TIN - Domáca úloha č. 2

Roman Dobiáš - xdobia11@stud.fit.vutbr.cz

3. decembra 2018

Úloha č.1

Podľa definície 4.29 (opora, str. 97) je Dyckov jazyk nad jednou dvojicou [] generovaný gramatikou s nasledujúcimi pravidlami, ktorú označme G_D:

$$S \to [S]|SS|\epsilon \tag{1}$$

Pre túto gramatiku zjavne platí, že každá postupnosť derivácii vedie ku terminálnemu reťazcu.

Časť A

V tejto časti ukážeme, že každý neprázdny reťazec $w \in L$ je možné vyjadriť v tvare $w = [u]v$, kde $u, v \in L$.

Uvažujme neprázdny reťazec $w \in L$. Uvažujme prvú deriváciu štartujúceho nonterminálu S v gramatike G_D. Potom podľa definície gramatiky G_D sú 3 možnosti, ktoré pravidlo mohlo byť použité v prvej derivácii pri generovaní reťazca w:

1. Pravidlo $S \to \epsilon$
   Potom $S \Rightarrow \epsilon$, teda $w = \epsilon$, čo je v spore s predpokladom, že w je neprázdny reťazec. Pravidlo teda nemôže byť použité v prvej derivácii.

2. Pravidlo $S \to [S]$
   Potom $S \Rightarrow [S] \Rightarrow^+ w$. Potom je reťazec w nutne konkatenáciou tvaru $[u]v$, kde $u \in L \land v = \epsilon \land \epsilon \in L$, pretože S je počiatočný nonterminál gramatiky G_D, teda derivuje slovo $u \in L$. Zároveň zjavne platí, že $|w| = 2 + |u|$ a $\#_[(w) = \#_[(u) + 1$.

3. Pravidlo $S \to SS$
   Vieme, že w je neprázdne. V gramatike G_D obsahuje len jedno pravidlo terminálne symboly, preto zjavne pravidlo $S \to [S]$ musí byť použité aspoň raz v postupnosti derivácii vetnej formy SS. Nutne teda platí, že z vetnej formy SS deriváciami získame vetnú formu S^N. Zjavne aspoň jedno S musí byť prepísané pravidlom $S \to [S]$. Zaujíma nás ten najľavejší prepísaný nonterminál S. Potom vzniknutá vetná forma má tvar $[S]S^{N-1}$. Zjavne platí, že $S \Rightarrow SS \Rightarrow SSS \Rightarrow^* S^{N-1} \Rightarrow v, v \in L$. Teda reťazec w je konkatenáciou $[u]v$, kde $S \Rightarrow^* u$ a $S \Rightarrow^* S^N \Rightarrow^* v$. Dokopy $S \Rightarrow SS \Rightarrow [S]S \Rightarrow^+ [u]S \Rightarrow^* [u]S^{N-1} \Rightarrow^* [u]v$. Zjavne platí, že $|w| = 2 + |u| + |v|$.

Ukázali sme teda, že každý neprázdny reťazec $w \in L$ je možné rozložiť na $[u]v$, kde $u, v \in L$ a zároveň počet znakov [je v u i v ostro menší ako vo w.

Časť B

Dôkaz. Dokazujeme, že $\forall i \in \mathbb{N}_0 : \forall w \in L : \#_[(w) = i \implies w \in L(G)$. V dôkaze využívame gramatiku G_D, definovanú vyžšie (1). Pokiaľ derivácia nie je označená konkrétnou gramatikou, implicitne uvažujeme gramatiku G zo zadania úlohy.

Pre $i = 0$ tvrdenie platí, pretože epsilon je jediný reťazec v jazyku L taký, že $\#_[(\epsilon) = 0$, zároveň platí, že $S \Rightarrow_{G_D} \epsilon$, teda $\epsilon \in L$ a tiež $S \Rightarrow_G \epsilon$, teda $\epsilon \in L(G)$.

Predpokladajme, že tvrdenie platí pre $j < i$ a uvažujme platnosť tvrdenia $\forall w \in L : \#_{[}(w) = i \implies w \in L(G)$ pre $i = j + 1$.

V a) sme ukázali, že pre $\forall w \in L : \exists u, v \in L : w = [u]v$. Teda isto platí, že $\forall w \in L \wedge \#_{[}(w) = i : \exists u, v \in L : w = [u]v$ a platí, že $\#_{[}(u) < i$ a zároveň $\#_{[}(v) < i$, pretože konkatenácia $[u]v$ pridáva reťazcu w nutne o jeden symbol [viac. Teda nutne $u, v \in L(G)$, pretože pre všetky $w \in L, \#_{[}(w) < i$ je veta už dokázaná. Zároveň v gramatike G existuje pravidlo $S \rightarrow [S]S$, a keďže $u, v \in L(G)$, potom nutne existujú postupnosti derivácii $S \Rightarrow^* u$ a $S \Rightarrow^* v$.

