Teoretická informatika - Úkol č.2

Buchal Petr, xbucha02

1. Prosinec, 2018

1. Příklad

1.1 Zadání

Mějme gramatiku $G = (\{S\},\ \{[,]\},\ P,\ S)$ s pravidly

$$S \to \epsilon |\ [S]S.$$

Nechť L je Dyckův jazyk nad jednou dvojicí závorek $\{[,]\}$. Dokažte, že $L \subseteq L(G)$. Postupujte takto:

(a) Ukažte, že každé neprázdné slovo $w \in L$ lze napsat ve formě $[u]v$ kde $u, v \in L$. (Nápověda: Jak se dá napsat nejkratší neprázdný prefix slova w, který je sám také v L?)

(b) Dokažte, že $L \subseteq L(G)$, a to indukcí k počtu [ve slově. Tvrzení (a) použijte v indukčním kroku. (Báze: důkaz pro 0; Indukční krok: důkaz pro $i > 0$ za předpokladu, že tvrzení platí pro všechna $j < i$.)

1.2 Řešení

1.2.1 Bod a

- Slovo patří do Dyckova jazyka pokud splňuje tyto podmínky:

$$\#_{[}(w) - \#_{]}(w) = 0$$

$$\#_{[}(w_{libovolný\_prefix}) - \#_{]}(w_{libovolný\_prefix}) \geq 0$$

  Podle těchto podmínek se ve slovech Dyckova jazyka generují a kombinují dvě struktury, vnořená struktura $[^i]^i$ a sériová struktura $([])^i$, kde i představuje iteraci a závorky "(" a ")" obsah slova.
- Nejkratší slovo jazyka L je prázdné slovo ϵ, nejkratší neprázdné slovo v jazyce L je [], rovněž se jedná i o nejkratší neprázdný prefix slova w, který rovněž náleží do L. Tohle slovo tedy musí být možné generovat pomocí výrazu $[u]v$.
- Doplněním ϵ na pozice u a v ve výrazu $[u]v$ dostaneme slovo $[] \in L$. Na pozice u a v tedy můžeme dále dosazovat slovo [], které je neprázdné a které je možné generovat výrazem $[u]v$.
- Generování neprázdného slova $w \in L$ z výrazu $[u]v$ pomocí prázdného slova ϵ a nejkratšího neprázdného slova [] lze rozdělit na dva případy.
- Prvním případem je dosazení ϵ za u a [] za v, tím vzniká $[\epsilon][]$ a dochází ke generování vnořené struktury, kde $i = 2$.
- Druhým případem je dosazení ϵ za v a [] za u, tím vzniká $[[]]\epsilon$ a dochází ke generování sériové struktury, kde $i = 2$.
- Dosazením ϵ nedochází ke změně počtu závorek a obě podmínky jsou splněny, dosazením nejkratšího prefixu slova w, který zároveň náleží do L, jsou změněny počty závorek, obě podmínky jsou ovšem splněny. Zároveň jsou generovány obě struktury, jak vnořená, tak sériová.
- Do výrazu $[u]v$ na pozice u a v můžeme vložit libovolná slova, která reprezentují vložené a sériové struktury popsané stejným výrazem $[u]v$, poté tedy lze vyjádřit každé neprázdné slovo v jazyce L výrazem $[u]v$.

