vypíše kód M_x , ktorý sa skladá zo štyroch komponent, ktoré odpovedajú vyššie uvedeným krokom. Tri z nich sú pritom konštantné (nezávisia na x) – konkrétne (1) zmazanie pásky, (2) test na dobré sformovanie instance MP a (3) simulácia daného TS na danom vstupe (pomocou úplného TS). TS implementujúci tieto kroky, ktoré evidentne existujú, môžeme pripraviť vopred a M_σ vypíše kód spolu s kódom na predanie riadenia. Zostáva vygenerovať kód TS , ktorý zapiše na pásku dané $x = a_1 a_2 \dots a_n$. To je možné ale ľahko realizovať pomocou $TS Ra_1 Ra_2 R \dots Ra_n$.

Skúmame možné jazyky $TS M_x$:

- $L(M_x) = \emptyset \iff (x \text{ nie je správne sformovaná instance } MP) \text{ alebo } (x = x_1 \# x_2 \text{ a } TS \text{ s kódom } x_1 \text{ na reťazci s kódom } x_2 \text{ odmietne}) \text{ alebo } (x = x_1 \# x_2 \text{ a } TS \text{ s kódom } x_1 \text{ na reťazci s kódom } x_2 \text{ neskončí t.j. cyklí})$
- $L(M_x) = \Sigma^* \iff (x \text{ je správne sformovaná instance } MP, \text{ kde } x = x_1 \# x_2 \text{ a } TS \text{ s kódom } x_1 \text{ na reťazci s kódom } x_2 \text{ prijme})$

Ak $L(M_x) = \Sigma^*$ je zrejmé, že jazyk $L(M_x)$ iste obsahuje aspoň jeden reťazec ktorý patrí do jazyka $Affine$.

Teraz už ľahko ukážeme, že σ zachováva členstvo $\langle M_x \rangle \in L \iff L(M_x) = \Sigma^* \iff x = x_1 \# x_2$ kde x_1 je kód TS , ktorý zastaví na vstupe s kódom $x_2 \iff x \in MP$.

3.2 Čiastočná rozhodnuteľnosť

Majme $TS\ M$ pre ktorý platí, že $\exists w \in Affine : w \in L(M)$ kde jazyk $Affine$ je rekurzívny jazyk.

K čiastočnému rozhodovaniu uvedeného problému môžeme využiť $TS\ M'$ ktorý na svojej prvej páske simuluje beh $TS\ M$ pre jednotlivé možné vstupné reťazce podľa určitého usporiadania (viz dole uvedený príklad) a na druhej páske vykonáva kontrolu podmienky patričnosti reťazca do jazyka $Affine$.

$konf_M(\varepsilon, 0)$
 $konf_M(\varepsilon, 1) \# konf_M(0, 0)$
 $konf_M(\varepsilon, 2) \# konf_M(0, 1) \# konf_M(1, 0)$
 $konf_M(\varepsilon, 3) \# konf_M(0, 2) \# konf_M(1, 1) \# konf_M(00, 0)$
 $konf_M(\varepsilon, 4) \# konf_M(0, 3) \# konf_M(1, 2) \# konf_M(00, 1) \# konf_M(01, 0)$
...

M' nemôže len systematicky generovať vstupy pre $TS\ M$ a na nich nechať $TS\ M$ neobmedzene bežať pretože hrozí zacyklenie.

$TS\ M'$ ale môže mať na svojej páske súčasne rozbehnutú simuláciu $TS\ M$ pre ľubovoľný počet vstupných reťazcov, jednotlivé konfigurácie pásky budú vhodne oddelené.

$TS\ M'$ môže vždy prejsť všetky aktuálne rozbehnuté simulácie a na každej vykonať práve jeden krok. Pokiaľ v niektorom prípade dôjde k prijatiu reťazca $TS\ M$, vykoná kontrolu patričnosti prijatého reťazca do jazyka $Affine$ na druhej páske. Ak kontrola patričnosti prijatého reťazca skončila úspechom, t.j. reťazec bol prijatý $TS\ M$ a súčasne patrí do jazyka $Affine$ môžeme tvrdiť, že existuje reťazec ktorý patrí do $Affine$ a súčasne do $L(M)$. Ak $TS\ M$ neprijal reťazec alebo reťazec nepatrí do jazyka $Affine$, tak $TS\ M'$ pridá páskovú konfiguráciu pre ďalší reťazec a celý krok opakuje.