Dohromady v G existuje derivácia $S \Rightarrow [S]S \Rightarrow^* [u]S \Rightarrow^* [u]v$. Potom ale $w = [u]v$ nutne patrí do L(G).

Tvrdenie teda platí pre $i = j + 1$, kde pre j je dokázané, teda platí pre $\forall i \in \mathbb{N}_0$. □

Úloha č.2

Dôkaz predvedieme pomocou Pumping Lemma pre bezkontextové jazyky. Dokážeme, že jazyk nie je bezkontextový.

Dôkaz. Nech L je bezkontextový jazyk nad abecedou Σ. Potom existuje konštanta $k > 0$ taká, že platí:

$$\forall z \in L \wedge |z| \geq k : \exists u, v, w, x, y \in \Sigma^* : z = uvwxy \wedge vx \neq \epsilon \wedge |vwx| \leq k \wedge \forall i \geq 0 : uv^i wx^i y \in L \qquad (2)$$

Nech existuje konštanta $k > 0$ a reťazec $z = a^p$, kde p je prvočíslo a zároveň $p > k$. Potom podľa Pumping Lemma platí, že:

$$\exists u, v, w, x, y \in \Sigma^* : z = uvwxy \wedge vx \neq \epsilon \wedge |vwx| \leq k \wedge \forall i \geq 0 : uv^i wx^i y \in L \qquad (3)$$

Uvažujme ľubovoľné konštanty $b, c, d \in \mathbb{N}_0$ také, že $a^b a^c \neq \epsilon$ a $v = a^b \wedge x = a^c \wedge w = a^d$ tak, že $|vwx| \leq k$ Zároveň zvoľme ľubovoľne uy tak, nech platí, že $z = uvwxy$. Plati, že $uvwxy = ua^b wa^c y \in L$.

Uvažujme iteráciu pumpovania $i = 1 + p$. Potom $|uv^i wx^i y| = |uv^{1+p} wx^{1+p} y| = |ua^{b*(1+p)} wa^{c*(1+p)} y| = |ua^{b+bp} wa^{c+cp} y| = |ua^b wa^c y| + |a^{bp}| + |a^{cp}| = p + bp + cp = p(1 + b + c)$. Podľa predpokladu má platiť, že $uv^{i+p} wx^{i+p} y \in L$, avšak dĺžka tohoto reťazca je $p(1 + b + c)$, čo je súčin prirodzených čísiel, teda nie je prvočíslom, teda $uv^{i+p} wx^{i+p} y \notin L$, čo je spor predpokladu. Jazyk $\{a^p|$ p je prvočíslo $\}$ nie je bezkontextovy. □

Úloha č.3

Najprv formálne definujme jazyky MP a jazyk Affine, medzi ktorými budeme definovať redukciu σ.

1. MP = {<M>#<w> | TS M s kódom <M> prijima reťazec s kódom <w>} $\subset \{0, 1, \#\}^*$
2. TuringAffine = {<M>| jazyk TS s kódom <M> obsahuje aspoň 1 reťazec z jazyka Affine} $\subset \{0, 1\}^*$

Nerozhodnuteľnosť problému *TuringAffine* dokážeme zavedením redukcie z *MP* na *TuringAffine*.

Idea redukcie

Pre kód <M>#<w> inštancie MP problému TS $M\sigma$ vygeneruje na výstupnú pásku TS Mx s kódom <Mx> taký, že:

1. TS Mx vymaže svoju vstupnú pásku a skopíruje kód <M>#<w>, ktorý má uložený v stavovom riadení na svoju pásku
2. TS Mx skontroluje či kód <M> je korektný kód TS. To je možné, nakoľko jazyk kódovania TS je regularným jazykom. V prípade, že kód <M> nie je validným kódom TS, TS Mx odmietne.
3. TS Mx pomocou UTS simuluje <M> na kóde <w>
4. Ak UTS cyklí, potom jazyk TS Mx je prázdny, teda neobsahuje reťazec z Affine (a <M>#<w> nepatrí do MP.)
5. Ak UTS zastaví a TS s kódom <M> prijal reťazec s kódom <w>, tak TS Mx prijime, inak odmietne.

Realizácia redukcie

- Kód univerzálneho TS M_{UTS}, ktorý na svojej páske v tvare $\Delta < M > \# < w > \Delta$ simuluje prijatie reťazca s kódom $< w >$ TS s kódom $< M >$, je konštantný literál, ktorý je nezávislý na obsahu vstupnej pásky a je možné vytvoriť úplný TS, ktorý vypíše kód univerzálneho TS na pásku.
- Kód TS M_{valid}, ktorý overuje, či kód TS <M> na páske je validne sformovaný, teda či <M> patrí do jazyka korektne zakódovaných TS, je literál TS simulujúceho beh konečného automatu. Tento kód je konštantný.
- Kód TS M_{erase}, ktorý vymaže svoju pásku, je konštantný literál.
- Pre každý reťazec $< M > \# < w >$ sme schopný vytvoriť kód TS M_{copy}, ktorý po spustení vypíše na svoju pásku reťazec $< M > \# < w >$.
- Nakoľko M_{UTS}, M_{valid} i M_{erase} sú konštantne dané TS, sme v stave vytvoriť kód TS M_{merged}, ktorý bude obsahovať tieto TS a pomocnú stavovú logiku, ktorá prijme ak M_{UTS} skončí.
- Nakoniec, sme v stave vytvoriť taký úplný TS $M\sigma$, ktorý pre svoj vstup $< M > \# < w >$ vypíše na výstup kód TS <Mx>, ktorý bude obsahovať spojený TS M_{merged}, TS M_{copy} a pomocné prechody medzi týmito TS.
- Redukcia je potom funkcia $\sigma : \{0, 1, \#\}^* \to \{0, 1\}^*$, ktorá je realizovaná úplnym TS $M\sigma$.

