

# Kapitola 4

## Rozhodnuteľnosť

### 4.1 Univerzálny TS

Zamyslime sa nad existenciou Turingovho stroja, ktorý by bol schopný simulovať výpočet ľubovoľného iného Turingovho stroja. Nazvime ho *univerzálny Turingov stroj* a označme UTS. Čo rozumieme simuláciou ľubovoľného Turingovho stroja? Je to schopnosť UTS simulovať výpočet Turingovho stroja  $T$  na vstupe  $w$  ak pozná kód  $T$  a  $w$ . Inými slovami – UTS *dostane na vstupe* kód nejakého TS  $T$  a vstupné slovo  $w$  a výstupom UTS bude presne výstup  $T$  na vstupe  $w$ . Konštrukciou UTS ukážeme, že takýto stroj existuje. Je zrejmé, že jeho existencia úzko súvisí s možnosťou zakódovať program/prechodovú reláciu Turingovho stroja takým spôsobom, aby mu výpočtový model Turingov stroj rozumel.

Dôkaz existencie UTS je konštruktívny - pozostáva z konštrukcie požadovaného UTS. Skôr, ako napíšeme samotný popis UTS, treba sa rozhodnúť pre *rozumné* kódovanie Turingových strojov<sup>1</sup>.

Keďže

*kódovanie TS*

- pásková abeceda UTS je fixovaná a pásková abeceda TS  $T$  je konečná, ale ľubovoľne veľká - jej veľkosť pri konštrukcii UTS nepoznáme
- počet stavov UTS je fixovaný, ale počet stavov TS  $T$  je síce konečný, ale dopredu neznámej ľubovoľnej veľkosti

musíme na kódovanie používať rozumnú fixovanú abecedu, ktorou môžeme kódovať akúkoľvek konečnú ľubovoľne veľkú abecedu. Pritom treba dať pozor, aby zvolené kódovanie bolo jednoznačné, pretože dekódovanie potrebujeme jednoznačné.

Zvoľme

- $\{0, 1, B\}$  **na kódovanie páskových symbolov** Bez ujmy na všeobecnosti môžeme predpokladať, že abeceda páskových symbolov každého TS pozostáva, okrem špeciálneho symbolu  $B$ , ktorý označuje tzv. "blank" =prázdny symbol, len zo symbolov  $0, 1$ . Nech by nejaká konečná abeceda  $\Sigma$  obsahovala  $k$  rôznych symbolov:  $\Sigma = \{a_1, \dots, a_k\}$ . Potom každý zo symbolov  $a_i$  môžeme zakódovať/identifikovať indexom, teda reťazcom núl a jednotiek dĺžky  $m = \log k + 1$ . Prvky štvorprvkovej množiny  $a_1, a_2, a_3, a_4$ , resp.  $a, b, c, d$  môžeme kódovať reťazcami dĺžky 2:

$$a_1 = a \rightsquigarrow 00 \quad a_2 = b \rightsquigarrow 01 \quad a_3 = c \rightsquigarrow 10 \quad a_4 = d \rightsquigarrow 11$$

---

<sup>1</sup>UTS má dostať ako vstup kód TS, treba ho rozumným spôsobom zapísať

- **$1^s$  na kódovanie stavov** Nech množina stavov má  $k$  stavov, bez ujmy na všeobecnosti nech  $K = \{q_1, \dots, q_k\}$ . Potom reťazcom  $1^s$  kódujeme/označujeme stav  $q_s$ .

Každý TS  $T$  možno jednoznačne určiť/definovať tabuľkou rozmeru *počet stavov*  $\times$  *počet páskových symbolov*, kde obsah príslušného políčka (*stav, symbol*) určuje hodnotu  $\delta(\text{stav}, \text{symbol})$ . Aby sme tabuľku (ako vstup) uložili do jednorozmernej pásky, budeme ju zapisovať na pásku po riadkoch - blokoch. Pritom jeden riadok popisuje činnosť TS, ktorý v odpovedajúcom stave číta príslušný symbol z pásky. Predpokladáme, že poradie symbolov páskovej abecedy je fixované - napr. 0,1,B.