1.2.2 Bod b

- $L \subseteq L(G)$ tedy $\forall w \in L : w \in L(G)$
- V bodu a jsme dokázali, že výraz $[u]v$ kde $u, v \in L$ generuje každé neprázdné slovo náležící do L. Každé neprázdné slovo je poté kombinací dvou typů struktur, vnořené struktury $[^i]^i$ a sériové struktury $([])^i$, kde i představuje iteraci a závorky "(" a ")" představují obsah slova. Pokud bychom chtěli výrazem $[u]v$ generovat nejkratší neprázdný prefix slova w, který sám náleží do L dostaneme $[]$. Tohle slovo požijeme pro bázový krok.
- Báze pro 0: $\#_{[} = 0$ pro $[^0]^0 = \epsilon \in L$ (vnořená struktura) a $([])^0 = \epsilon \in L$ (sériová struktura), poté rovněž $\epsilon \in L(G)$, protože $S \rightarrow \epsilon$.
- Indukční krok pro $i > 0$ za předpokladu, že tvrzení platí pro všechna $j < i$:
  - Ukažme, že tvrzení platí pro takové w, kde $\#_{[} = i$, generování w kde $\#_{[}(w) = i$ popíšeme pro obě struktury, které se ve slově mohou vyskytnout - vnořenou a sériovou a dále pro obecné slovo.
  - Slovo s vnořenou strukturou má tvar $w = [^i]^i$, pro $[^i]^i$ platí $\#_{[}(w) = i \wedge [^i]^i \in L$, dále dle indukčního předpokladu $[^i]^i \in L(G)$, z G relací derivace dostaneme $S \Rightarrow [S]S \Rightarrow^* [^{i-1}[S]S]^{i-1} \Rightarrow [^{i-1}[\epsilon]\epsilon]^{i-1} \Rightarrow [^i]^i$, tedy $S \Rightarrow^* w$, tedy $w \in L(G)$.
  - Slovo se sériovou strukturou má tvar $w = ([])^i$, pro $([])^i$ platí $\#_{[}(w) = i \wedge ([])^i \in L$, dále dle indukčního předpokladu $([])^i \in L(G)$, z G relací derivace dostaneme $S \Rightarrow [S]S \Rightarrow^* ([])^{i-1}[S]S \Rightarrow ([])^{i-1}[\epsilon]\epsilon \Rightarrow ([])^i$, tedy $S \Rightarrow^* w$, tedy $w \in L(G)$.
  - Obecné slovo má tvar $w = [u^a]v^b$, kde $a + b = i - 1$, pro $[u^a]v^b$ platí $\#_{[}(w) = i \wedge [u^a]v^b \in L$, dále dle indukčního předpokladu $[u^a]v^b \in L(G)$, z G relací derivace dostaneme $S \Rightarrow [S]S \Rightarrow^* [S^a]S^b$, tedy $S \Rightarrow^* w$, tedy $w \in L(G)$.

2. Příklad

2.1 Zadání

Je jazyk $L_{primes} = \{a^n | n \text{ je prvočíslo}\}$ bezkontextový? Dokažte.

2.2 Řešení

- Předpokládejme, že $L_{primes} \in \mathcal{L}_2$.
- $\exists k > 0 : \forall z \in L : |z| \geq k \Rightarrow \exists u, v, w, x, y \in \Sigma^* : z = uvwxy \wedge vx \neq \epsilon \wedge |vwx| \leq k \wedge \forall i \geq 0 : uv^iwx^iy \in L$
- Uvažme libovolné $k > 0$ splňující uvedené tvrzení.
- Zvolme $z = a^b$, kde b je prvočíslo, které je rovno k nebo je nejbližším prvočíslem větším než k, $|z^b| = b \geq k$.
- Tedy $\exists u, v, w, x, y \in \Sigma^* : z = uvwxy \wedge vx \neq \epsilon \wedge |vwx| \leq k \wedge \forall i \geq 0 : uv^iwx^iy \in L$.
- Pokud v a x vyjádříme tak, že $v = a^c$ a $x = a^d$ kde $c \neq 0 \vee d \neq 0$, tak bude splněna podmínka $vx \neq \epsilon$, protože $c + d > 0$.
- Délku $|uwy|$ si můžeme vyjádřit jako $e = |uwy| - |v| - |x|$.
- Když zvolíme i rovno e, dostaneme slovo délky $|uv^ewx^ey| = e + c * e + d * e$
- Z tohoto tvaru vytkneme na $e(1 + c + d)$, pokud $e \neq \{0, 1\}$ délka slova není prvočíslo, protože je dělitelná e a zároveň $(1 + c + d)$, tedy slovo nepatří do jazyka.
- Pro $e = 0$ zvolíme $i = 2$, délka výsledného slova bude sudá a tudíž dělitelná 2 a slovo nebude náležet do jazyka.
- V případě $e = 1$, bude k délce $|vx|$ vždy přičtená 1, díky které se budou objevovat případy, kdy délka slova nebude dělitelná dvěma. Rovněž nelze využít první vztah, protože by se délka slova nezměnila a zůstala by prvočíslem. Zkusme provést úpravu, aby slovo bylo mocninou.
- Úpravu zkusíme provést pro mocninu dvou

$$b^2 = 1 + (c * i) + (d * i)$$

$$b^2 = 1 + i * (c + d), c + d = b - 1$$

$$b^2 = 1 + i * (b - 1)$$

$$\frac{b^2 - 1}{b - 1} = i$$

$$\frac{(b-1)(b+1)}{b-1} = i$$
$$i = b+1$$

- Pro $i = (b+1)$, kde b je délka původního slova, nebude nové slovo patřit do jazyka, neboť je jeho délka druhá mocnina původní délky, což není prvočíslo.
- Ve všech rozděleních lze nalézt takové i, že $|uv^iwx^iy| \notin L_{primes}$. Nastává tedy spor a $L \notin \mathcal{L}_2$.