Korektnosť redukcie σ

Skúmajme jazyk $L(Mx)$:

1. $L(Mx) = \emptyset \iff$ <M> nie je korektný kód TS alebo TS <M> na slove <w> cyklí
2. $L(Mx) = \Sigma^* \iff$ <M> je správne sformulovaný kód TS a TS s kódom <M> prijíma kód <w>

Dokážeme, že redukcia je korektná.

1. <M>#<w> $\in MP \iff$ TS M je správne sformovaný a TS <M> príjme <w> $\iff L(Mx) = \Sigma^* \iff$ TS <Mx> taký, že jazyk L(Mx) obsahuje aspoň jeden reťazec z jazyka Affine $\iff$ <Mx> $\in$ TurringAffine

Nakoniec je nutné ukázať, že pre každú kombináciu konštant a_0 a a_1 je jazyk Affine vždy neprázdny, teda je možné nájsť aspoň jeden reťazec jazyka Affine. Ak by bol jazyk Affine prázdny, potom by nebolo možné redukciou vyjadriť oba prípady rozhodovacieho problému MP a redukcia by teda neexistovala.

Avšak, pre ľubovoľné konštantny a_0 a a_1 vždy existuje reťazec, ktorý patrí do daného jazyka Affine. Napríklad pre $w = 0^{a_1}$ vždy $w \in Affine$, pretože platí $a_0 \times a_1 + a_1 \times 0 - a_1 \times a_0 = 0$.

Dokázali sme teda, že existuje redukcia z MP, ktorý je nerozhodnuteľný, na problém TuringAffine. TuringAffine je teda nutne nerozhodnuteľný.

Idea čiastočnej rozhodnuteľnosti

Čiastočnú rozhodnuteľnosť je možné dokázať redukciou problému TuringAffine na iný rozhodovací problém, ktorý je práve čiastočne rozhodnuteľný, napr. problém neprázdnosti jazyka TS (problém NEP). Problém NEP je čiastočne vyčísliteľný podľa tvrdenia zo skript, str. 129.

Potom totiž platí vzťah $TurringAffine \le NEP$ a podla vety 6.3.1 (skripta, str. 128) platí, že ak NEP je rekurzívne vyčísliteľný, potom aj TurringAffine je rekurzívne vyčísliteľný, teda čiastočne rozhodnuteľný.

Následne si ukážeme, akým spôsobom by bola prevedená redukcia σ. Formálne sú rozhodovacie problémy nasledujúce:

1. TuringAffine = {<M>| jazyk TS s kódom <M> obsahuje aspoň 1 reťazec z jazyka Affine} $\subset \{0, 1\}^*$
2. NEP = {<M>| jazyk TS s kódom <M> obsahuje aspoň 1 reťazec (je neprázdny)} $\subset \{0, 1\}^*$

Redukcia $\sigma : \{0, 1\}^* \to \{0, 1\}^*$, ktorá každej inštancii z jazyka TuringAffine priradí inštanciu jazyka NEP, by bola realizovaná nasledujúco: Úplný TS realizujúci redukciu σ pre daný vstup má na svojom výstupe TS s kódom <Mx>, ktorý realizuje nasledujúcu činnosť:

1. TS Mx overí, či reťazec w na jeho vstupe patrí do jazyka Affine. Príslušnosť reťazca do jazyka Affine je realizovateľná pomocou TS - stačí spočítať počet symbolov 0 a 1 v reťazci w, previesť príslušné násobenie a odčítanie a porovnať výsledok voči nule. Konštanty a0 a a1 môže mať daný TS zakódované vo svojom stavovom riadení. Overenie, či reťazec patrí do jazyka Affine je rozhodnuteľný problém.

2. Ak reťazec w nepatrí do jazyka Affine, potom TS Mx odmietne.

3. Inak TS Mx posunie reťazec w, ktorý má zapísaný na páske, a skopíruje kód TS M <M>, čím na páske vznikne <M>#w

4. TS Mx overí, či TS s kódom <M> je korektne sformovaný TS. Ak nie odmietne.

5. Napokon, TS Mx s pomocou UTS simuluje prijatie reťazca w TS s kódom <M>. Ak TS s kódom <M> odmietne reťazec w alebo cyklí, potom TS Mx odmietne, inak prijme.