$TS\ M'$ vykonáva kontrolu patričnosti reťazca do jazyka $Affine$ na druhej páske tak, že **TODO**.

Funkčnosť $TS\ M'$ môžeme chápať tak, že najskôr $TS\ M'$ spustí simuláciu $TS\ M$ na prvej páske. Ak $TS\ M$ zastaví a prijme určitý reťazec, tak tento reťazec $TS\ M'$ overí na druhej páske zda patrí do jazyka $Affine$ t.j. skontroluje podmienku patričnosti tohto reťazca do jazyka $Affine$.

4 Príklad číslo 4

4.1 (a)

Dôkaz, že pre každý $TS\ M$ nad abecedou $\{0,1\}$ a reťazec $w \in \{0,1\}^*$ je možné zostrojiť program P_M v jazyku *RationalC* a zvoliť počiatočnú hodnotu x_0 pre register x tak, že P_M skončí s návratovou hodnotou 1 práve vtedy keď $w \in L(M)$.

Uvažujme $w = 101110111$.

Prvým krokom je potreba reverzovať reťazec w na páske.

$$\begin{array}{l|l|l} \text{páska s } w & \Delta w \Delta^\omega & \Delta 101110111 \Delta \Delta \Delta \dots \\ \text{páska s } w^R & \Delta w^R \Delta^\omega & \Delta 111011101 \Delta \Delta \Delta \dots \end{array}$$

Druhým krokom je potreba tento reverzovaný reťazec w^R na páske spolu so začiatkom pásky prekódovať (do registra je možné ukladať iba hodnoty 0 a 1) za pomoci pravidiel uvedených dole kde symbol *EOT*¹ znázorňuje *End Of Tape* (koniec pásky).

$$\begin{array}{l} \text{páskový symbol} \rightarrow \langle \text{infobit} \rangle \langle \text{valuebit} \rangle \\ \Delta \rightarrow 00 \\ \text{EOT} \rightarrow 01 \\ 1 \rightarrow 11 \\ 0 \rightarrow 10 \end{array}$$

Symbol *EOT* je potrebný z dôvodu detekcie prepadnutia hlavy. Páskový symbol je reprezentovaný dvoma bitmi kde *infobit* udáva zda sa jedná o hodnotu numerickú (v prípade pre 1) alebo o blank, *EOT* (v prípade pre 0).

$$\begin{array}{l|l|l} \text{páska s } w^R & \Delta w^R \Delta^\omega & \Delta \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ \Delta \ \Delta \ \Delta \dots \\ \text{časť ktorá sa bude kódovať} & \text{EOT} \Delta w^R & \text{EOT} \ \Delta \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \\ \text{zakódovaná časť pásky} & \text{encoded}(\text{EOT} \Delta w^R) & 01 \ 00 \ 11 \ 11 \ 11 \ 10 \ 11 \ 11 \ 11 \ 10 \ 11 \end{array}$$

Do registra sa bude ukladať zakódovaná časť pásky tak, že v registri bude *LSB* (*Least Significant Bit*) značiť symbol *EOT* (*End Of Tape*) (do registra sa to ukladá opačne ale dvojice sa neotáčajú).

$$\begin{array}{l|l} \text{zakódovaná časť pásky} & 01 \ 00 \ 11 \ 11 \ 11 \ 10 \ 11 \ 11 \ 11 \ 10 \ 11 \\ \text{hodnota v registri } x & 11 \ 10 \ 11 \ 11 \ 11 \ 10 \ 11 \ 11 \ 11 \ 00 \ 01 \end{array}$$

Takže máme v registri x zapísanú hodnotu kde je ale desatinná bodka na nesprávnom mieste ktorá je znázornená červenou farbou.

$$\text{register } x \mid 11 \ 10 \ 11 \ 11 \ 11 \ 10 \ 11 \ 11 \ 11 \ 00 \ 01.$$

Aby sme desatinnú čiarku presunuli na správne miesto, tak bude treba využiť inicializačnú programovú rutinu ktorá vykoná nasledujúci podprogram, v ktorom dva krát vykoná príkaz $x \neq 2$. Tým pádom sa register nastaví a jeho hodnota bude reprezentovaná ako

$$\text{register } x \mid 11 \ 10 \ 11 \ 11 \ 11 \ 10 \ 11 \ 11 \ 11 \ 00. \ 01$$

¹Pre zlepšenie čitateľnosti bude využitá zelená farba k značeniu *EOF*.