3. Příklad

3.1 Zadání

Nechť $a_0, a_1 \in \mathbb{N}\backslash\{0\}$ jsou dané konstanty. Uvažujte jazyk

$$Affine = \{w \in \{0,\ 1\}^* \ | a_0 \cdot \#_0(w) + a_1 \cdot \#_1(w) - a_0a_1 = 0\}.$$

Dokažte pomocí redukce z problému členství (membership problem), že problém, zda jazyk daného Turingova stroje M obsahuje alespoň jeden řetězec z jazyka *Affine*, je nerozhodnutelný. Dále uveďte ideu důkazu, že problém je částečně rozhodnutelný.

3.2 Řešení

3.2.1 Problém je nerozhodnutelný

- Provedeme důkaz redukcí a to problému z problému členství (membership problem).
- Problémy budeme charakterizovat těmito jazyky:

$$MP = \{< M > \# < w > \ | \ M \text{ je TS, který přijme } w\}$$

$$AF = \{< M > \ | \ M \text{ je TS takový, že } \exists w \in \ L(M) : w \in L_{Affine} \ \}$$

- Sestavíme redukci σ: $\{0, 1, \#\}^* \rightarrow \{0, 1\}^*$ redukující MP na AF
- Redukci σ budeme reprezentovat TS M_σ, který ke vstupu $x \in \{0, 1, \#\}^*$ vygeneruje kód TS M_x pracujícího následovně:
  1. M_x svůj vstup $w \in \{0, 1\}$ smaže.
  2. M_x zapíše na vstupní pásku řetězec x, který má uložen ve svém stavovém řízení.
  3. M_x ověří, zda x má strukturu $x_1\#x_2$ pro x_1 platný kód TS a pro x_2 platný kód vstupu. Pokud x není platným kódem TS a jeho vstupu, M_x odmítne, tedy $L(M_x) = \emptyset$.
  4. M_x odsimuluje běh TS s kódem x_1 na vstupu s kódem x_2 s využitím UTS, který je jeho komponentou. Pokud TS s kódem x_1 na vstupu s x_2 skončí, pak M_x přijme, jinak nepřijme svedením do sink stavu.
- M_σ lze snadno realizovat jako úplný TS. Konečně M_σ vygeneruje kód TS M_x, který sestává převážně z předem známých a pevných komponent (smazání pásky, UTS, ověření platnosti kódu TS a jeho vstup) mezi kterým se předává řízení. Navíc dodáme veškeré chybějící přechody jejich svedením do sink stavu. Jedinou komponentou, která není pevná je zápis $x = a_1a_2...a_n$ na vstupní pásku. To lze snadno realizovat TS $Ra_1Ra_2...Ra_n$, jehož kód M_σ pro dané x snadno vypíše.
- Zbývá ukázat, že redukce σ implementovaná TS M_σ zachovává členství v jazyce vůči MP a AF:
  (a) $L(M_x) = \emptyset \Leftrightarrow x$ nemá strukturu $x_1\#x_2$ pro platný kód TS x_1 a jeho vstupu x_2 nebo pokud $x = x_1\#x_2$ a TS s kódem x_1 na vstupu x kódem x_2 nepřijme.
  (b) $L(M_x) = \{0, 1\}^* \Leftrightarrow x = x_1\#x_2$ a TS s kódem x_1 přijme vstup s kódem x_2.
- $x \in$ MP $\Leftrightarrow x$ má strukturu $x_1\#x_2$ a x_1 je TS, který přijme řetězec s kódem x_2
- $x \in$ MP $\Leftrightarrow L(M_x) = \{0, 1\}^* \Leftrightarrow < M_x > \in$ AF