Skúmajme jazyk TS Mx:

1. $L(Mx) = \emptyset \iff$ TS s kódom <M> nie je správne sformovaný alebo TS <M> je správne sformovaný, ale L(M) neobsahuje ani jeden reťazec z jazyka Affine.

2. $L(Mx) \neq \emptyset \iff$ TS <M> je správne sformovaný a L(M) obsahuje aspoň jeden reťazec z jazyka Affine

Zjavne platí, že TS <M> obsahuje reťazec z jazyka Affine $\iff$ TS <M> je správne sformovaný a L(M) obsahuje aspoň jeden reťazec z jazyka Affine $\iff L(Mx) \neq \emptyset \iff$ TS s kódom <Mx> a jazyk L(Mx) je neprázdny.

Úloha č.4

Turingovú úplnosť programu *RacionalC* dokážeme konštruktívne. Ukážeme, že každý TS je možné zakódovať ako ekvivalentný program *RacionalC*, a naopak, že každý program *RacionalC* je možné vytvoriť ekvivalentný TS.

Pred popisom jednotlivých prevodov je nutné uvedomiť si, akým spôsobom sú si TS a *RacionalC* podobné. *RacionalC* pracuje s racionálnym číslom, pričom umožňuje toto číslo deliť a násobiť 2, prípadne nastavovať jeho párnosť, či nepárnosť, a rovnako disponuje príkazmi pre zmenu toku programu podľa párnosti. Efekt týchto operácii môžeme skúmať na binárnom kóde racionálneho čísla v registri x. Zjavne platí, že operácie deleno / násobenie 2 posúvajú desatinnú čiarku v binárnej reprezentácii čísla. Podobne operácia even / odd v skutočnosti len mení hodnotu bitu čísla x, ktorý leží tesne pred desatinnou čiarkou. Nakoniec, príkaz if modulo v skutočnosti zisťuje aktuálnu hodnotu bitu pred desatinnou čiarkou a podľa toho skáče v programe.

Ekvivalentné operácie ku spomenutým operáciam môžeme vidieť aj v TS. Uvažujme pásku automatu len nad symbolmi 0 a 1. Potom pozíciu čítacej hlavy na páske môžeme vnímať ako pozíciu desatinnej čiarky v čísle s kódom odpovedajúcemu páske. Operácie násobenie a delenie je možné realizovať v TS ako posun čítajúcej hlavy. Podobne, zápis symbolu pod čtecí hlavou je možné sémanticky vnímať ako operácie odd/even. Nakoniec, samotné prechody TS realizujú podmienené vykonávanie v závislosti na hodnote symbolu pod čtecí hlavou na páske.

Jednotlivé operácie je teda možné zakódovať ekvivalentne v TS aj v programe *RacionalC*.

V následujúcej časti predstavený príkladný spôsob zakódovania spolu s ošetrením špeciálneho chovania, akým je *prepadnutie hlavy TS*, či *nekonečné rozširovanie reprezentácie racionálneho čísla* do oboch smerov v programe *RacionalC*.

A - Prevod TS na RationalC

Podľa zadania prevádzame deterministický TS s abecedou $\Sigma = \{0, 1\}$ a páskovou abecedou $\Gamma = \Sigma \cup \{\Delta\}$. V prípade, že by pásková abeceda obsahovala ďalšie znaky, je možné TS previesť na TS s $\Gamma = \Sigma \cup \{\Delta\}$ pomocou zakódovania symbolov rozšírenej páskovej abecedy ako postupností symbolov z Γ (opora, dôkaz vety 6.1.1 str. 123).

Samotný prevod je definovaný ako prevod vstupnej pásky w TS a prevod samotného TS na program.

Prevod vstupnej pásky TS na číslo x_0

Vstupná páska TS má tvar $\Delta w \Delta$, kde $w \in \Sigma$. Keďže vieme, že program $RacionalC$ má na vstupe celé číslo x_0 a naopak TS začína na pozícii Δ, jednoduchou konverziou by sme získali číslo, ktoré má len desatinné miesta a prípadne jednotku. Preto musíme pred prevodom reťazec $\Delta w \Delta$ reverzovať. Následne je páska $\Delta w^R \Delta$ obsahujúca znaky $0, 1, \Delta$ je prevedená nasledujúco:

$$\Delta \implies 00 \qquad \# \implies 01 \qquad 0 \implies 10 \qquad 1 \implies 11$$

Každý symbol z Γ zakódujeme ako dvojicu bitov prirodzeného čísla $x0$. Kódovanie sme rozšírili o špeciálny symbol $\#$, ktorý nie je súčasťou páskovej abecedy. Tento špeciálny symbol je nutný pre korektné detekovanie stavu, kedy TS prepadla hlava.

Po prevedení symbolov reťazca w na takéto binárne kódovanie je výsledný reťazec interpretovaný ako binárne zakódované prirodzené číslo rozšírené o kód symbolu $\#$.