Inicializačný kód:

```
x /= 2  
x /= 2
```

Následne po správnej inicializácii registra sme schopný vykonávať jednotlivé činnosti $TS\ M$ pomocou určitých podprogramov. Samotný podprogram reprezentuje rutinu ktorá sa správa rovnako ako chovanie jednej činnosti $TS\ M$.

Rutiny pre jednotlivé činnosti $TS\ M$ môžeme rozdeliť na základe ich závislosti na posuvné, zápisové a overovacie. Posuvné realizujú posuv hlavy zo symbolu na symbol ktoré ale nekontrolujú EOT pretože prechod $TS\ M$ závisí od aktuálneho symbolu pod hlavou, ktorý sa kontroluje zda to nie je EOT v overovacích rutinách, ktoré realizujú čítanie symbolu pod hlavou. Každá posuvná a zápisová rutina je podmienená overovacou rutinou, ktorá vždy overí zda sa tam nenachádza EOT . Každá overovacia rutina obsahuje na začiatku kód pre overenie EOT pod ktorým sa nachádza kód pre overenie 0,1 alebo Δ . Ak overovacia rutina odhalí EOT tak beh programu je ukončený s návratovou hodnotou 0, ak neodhalí EOT symbol tak overí zda sa jedná o očakávaný symbol a ak sa jedná o očakávaný symbol tak sa vykoná posuvná alebo zápisová rutina. Overovacie rutiny obsahujú na konci $NOOP$; ktorý je možný realizovať za pomoci dvoch príkazov $x \neq 2$; $x \neq 2$; ktoré realizujú neutrálnu operáciu. Je to z toho dôvodu, že bolo potrebné pre overovaciu rutinu vytvoriť koncovú návesť.

Posuvné rutiny:

- Posun hlavny do ľava

```
A | x *= 2;  
B | x *= 2;
```

- Posun hlavy do prava

```
A | x /= 2;  
B | x /= 2;
```

Zápisové rutiny:

- Zápis symbolu 1

```
A | x /= 2;  
B | odd(x);  
C | x *= 2;  
D | odd(x);
```

- Zápis symbolu 0

```
A | x /= 2;  
B | odd(x);  
C | x *= 2;  
D | even(x);
```

- Zápis symbolu Δ

```
A | x /= 2;  
B | even(x);  
C | x *= 2;  
D | even(x);
```

Overovacie rutiny:

- Kontrola symbolu 1

<i>A</i>		if x % 2 == 0 goto F;
<i>B</i>		x /= 2;
<i>C</i>		if x % 2 == 1 goto E;
<i>D</i>		return 0;
<i>E</i>		x *= 2;
<i>F</i>		if x % 2 == 0 goto K;
<i>G</i>		x /= 2;
<i>H</i>		if x % 2 == 0 goto J;
<i>I</i>		rutina pre zápis alebo posuv
<i>J</i>		x *= 2;
<i>K</i>		NOOP;

- Kontrola symbolu 0

<i>A</i>		if x % 2 == 0 goto F;
<i>B</i>		x /= 2;
<i>C</i>		if x % 2 == 1 goto E;
<i>D</i>		return 0;
<i>E</i>		x *= 2;
<i>F</i>		if x % 2 == 1 goto K;
<i>G</i>		x /= 2;
<i>H</i>		if x % 2 == 0 goto J;
<i>I</i>		rutina pre zápis alebo posuv
<i>J</i>		x *= 2;
<i>K</i>		NOOP;