3.2.2 Problém je částečně rozhodnutelný

- Můžeme sestrojit Turingův stroj M_{Affine}, který na své pásce simuluje výpočet vstupního Turingova stroje M pro jednotlivé možné vstupní řetězce, které náleží do L_{Affine}.
- TS M_{Affine} na své pásce postupně rozbíhá více simulací Turingova stroje M pro jednotlivé možné vstupní řetězce, které náleží do L_{Affine}, v lexikografickém uspořádání. U každé dílčí simulace si M_{Affine} pamatuje také stav řízení M při zpracování daného vstupu. TS M_{Affine} má jednotlivé rozběhnuté simulace má vhodným způsobem odděleny a může jim zvětšovat potřebný prostor.
- Simulace probíhá tak, že M_{Affine} vždy provede jeden krok na každé rozběhnuté simulaci a pokud nějaká z nich vede k přijetí, přijme. Jinak rozběhne další simulaci pro další vstupní řetězec a tento postup opakuje.
- TS M_{Affine} přijme, je-li $\exists w \in L(M) : w \in L_{Affine}$, jinak neskončí.

4. Příklad

4.1 Zadání

Uvažujte programovací jazyk **RationalC** s následující gramatikou:

$$\langle stmt \rangle ::= \mathrm{x} = \lceil \mathbf{odd}(\mathrm{x}) \rceil \mid \mathrm{x} = \lfloor \mathbf{even}(\mathrm{x}) \rfloor \mid \mathrm{x}* = 2 \mid \mathrm{x}/ = 2 \mid \mathbf{return}\ b \mid \mathbf{if}\ \mathrm{x}\ \%\ 2 == b\ \mathbf{goto}\ n$$

$$\langle stmt - list \rangle := \langle stmt \rangle ; \langle stmt - list \rangle \mid \langle stmt \rangle$$

$$\langle program \rangle ::= \langle stmt - list \rangle ;\ \mathbf{return}\ b;$$

kde $n \in \mathbb{N}, b \in \{0,\ 1\}$ a počáteční neterminál je $\langle program \rangle$ (uvažujeme že $0 \in \mathbb{N}$). Sémantika je následující:

- Program v **RationalC** je spouštěn na stroji s jedním registrem x, jenž může obsahovat racionální číslo s neomezenou přesností.
- Na začátku běhu programu je v registru x uloženo přirozené číslo $x_0 \in \mathbb{N}$ (vstup programu).
- Příkaz $\mathrm{x} = \lceil \mathbf{odd}(\mathrm{x}) \rceil$ změní celou část čísla v registru x na nejbližší větší nebo rovné liché číslo. Např. $\lceil \mathbf{odd}\ (42.1337)\ \rceil = 43.1337$ a $\lceil \mathbf{odd}\ (1.00777)\ \rceil = 1.00777$.
- Příkaz $\mathrm{x} = \lfloor \mathbf{even}\ (\mathrm{x}) \rfloor$ změní celou část čísla v registru x na nejbližší menší nebo rovné sudé číslo. Např. $\lfloor \mathbf{even}\ (42.1337)\ \rfloor = 42.1337$ a $\lfloor \mathbf{even}\ (1.00777)\ \rfloor = 0.00777$.
- Příkaz $\mathrm{x}* = 2$ vynásobí číslo v registru x dvěma.
- Příkaz $\mathrm{x}/ = 2$ vydělí číslo v registru x dvěma.
- Příkaz **return** b ukončí program s návratovou hodnotou b.
- Příkaz **if** x % 2 == b **goto** n provede podmínečný skok na n-tý příkaz (příkazy jsou čslovány od 1 a odděleny znakem středníku) v případě, že celá část čísla v registru x je (pro $b = 0$) či není (pro $b = 1$) dělitelná dvěma. Uvažujte, že syntakticky správný program neobsahuje skoky, kde n je větší než číslo posledního příkazu v programu.

Obrázek 1 obsahuje příklady programu v jazyce **RationalC**.
Dokažte, že programovací jazyk **RationalC** je Turingovsky uplný, tj., dokažte, že

Příklad 1: program vracející 1 právě tehdy, když je x_0 dělitelné 8

```
0 if x % 2 == 1 goto 6;
1 x /= 2;
2 if x % 2 == 1 goto 6;
3 x /= 2;
4 if x % 2 == 1 goto 6;
5 return 1;
6 return 0;
```