Príklad: $w = \Delta 011 \Delta$ sa po reverzácii na $\Delta 110 \Delta$ a pridaní zarážky, reprezentujúcej prepad hlavy rovná reťazcu $\Delta 110 \Delta \#$, čo sa zakóduje na 001110100001, teda na prirodzené číslo 929. Vďaka kódu 00 pre Δ obsahuje racionálne číslo nekonečno znakov reprezentujúcich Δ z prava aj z ľava pôvodonej pásky w, pretože dvojica 00 pridaná pred aj na koniec binárnej reprezentácie (pozn. za desatinnú čiarku) čísla nemení jeho význam.

Nakoľko sme oproti reprezentácii pásky TS posunuli desatinnú čiarku pridaním oddelovača $\#$, je nutné na začiatku programu *RacionalC* vykonať 2x príkaz $x/ = 2$; aby sme posunuli čiarku na správne miesto a program začal s desatinnou čiarkou pred symbolom $\#$, teda pred suffixom 01.

Prevod jednotlivých pravidiel TS na príkazy jazyka *RacionalC*

Jednotlivé stavy TS sú v *RationalC* reprezentované sekvenciou príkazov. Keďže jednotlivé symboly TS sú v RationalC reprezentované dvojicami bitov racionálneho čísla, potom posun hlavy je realizovaný 2x násobením alebo 2x delením, čo má za následok posun desatinnej čiarky o dvojicu bitov.

Realizácia koncového stavu je nasledujúca:

```
q_f: return 1;
```

Symboly q_f, q, q_0 a pod. označujú symbolicky číslo riadku, na ktorom začína daná sekvencia. Realizáciu nejakého nekoncového stavu q z TS je možné vyjadriť nasledujúcim pseudo kódom:

```
q:
q_0:   Over či symbol je rovný 0. Ak nie skáč na q_1
q_0+1: Vykonaj prechod definovaný pre d(q,0).
q_1:   Over či symbol je rovný 1. Ak nie skáč na q_D
q_1+1: Vykonaj prechod definovaný pre d(q,1).
q_D:   Over či symbol je rovný Delta. Ak nie skáč na q_#
q_D+1: Vykonaj prechod definovaný pre d(q,Delta).
q_#:   return 0 // pre # doslo k prepadnutiu hlavy
```

Poznámka: ak pre daný stav nie je prechod pre daný symbol z Γ v TS definovaný, potom je sekvencia "vykonaj prechod"nahradená jediným príkazom $return 0$;, ktorý odpovedá zastaveniu TS na nedefinovanom prechode.

Realizáciu jednotlivých overení pre každý zakódovaný symbol S z Γ môžeme vyjadriť nasledujúcou sekvenciou príkazov, kde X a Y je kód symbolu z prekódovania vyššie (napr. $0 = 10 \implies X = 1 \wedge Y = 0$).

```
// OVERENIE pre symbol S
q_S:   if x \% 2 == Y goto q_S+3    // zaciname na bite Y
q_S+1: GOTO(q_next)                 // ak Y nesedi, skusime iny symbol
q_S+3: x /= 2                       // posun na bit X
q_S+4: if x \% 2 == X goto q_S+7
q_S+5: x *= 2       // ak nesedi ani X, vratime hlavu na Y
q_S+6: GOTO(q_next) // nepodmienecny skok na overenie d'alšieho symbolu
q_S+7: x *= 2       // d'alej následuje vykonanie prechodu pre symbol S
```

Poznámka: Symbol **q_next** je nasledujúci symbol páskovej abecedy z Γ. Pre Δ je to nakoniec $\#$.

Poznámka: Makro **GOTO(state)** sa expanduje na sekvenciu príkazov `if x % 2 == 0 goto state; if x % 2 == 1 goto state`

Sekvencia príkazov odpovedajúca vykonaniu prechodu je potom závislá na konkrétnom type prechodu.

Pre **zápis symbolu** $x \in \Gamma$, ktorého kód je rovný XY a prechod do stavu q_{next} ide o nasledujúcu sekvenciu:

```
q_S_X:   WRITE(Y)
q_S_X+1: x /= 2
q_S_X+2: WRITE(X)
q_S_X+3: x *= 2
q_S+6:   GOTO(q_next)
```

Poznámka: Makro **WRITE(Z)** sa expanduje pre $Z = 0$ na $x = even(x)$, pre $Z = 1$ na $x = odd(x)$

Posun hlavy na páske smerom $Z = L, R$ a prechod na stav q_{next}:

```
q_S_X: SHIFT(Z)
q_S_X: SHIFT(Z)
q_S+6: GOTO(q_next)
```

Poznámka: Makro **SHIFT(Z)** sa expanduje pre $Z = R$ na $x\div = 2$, pre $Z = L$ na $x* = 2)$ Poznámka: Invertovanie posunu na páske TS voči posunu desatinnej čiarky vzniklo reverzovaním reťazca w.