- Kontrola symbolu Δ

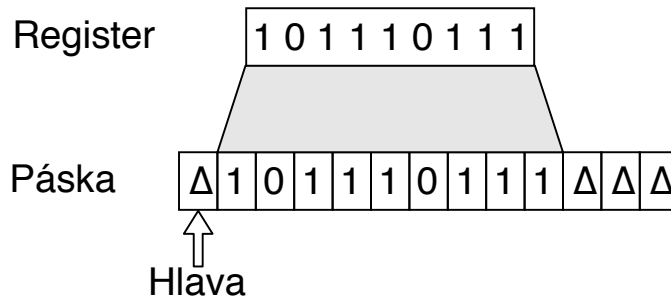
<i>A</i>		if x % 2 == 0 goto F;
<i>B</i>		x /= 2;
<i>C</i>		if x % 2 == 1 goto E;
<i>D</i>		return 0;
<i>E</i>		x *= 2;
<i>F</i>		if x % 2 == 1 goto K;
<i>G</i>		x /= 2;
<i>H</i>		if x % 2 == 1 goto J;
<i>I</i>		rutina pre zápis alebo posuv
<i>J</i>		x *= 2;
<i>K</i>		NOOP;

Všetky rutiny obsahujú na ľavej strane kódu písmená ktoré reprezentujú číslne označenie jednotlivých po sebe idúcich riadkov.

4.2 (b)

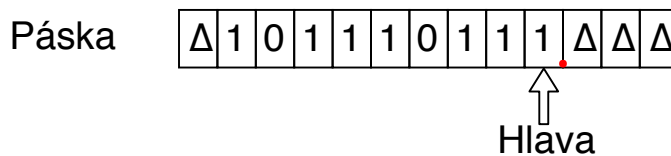
Dôkaz, že pre každý program P v jazyku *RationalC* a počiatočnú hodnotu $x_0 \in \mathbb{N}$ ($0 \in \mathbb{N}$) je možné zostrojiť $TS M_P = (Q, \Sigma, \Gamma, \delta, q_{first}, F)$ a reťazec $w \in \{0, 1\}^*$ tak, že $w \in L(M_P)$ práve vtedy, keď P s počiatočnou hodnotou x_0 skončí s návratovou hodnotou 1.

Na začiatku si zapíšeme obsah registra x v ktorom sa nachádza číslo $x_0 \in \mathbb{N}$ v binárnej podobe na pásku $TS M_P$ čo je zobrazené v ilustračnom obrázku 1.



Obr. 1: Ilustračný obrázok prevodu registra x na pásku $TS M_P$.

Potom, pre $TS M_P$ je potreba inicializovať hlavu čo v našom prípade znamená, že posunieme hlavu $TS M_P$ na *least significant bit (LSB)* t.j. na *most right nonblank symbol* (najpravejší neblankový symbol) čo je realizovateľné TS . Táto inicializácia nášho $TS M_P$ je nutná z toho dôvodu, že na začiatku máme celé nezáporné číslo ale neskôr sa môžeme dopracovať ku desatinným číslam kde je potreba vedieť kde je desatinná čiarka. V $TS M_P$ budeme desatinnú čiarku reprezentovať tak, že pozícia hlavy ukazuje na *LSB* čísla z čoho vieme, že desatinná bodka sa nachádza na pravej strane od tohto *LSB* čísla. Proces inicializácie hlavy je ilustrovaný na obrázku 2 v ktorom je hypotetická desatinná bodka znázornená červenou bodkou.



Obr. 2: Ilustračný obrázok inicializácie hlavy $TS M_P$ na páske.

Predpokladajme, že zdrojový kód pre program P v jazyku *RationalC* má určitú formu kde každý jeden príkaz je zapísaný na osobitnom riadku kde všetky riadky sú očíslované od 0 (pre prvý riadok) až po N (posledný riadok).

Potom, pre každý jeden riadok, kde označíme číslo riadka ako N , vytvoríme jeden stav $q_N \in Q$ a pre každý príkaz na týchto riadkoch platí, že jednotlivé príkazy vieme previesť podľa nasledujúcich pravidiel.

- Príkaz **if** $x \% 2 == A$ **goto** B kde $A = \{0, 1\}$ a $0 \leq B \leq N_{MAX}$ na riadku N

$TS\ M_P$ bude obsahovať nasledujúce prechodové pravidlá pre tento príkaz

$$\begin{aligned}\delta(q_N, A) &= (q_B, A) \\ \delta(q_N, \bar{A}) &= (q_{N+1}, \bar{A})\end{aligned}$$

kde $q_B, q_N, q_{N+1} \in Q$.

Tento príkaz je možné chápať ako zmena stavu v $TS\ M_P$ pričom sa hlava neposunie a ponechá sa pôvodný symbol pod pozíciou hlavy.