Příklad 2: program vracející 1 právě tehdy, když binární zápis čísla x_0 patří do jazyka popsaného regulárním výrazem $(0+1)^*0011(01+10)^*$.

```
0 if x % 2 == 0 goto 3;
1 if x % 2 == 1 goto 3;
2 x /= 2;                 // loop head
3 if x % 2 == 1 goto 7;
4 x /= 2;
5 if x % 2 == 1 goto 2;
6 if x % 2 == 0 goto 14;
7 x /= 2;
8 if x % 2 == 0 goto 2;
9 x /= 2;                 // found '11'
10 if x % 2 == 1 goto 14;
11 x /= 2;
12 if x % 2 == 1 goto 14;
13 return 1;
14 return 0;
```

Obrázek 1: Příklady programu v jazyce **RationalC**

(a) pro každý TS M nad abecedou $\{0,\ 1\}$ a řetězec $w \in \{0,\ 1\}^*$ lze sestrojit program P_M v jazyce **RationalC** a zvolit počáteční hodnotu x_0 tak, že P_M skončí s návratovou hodnotou 1 právě tehdy, když $w \in L(M)$;

(b) pro každý program P v jazyce **RationalC** a počáteční hodnotu x_0 lze sestrojit TS M_P a řetězec $w \in \{0,\ 1\}^*$tak, že $w \in L(M_P)$ právě tehdy, když P s počáteční hodnotou x_0 skončí s návratovou hodnotou 1.

Nápověda: binární zápis čísla 42.625_{10} je 101010.101_2.

4.2 Řešení

4.2.1 Bod a

Ukažme, že jednotlivé konstrukce Turingova stroje lze zapsat pomocí programovacího jazyka **RationalC**.

- Kromě vstupní abecedy skládající se z 0 a 1, potřebujeme mít možnost značit symbol $\triangle$. Zavedeme si tedy speciální kódování, 0 budeme značit jako 01, 1 jako 11 a $\triangle$ jako 00.
- x - Změnit hodnotu x lze pomocí příkazů **odd**(x) a **even**(x). Abecedou ve které je zakódovaná pásková abeceda je množina 0,1. Tyto hodnoty mohou být měněny tak, že 0 se příkazem **odd**(0) změní na 1 a dále 1 se příkazem **even**(1) změní na 0. Příkazy **odd**(x) a **even**(x) mění vždy první číslici před desetinou čárkou, kvůli tomu je třeba využít operaci L potažmo R pro posun na číslici, kterou chceme měnit. Pro kódované symboly se tento proces uskuteční dvakrát, to protože je každý symbol páskové abecedy je zakódován dvěma číslicemi.
- L - Posun doleva lze realizovat pomocí příkazu x/=2, který se dvakrát zopakuje kvůli zdvojenému kódování.
- R - Posun doprava lze realizovat pomocí příkazu x*=2, který se dvakrát zopakuje kvůli zdvojenému kódování.
- $\mathrm{L_a}$ - Hledání číslice a směrem doleva lze realizovat pomocí této jazykové konstrukce (příklad pro $\mathrm{L}_{(1)}$, předpokládám, že čtecí hlava je na pravé části kódovaného symbolu):

  n x/=2

  n+1 x/=2

  n+2 **if** x%2 == 1 **goto** n+4;

n+3 **if** x%2 == 0 **goto** n;

n+4 x/=2

n+5 **if** x%2 == 1 **goto** n+7;

n+6 **if** x%2 == 0 **goto** n+1;

n+7 ...

- R_a - Hledání číslice a směrem doleva lze realizovat pomocí této jazykové konstrukce (příklad pro $R_{(1)}$, předpokládám, že čtecí hlava je na levé části kódovaného symbolu):

  n x*=2

  n+1 x*=2

  n+2 **if** x%2 == 1 **goto** n+4;

  n+3 **if** x%2 == 0 **goto** n;

  n+4 x*=2

  n+5 **if** x%2 == 1 **goto** n+7;

  n+6 **if** x%2 == 0 **goto** n+1;

  n+7 ...

- $L_{\neg x}$ - Hledání číslice, která se nerovná a směrem doleva lze realizovat pomocí této jazykové konstrukce (příklad pro $L_{(\neg 1)}$, předpokládám, že čtecí hlava je na pravé části kódovaného symbolu):

  n x/=2

  n+1 x/=2

  n+2 **if** x%2 == 0 **goto** n+7;

  n+3 **if** x%2 == 1 **goto** n+4;

  n+4 x/=2

  n+5 **if** x%2 == 0 **goto** n+7;

  n+6 **if** x%2 == 1 **goto** n+1;

  n+7 ...