Na začiatku program prevedie shift o 2 pozície do ľava, aby umiestnil destinnú čiarku pred symbol $\#$. Následne nasleduje vykonávanie sekvencie reprezentujúcej počiatočny stav TS.

Týmto spôsobom sme teda ukázali, že každý stav z TS a jeho prechody sme schopný zakódovať ekvivalentne v príkazoch *RationalC*. Zároveň program v *RationalC* korektne detekuje prepadnutie hlavy, zastavenie na nedefinovanom prechode pre daný stav (return 0) a konečne prijatie v stave q_f.

Príklad prevodu TS na program

Uvažujme TS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_f)$, kde $Q = \{q_0, q_1, q_f\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \Delta\}$, a δ: $\delta(q0, \Delta) = (q_1, R)$ $\delta(q1, 1) = (q_f, R)$. Tento TS prijíma len reťazce začínajúce znakom 1. Podľa postupu vyžšie prevedieme tento TS nasledujúco:

1. Na začiatku vygenerujeme posunutie desatinnej čiarky
2. Pre každý stav s vygenerujeme sekvenciu začínajúcu na riadku s, ktorá realizuje jeho prechody.
3. Pre koncový stav q_f vygenerujeme $q_f : return 1$.

Dostaneme nasledujúci kód programu *RationalC*, reprezentujúci TS M:

```
0: SHIFT(R) // posun desatinnej ciarky
1: SHIFT(R)
2: GOTO(q_0)    // skok na pociatocny stav
q_0:
q_0_0:   if x \% 2 == 0 goto q_0_0+3    // 0 = 10
q_0_0+1: GOTO(q_0_1)                  // ak Y nesedi, skusime iny symbol
q_0_0+3: x /= 2                       // posun na bit X
q_0_0+4: if x \% 2 == 1 goto q_0_0+7
q_0_0+5: x *= 2       // ak nesedi ani X, vratime hlavu na Y
q_0_0+6: GOTO(q_0_1) // nepodmienecny skok na overenie d'alšieho symbolu
q_0_0+7: x *= 2       // d'alej následuje vykonanie prechodu pre symbol S
q_0_0+8: return 0;    // 0 nie je definovana pre stav q_0

q_0_1:   if x \% 2 == 1 goto q_0_1+3    // 1 = 11
q_0_1+1: GOTO(q_0_D)                 // ak Y nesedi, skusime iny symbol
q_0_1+3: x /= 2                       // posun na bit X
q_0_1+4: if x \% 2 == 1 goto q_0_1+7
q_0_1+5: x *= 2       // ak nesedi ani X, vratime hlavu na Y
q_0_1+6: GOTO(q_0_D) // nepodmienecny skok na overenie d'alšieho symbolu
```

```
q_0_1+7: x *= 2       // d'alej následuje vykonanie prechodu pre symbol S
q_0_1:   return 0;    // 1 nie je definovana pre stav q_0

q_0_D:   if x \% 2 == 0 goto q_0_D+3    // Delta = 00
q_0_D+1: GOTO(q_0_#)                  // ak Y nesedi, skusime iny symbol
q_0_D+3: x /= 2                        // posun na bit X
q_0_D+4: if x \% 2 == 0 goto q_0_D+7
q_0_D+5: x *= 2       // ak nesedi ani X, vratime hlavu na Y
q_0_D+6: GOTO(q_0_#) // nepodmienecny skok na overenie d'alšieho symbolu
q_0_D+7: x *= 2       // d'alej následuje vykonanie prechodu pre symbol S
q_0_D+8:  SHIFT(R)    // podla d(q_0, delta) = (q_1, R)
q_0_D+9:  SHIFT(R)
q_0_D+10:  GOTO(q_1)
q_0_#:   return 0;

q_1:
q_1_0:   if x \% 2 == 0 goto q_1_0+3    // 0 = 10
q_1_0+1: GOTO(q_1_1)                   // ak Y nesedi, skusime iny symbol
q_1_0+3: x /= 2                        // posun na bit X
q_1_0+4: if x \% 2 == 1 goto q_1_0+7
q_1_0+5: x *= 2       // ak nesedi ani X, vratime hlavu na Y
q_1_0+6: GOTO(q_1_1) // nepodmienecny skok na overenie d'alšieho symbolu
q_1_0+7: x *= 2       // d'alej následuje vykonanie prechodu pre symbol S
q_1_0+8: return 0;    // 0 nie je definovana pre stav q_1

q_1_1:   if x \% 2 == 1 goto q_1_1+3    // 1 = 11
q_1_1+1: GOTO(q_1_D)                  // ak Y nesedi, skusime iny symbol
q_1_1+3: x /= 2                        // posun na bit X
q_1_1+4: if x \% 2 == 1 goto q_1_1+7
q_1_1+5: x *= 2       // ak nesedi ani X, vratime hlavu na Y
q_1_1+6: GOTO(q_1_D) // nepodmienecny skok na overenie d'alšieho symbolu
q_1_1+7: x *= 2       // d'alej následuje vykonanie prechodu pre symbol S
q_1_1+8:  SHIFT(R)    // podla d(q_1, 1) = (q_f, R)
q_1_1+9:  SHIFT(R)
q_1_1+10:  GOTO(q_f)  // uspesny koniec

q_1_D:   if x \% 2 == 0 goto q_1_D+3    // Delta = 00
q_1_D+1: GOTO(q_1_#)                  // ak Y nesedi, skusime iny symbol
q_1_D+3: x /= 2                        // posun na bit X
q_1_D+4: if x \% 2 == 0 goto q_1_D+7
q_1_D+5: x *= 2       // ak nesedi ani X, vratime hlavu na Y
q_1_D+6: GOTO(q_1_#) // nepodmienecny skok na overenie d'alšieho symbolu
q_1_D+7: x *= 2       // d'alej následuje vykonanie prechodu pre symbol S
q_1_D+8: return 0;    // 1 nie je definovana pre stav q_1
q_1_#:   return 0;

q_f: return 1
```