- Príkaz $x \neq 2$ na riadku N

$TS\ M_P$ bude obsahovať nasledujúce prechodové pravidlá pre tento príkaz

$$\begin{aligned}\delta(q_N, 0) &= (q_{N+1}, L) \\ \delta(q_N, 1) &= (q_{N+1}, L)\end{aligned}$$

kde $q_N, q_{N+1} \in Q$.

Tento príkaz je možné chápať ako posun hlavy $TS\ M_P$ do ľava.

- Príkaz $x \neq 2$ na riadku N

$TS\ M_P$ bude obsahovať nasledujúce prechodové pravidlá pre tento príkaz

$$\begin{aligned}\delta(q_N, 0) &= (q_{N+1}, R) \\ \delta(q_N, 1) &= (q_{N+1}, R)\end{aligned}$$

kde $q_N, q_{N+1} \in Q$.

Tento príkaz je možné chápať ako posun hlavy $TS\ M_P$ do prava.

- Príkaz **return** 0 na riadku N

$TS\ M_P$ bude obsahovať nasledujúce prechodové pravidlá pre tento príkaz

$$\begin{aligned}\delta(q_N, 0) &= (q_{reject}, 0) \\ \delta(q_N, 1) &= (q_{reject}, 1)\end{aligned}$$

kde $q_N, q_{reject} \in Q$ a $q_{reject} \in F$.

Tento príkaz je možné chápať ako prechod $TS\ M_P$ do koncového stavu ktorý je odmietajúci.

- Príkaz **return** 1 na riadku N

$TS\ M_P$ bude obsahovať nasledujúce prechodové pravidlá pre tento príkaz

$$\begin{aligned}\delta(q_N, 0) &= (q_{accept}, 0) \\ \delta(q_N, 1) &= (q_{accept}, 1)\end{aligned}$$

kde $q_N, q_{accept} \in Q$ a $q_{accept} \in F$.

Tento príkaz je možné chápať ako prechod $TS\ M_P$ do koncového stavu ktorý je akceptujúci.

- Príkaz `odd(x)` na riadku N

$TS\ M_P$ bude obsahovať nasledujúce prechodové pravidlá pre tento príkaz

$$\begin{aligned}\delta(q_N, 0) &= (q_{N+1}, 1) \\ \delta(q_N, 1) &= (q_{N+1}, 1)\end{aligned}$$

kde $q_N, q_{N+1} \in Q$.

Tento príkaz je možné chápať ako zápis symbolu 1 na aktuálnej pozícii hlavy.

- Príkaz `even(x)` na riadku N

$TS\ M_P$ bude obsahovať nasledujúce prechodové pravidlá pre tento príkaz

$$\begin{aligned}\delta(q_N, 0) &= (q_{N+1}, 0) \\ \delta(q_N, 1) &= (q_{N+1}, 0)\end{aligned}$$

kde $q_N, q_{N+1} \in Q$.

Tento príkaz je možné chápať ako zápis symbolu 0 na aktuálnej pozícii hlavy.

Pri násobení a delení (pri použití príkazov $x /= 2$ a $x *= 2$) môže ale dôjsť ku tomu, že sa hlava dostane zo symbolu 0 alebo 1 na symbol blank. Môžu nastať dve možnosti a to keď sa hlava dostaneme na ľavý blank (prvý symbol pásky) alebo na pravý blank (blank za posledným znakom 0 alebo 1 ktorý je uložený na páske). Tieto prípady je potreba vyriešiť pričom môžu nastať iba vtedy, ak sme sa dostali na krajný (najpravejší alebo najľavejší) symbol 0 alebo 1 vedľa ktorého je z ľavej strany blank (pre najľavejší) alebo z pravej strany blank (pre najpravejší). Takže môžu nastať dva prípady.

- Hlava je na najpravejšom symbole 0 alebo 1 a treba vykonať príkaz $x *= 2$.

Príkladový prípad hlavy a pásky

$\Delta 10\Delta\Delta\Delta \dots$

Po vykonaní príkazu $x *= 2$ sa hlava dostane z najpravejšieho neblankového symbolu na pravý blankový symbol.