- $R_{\neg x}$ - Hledání číslice, která se nerovná a směrem doprava lze realizovat pomocí této jazykové konstrukce (příklad pro $R_{(\neg 1)}$, předpokládám, že čtecí hlava je na levé části kódovaného symbolu):

  n x*=2

  n+1 x*=2

  n+2 **if** x%2 == 0 **goto** n+7;

  n+3 **if** x%2 == 1 **goto** n+4;

  n+4 x*=2

  n+5 **if** x%2 == 0 **goto** n+7;

  n+6 **if** x%2 == 1 **goto** n+1;

  n+7 ...

- S_R, S_L - Shift doprava lze realizovat pomocí následující konstrukce, z důvodu zkrácení zápisu je v ní použito značení některých komponent TS, které jsou již výše sestaveny. Shift doleva se realizuje analogicky.

  n $R_\triangle$ //zastavení na pravé části zakódovaného znaku

  n+1 L

  n+2 L

  n+3 **if** x%2 == 0 **goto** n+8;

  n+4 **if** x%2 == 1 **goto** n+5;

  n+5 L

n+6 **if** x%2 == 0 **goto** n+8;

n+7 **if** x%2 == 1 **goto** n+15;

n+8 **if** x%2 == 0 **goto** n+25;

n+9 R

n+10 **even**(x)

n+11 $R_{\triangle}$ //zastavení na pravé části zakódovaného znaku

n+12 **odd**(x)

n+13 $L_{\triangle}$ //zastavení na levé části zakódovaného znaku

n+14 **if** x%2 == 0 **goto** n+2;

n+15 **even**(x)

n+16 R

n+17 **even**(x)

n+18 $R_{\triangle}$ //zastavení na pravé části zakódovaného znaku

n+19 **odd**(x)

n+20 L

n+21 **odd**(x)

n+22 $L_{\triangle}$ //zastavení na levé části zakódovaného znaku

n+24 **if** x%2 == 0 **goto** n+1;

n+25 ...

4.2.2 Bod b

Ukažme, že jednotlivé konstrukce programovacího jazyka **RationalC** lze zapsat Turingovým strojem. Uvažujme Turingův stroj se dvěma páskami, na první pásce bude zapsáno číslo z registru x v binární soustavě, na druhé pásce budou zapsány instrukce programu v prefixovém kódu. Hlava na první pásce bude simulovat pozici desetinné tečky, hlava na druhé pásce bude simulovat pozici v programu. Nastavení hlavy na první pásce proběhne na základě instrukcí zapsaných na druhé pásce, ještě před instrukcemi samotného programu.

- **odd**(x) - TS na druhé pásce přečte prefixový kód pro instrukci **odd**(x), na první pásce zapíše 1.
- **even**(x) - TS na druhé pásce přečte prefixový kód pro instrukci **even**(x), na první pásce zapíše 0.
- x*=2 - TS na první pásce přečte prefixový kód pro instrukci x*=2, na první pásce posune hlavu doprava.
- x/=2 - TS na první pásce přečte prefixový kód pro instrukci x/=2, na první pásce posune hlavu doleva.
- **return** b - TS má odlišné prefixové instrukce pro **return** 0 a **return** 1. Pokud TS druhou pásku přijme a skončí na prefixovém kódu instrukce symbolizující **return** 0, program vrací 0, pokud TS skončí na prefixovém kódu instrukce symbolizující **return** 1 program vrací 1.
- **if** x % 2 == b **goto** n - TS má odlišné instrukce pro **if** x % 2 == 0 a **if** x % 2 == 1, dále pro každé if návěští generuje unikátní instrukci, označme si ji A. Jedná se o ukazatel, který říká, kam má v případě splnění podmínky **goto** skočit. Když TS přečte A, provede instrukci R. Když TS přečte instrukci návěští if, přečte symbol z první pásky, vyhodnotí podmínku a přesune hlavu na požadované místo. Např. při **if** x % 2 == 1 **goto** n, se přečte hlava první pásky, pokud byla přečtena 1 vyhledá se instrukce A následovně - na druhé pásce se provede instrukce $L_{\triangle}$, která přemístí hlavu na levý konec pásky a z něj se provede instrukce R_A, která přemístí hlavu druhé pásky na instrukci A a odtud TS pokračuje dál, při nesplnění podmínky se provede instrukce R.