Napr. pre $w = 1$, ktoré patrí do $L(M)$ by binárne zakódovanie x_0 bolo 00110001. Po štarte by program najpr spravil 2x shift R, čo je delenie dvomi, teda stav registra x by nadobudol hodnotu 001100.01, teda desatinná čiarka by bola za znakom, ktorý reprezentuje Δ. Po následnej sekvencii overovania by program vykonal príkazy, začínajúce na riadku `q_0_D` a posunul sa na riadok `q_1` so stavom registra $x = 0011.0001$. Následne by program vykonal sekvenciu na riadku `q_1_1` odpovedajúcu prechodu $\delta(q_1, 1) = (q_f, R)$, a nakoniec so stavom $x = 00.110001$ by vykonal príkaz $return 1$.

B - RationalC na TS

Prirodzené číslo x_0 môžeme reprezentovať ako binárne kódované číslo. Na počiatku má teda páska tvar $\Delta w^R \Delta$, kde w je binárne zakódované prirodzené číslo x_0 a zároveň obsahuje TS sekvenciu prechod do prava, ktoré posunú hlavu na tú pozíciu, ktorá v pôvodnom racionálnom čísle predstavovala desatinnú čiarku. Desatinná čiarka simulovaného registra x_0 sa v TS nachádza vždy za pozíciou hlavy. Podobne ako pri prevode na program, aj tu je nutno užiť reverzáciu reťazca z dôvodu, že program má pri štarte desatinnú čiarku na konci reťazca, kdežto TS na začiatku pásky. V prípade, že by prirodzené číslo bolo rovné 0, volíme neprázdnu binárnu reprezentáciu pomocou 0. Konvertovaný TS teda po počiatočnom posunutí hlavy vždy začne na takej pozícii pásky, ktorá obsahuje len znaky 0 a 1.

Príklad: Prirodzené číslo 29 má binárne zakódovanie 11101. Jeho reverzáciou získavame 10111 a dokopy počiatočnu pásku $\Delta 11101 \Delta$. Zároveň pridáme do TS jeden extra počiatnočný stav, ktorý posunie hlavu do prava a prejde na na stav TS, reprezentujúci počiatok programu.

Prevod programu na stavy

Pred tým, než zadefinujeme prevod na TS, je nutné uvažovať na situáciami, ktoré musí TS korektne ošetriť. Pôvodný program totiž pracuje s neobmedzeným racionálnym číslom, teda s páskou, ktorá rastie do oboch strán. Naproti tomu, TS má pásku obmedzenú z ľava. Preto nutne bude TS pri *posunutí hlavy do ľava* kontrolovať, či po prechode nemá pod hlavou blank. Ak áno, potom aktivuje TS, ktorý skočí na ďalší blank do prava, vykoná shift do prava, skočí na blank v ľavo, zapíše 0 a pokračuje.

Podobne, pri skoku do ľava je nutné prepísať prípadný blank, ktorý sa po prechode môže vyskytnúť pod hlavou na 0, zavedením pomocného stavu a dvojicou prechodou (posun hlavy a zápis 0).

Samotný prevod príkazov vykonáme nasledovne. Každý riadok s príkazom programu *RationalC* je realizovaný ako jeden unikátny stav v TS. Implicitne, po vykonaní každého príkazu (okrem if) následuje zmena stavu na stav, reprezentovaný nasledujúcim riadkom v programe.

Uvažujme nasledujúce ekvivalentné vyjadrenie:

1. **return** 0 => abnormálne zastavenie TS (napr. pomocou prepadnutia hlavy)
2. **return** 1 => prechod do koncového stavu q_F
3. x = **odd**(x) => nepodmienenený zápis 1 pod čtecí hlavou
4. x = **even**(x) => nepodmienenený zápis 0 pod čtecí hlavou
5. x ***=** 2 => posun hlavy do ľava (reverzácia)
6. x /**=** 2 => posun hlavy do prava (reverzácia)
7. if **x %2** == b goto N => pre symbol B = {0,1} na páske preveď prechod do stavu na riadku N. Zároveň je pridaný prechod pre symbol $C = \{0,1\}/\{B\}$, ktorý vykoná prechod do stavu na riadku pod aktuálnym IF príkazom.