$\Delta 10\Delta\Delta\Delta \dots$

V hore uvedenom prípade by sme mali chybnú hodnotu čísla z toho dôvodu, že u nás pozícia hlavy určuje *LSB* a tým pádom by sme mali nevalidné číslo keďže jeho *LSB* by bol blankový symbol. Aby sme sa vyhli tejto situácii, pri každom posune doprava treba vykonať kontrolu zda tam je blankový symbol, ak tam je, tak ho prepíšeme na symbol 0 a posunieme hlavu na tento symbol 0, ak tam nie je, tak iba presunieme hlavu na ten symbol ktorým je 0 alebo 1.

Páska	Hodnota	Riadok	Nasledujúci príkaz
$\Delta 10\Delta\Delta\Delta \dots$	1	n	$x *= 2$
$\Delta 10\Delta\Delta\Delta \dots$	2	$n + 1$	$x *= 2$
$\Delta 100\Delta\Delta\Delta \dots$	4	$n + 2$	$x *= 2$
$\Delta 1000\Delta\Delta\Delta \dots$	8	$n + 3$	$x *= 2$

V tabuľke stĺpec '*páska*' zachytáva aktuálny stav pásky spolu s pozíciou hlavy čo je znázornené podčiarknutým červeným symbolom. Stĺpec '*hodnota*' vyjadruje numerickú hodnotu ktorá sa nachádza na páske, '*riadok*' udáva pozíciu v programe a '*nasledujúci príkaz*' zobrazuje príkaz ktorý bude vykonaný.

- Hlava je na najľavejšom symbole 0 alebo 1 a treba vykonať príkaz $x \neq 2$.

Príkladový prípad hlavy a pásky

$\Delta \underline{1}0\Delta\Delta\Delta \dots$

Po vykonaní príkazu $x \neq 2$ sa hlava dostane z najľavejšieho neblankového symbolu na ľavý blankový symbol.

$\underline{\Delta}10\Delta\Delta\Delta \dots$

Ak by sme znovu vykonali tento príkaz, tak by došlo k abnormálnemu zastaveniu $TS M_P$ pretože by nám spadla hlava. Tento prípad je možné riešiť tak, že ak sme na tomto ľavom blankovom symbole a treba vykonať príkaz pre posun do ľava t.j. vykonaným príkazom bude príkaz $x \neq 2$, tak sa celý obsah pásky posunieme o jeden symbol do prava (vykonáme S_R) a pred obsah pásky zapíšeme symbol 0 pričom hlavu musíme vrátiť na pôvodné miesto – na miesto ľavého blanku. Pre ilustráciu funkčnosti viz tabuľku dole.

Páska	Hodnota	Riadok	Nasledujúci príkaz
$\Delta \underline{1}0\Delta\Delta\Delta \dots$	2	n	$x \neq 2$
$\Delta \underline{1}0\Delta\Delta\Delta \dots$	1	$n + 1$	$x \neq 2$
$\underline{\Delta}10\Delta\Delta\Delta \dots$	0.5	$n + 2$	$x \neq 2$
$\underline{\Delta}010\Delta\Delta\Delta \dots$	0.25	$n + 3$	$x \neq 2$
$\underline{\Delta}0010\Delta\Delta\Delta \dots$	0.125	$n + 4$	$x \neq 2$

V tabuľke stĺpec 'páska' zachytáva aktuálny stav pásky spolu s pozíciou hlavy čo je znázornené podčiarknutým červeným symbolom. Stĺpec 'hodnota' vyjadruje numerickú hodnotu ktorá sa nachádza na páske, 'riadok' udáva pozíciu v programe a 'nasledujúci príkaz' zobrazuje príkaz ktorý bude vykonaný.

Na základe tohto hore uvedeného popisu je možné zostrojiť pre program P v jazyku *RationalC* s počiatočnou hodnotou x_0 uloženou v registri x ekvivalentný $TS M_P$ a reťazec $w \in \{0,1\}^*$ tak, že $w \in L(M_P)$ práve vtedy keď program P s počiatočnou hodnotou x_0 skončí s návratovou hodnotou 1.

5 Literatúra

- [1] M. Češka, T. Vojnar, A. Smrčka, A. Rogalewicz: *Teoretická informatika - Studijní text*. 2018-08-23, [Online; Accessed: 2018-10-15].
URL: <http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/TIN-studijni-text.pdf>