Pri prevode jednotlivých príkazov teda zavedieme nové stav pre každý príkaz a sémantiku stavu vyjadríme zavedením prechodov TS zo stavu príkazu do ďalšieho stavu (prípadne do stavu N v prípade príkazu *if.*). V prípade jednotlivých príkazov môžeme predpokladať, že pod hlavou budú vždy znaky 0 a 1. Pri realizácii príkazov $x* = 2$ a $x/ = 2$ je nutné zaviesť pomocné stavy, ktoré umožňia ošetriť prechod, pri ktorom by sa pod hlavou objavil symbol Δ. Pri prechode do prava v TS je nutné vygenerovať pomocný stav, ktorý pre symbol Δ zapíše 0 na pásku. Pre zvyšné symboly Γ vygenerujeme prechod do ďalšieho stavu, do ktorého pôvodne mal prejsť príkaz programu. Pre prechod do ľava vygenerujeme unikátny stav, ktorý pre symboly $0, 1$ prejde do dalšieho stavu. Pre symbol Δ sa aktivuje podstroj TS, ktorý sa dá popísať nasledujúco: $R_\Delta S_R L_\Delta 0$. Stroj najpr presunie hlavu na pozíciu prvej Δ na páske. Následne posunie celý obsah pásky o jednu pozíciu smerom do prava. Následne sa stroj presunie do ľava na prvú pozíciu blanku, na ktorú zapíše 0. Následne prechádza stroj do stavu, do ktorého by prešiel pôvodný príkaz. Stroj vlastne realizuje rozšírenie pásky o 1 nulový symbol z prava. Týmto sme ošetrili situáciu, kedy program rozširuje desatinné miesta racionálneho čísla a TS by prepadla hlava.

V predchádzajúcom texte sme predstavili spôsob, akým ekvivalentne zakódovať program v *RationalC* do TS. Prevod ošetruje situácie, ktoré by mohli viesť ku problémom (rozširovanie reprezentácie racionálneho čísla v registry x0).

Demonštrácia prevodu programu na TS

Uvažujme príklad č.1 zo zadania domácej úlohy. Pre každý príkaz s číslom $\{0, 1..., 6\}$ vytvoríme stav $q_i, i \in 0, ...6$ Pred jednotlive riadky programu následne pridáme prechody.

Pre daný program vytvoríme prechodovú funkciu nasledujúco:
$\delta(q_0, 1) = (q_6, 1)$ // if x % 2 == 1 goto 6
$\delta(q_0, 0) = (q_1, 0)$ // skok na ďalší riadok v programe

$\delta(q_1, 1) = (q_{1t}, R)$ // x /= 2
$\delta(q_1, 0) = (q_{1t}, R)$ // x /= 2
$\delta(q_2, 1) = (q_6, 1)$ // if x % 2 == 1 goto 6
$\delta(q_2, 0) = (q_3, 0)$ // skok na ďalší riadok v programe
$\delta(q_3, 1) = (q_{3t}, R)$ // x /= 2
$\delta(q_3, 0) = (q_{3t}, R)$ // x /= 2
$\delta(q_4, 1) = (q_6, 1)$ // if x % 2 == 1 goto 6
$\delta(q_4, 0) = (q_5, 0)$ // skok na ďalší riadok v programe
$\delta(q_5, 1) = (q_f, 1)$ // return 1
$\delta(q_5, 0) = (q_f, 0)$ // return 1
$\delta(q_6, 1) = (q_6, L)$ // return 0 (abnormálne zastavenie -
$\delta(q_6, 0) = (q_6, L)$ // return 0 (- prepadom hlavy)

Stavy q_1t a q_3t sú pomocné stavy ošetrenie posunu hlavy do prava. Pre tieto stavy je nutné vygenerovať prechody, ktorý zabezpečia rozšírenie pásky o znak 0 v prípade, že po delení je pod hlavou symbol Δ.
$\delta(q_1t, \Delta) = (q_2, 0)$
$\delta(q_1t, 0) = (q_2, 0)$
$\delta(q_1t, 1) = (q_2, 1)$

$\delta(q_3t, \Delta) = (q_4, 0)$
$\delta(q_3t, 0) = (q_4, 0)$
$\delta(q_3t, 1) = (q_4, 1)$

Vznikol nám TS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_f)$, kde $Q = \{q_0, q_1, q_1t, q_2, q_3, q_3t, q_4, q_5, q_6,_q f\}$, $\Sigma = 0, 1, \Gamma = \Sigma \cup \Delta$ realizuje ekvivalentne program z príkladu č.1.

Záver úlohy č.4

Zjavne sme ukázali, že TS je možné ekvivalentne zakódovať v programe *RacionalC* a naopak, teda nutne *RacionalC* je Turing úplný.