kód TS  $T$              $\#\#\#$  tabuľka  $\#\#\#$   
 celá tabuľka         $\#\#\#$  riadok  $\#\#\dots\#\#\#$  riadok  $\#\#\#$   
 riadok tabuľky      $\#\#1^j\#1^{S(0)}P_0Z_0\#1^{S(1)}P_1Z_1\#1^{S(B)}P_BZ_B\#\#\#$ , pričom  
                            $1^j$  znázorňuje, že nasleduje riadok pre spracovanie  $j$ -teho stavu,  
                           nasledujú podbloky pre spracovanie symbola 0, 1, B (v poradí)  
 jedno políčko  $j$ -teho riadku tabuľky (uložené vo viacerých políčkach vstupnej pásky)  
                            $\#1^{S(a)}, P_a, Z_a\#$ , kde  
                            $\delta(j, a) = (S(a), P_a, Z_a)$ ,  
                            $a \in \{0, 1, B\}$  je symbol čítaný pod hlavou  
                            $S(a)$  je stav, do ktorého sa má prejsť  
                            $P_a \in \{1, 0, -1\}$  je posun hlavy  
                            $Z_a \in \{0, 1\}$  je symbol, ktorý sa má zapísať

ak prechod na daný znak nie je definovaný, bude príslušné políčko tabuľky 0.

*simulácia zakódovaného TS*    UTS dostane na vstupe kód Turingovho stroja  $T$  a vstupné slovo  $w$  v tvare

$$\underbrace{\#\#\# \text{ tabuľka } \#\#\#}_{\text{kód TS } T} \quad \underbrace{w}_{\text{vstup do } T}$$

Obsah pásky tak pozostáva z dvoch častí - časť kódu a časť stroja  $T$ . Na začiatku pásky je kód, ktorý si potrebujeme počas celého výpočtu uchovať, druhú časť tvorí obsah pásky zakódovaného stroja  $T$  v priebehu výpočtu na vstupnom slove  $w$ . V oboch častiach si značkami budeme uchovávať informáciu, ktorá nám simuláciu uľahčí; predpokladáme preto, že páska je v priebehu výpočtu dvojstopá.

Stav stroja  $T$  si pamätáme tak, že máme v časti kódu značku stavu  $S$  umiestnenú v riadku, ktorý popisuje príslušný stav. Polohu hlavy na páske stroja  $T$  si pamätáme umiestnením značky hlavy  $H$  v spodnej stope strojom  $T$  snímaného políčka.

#	#	1	1	#			◦ ◦ ◦	#	#	#	1	0	1	B	B
				S							H				

simulovaný TS je v stave  $q_2$  a sníma symbol 0

*prípravná fáza*

- vytvor na vstupnej páske druhú stopu, pričom prepíš vstup do hornej stopy
- daj značku stavu pod  $\#$  nasledujúci za  $1^1$   
( $T$  začína svoj výpočet v počiatočnom stave 1)
- daj značku hlavy pod prvý symbol vstupného slova  $w$ . Zapamätaj si v riadkovej jednotke symbol  $a$ , ktorý sa nachádza pod hlavou  
(výpočet stroja  $T$  začína s hlavou umiestnenou na prvom políčku)

- nastav hlavu UTS na najľavejší symbol  $\#$  a hodnotu *pokračuj* na true

Keďže stroj UTS si v stave pamätá strojom  $T$  snímaný symbol a má označený stav, *simulačná fáza* v ktorom sa stroj  $T$  "momentálne" nachádza, môže z kódu  $T$  vyčítať a následne aplikovať príslušný krok  $T$ .

```

while pokračuj do
  nájdí v tabuľke značku stavu
  nájdí podblok  $\#1^{S(a)}P_aZ_a\#$  odpovedajúci spracovaniu symbola  $a$  pod hlavou 2
  if nie je definovaný (rovná sa  $\#0\#$ ) then
    pokračuj  $\leftarrow$  false
  else
    zapamätaj si v riadiacej jednotke  $S(a), P_a, Z_a$ 
    prepíš symbol pod značkou hlavy na  $Z_a$ , zapamätaj si  $Z_a$  v riadiacej
    jednotke ako nový symbol pod hlavou
    posuň značku hlavy o  $P_a$ 
    presuň hlavu stavu pod  $\#$  nasledujúci za  $1^{S(a)}$ 

```

□

**Poznámka:** Skonstruovaný univerzálny TS má fixovaný počet stavov a abecedu, ktorá je 4 (počet symbolov na hornej stope) x 3 (počet symbolov na spodnej stope) prvková. Opäť nie je problém prerobiť tento stroj tak, aby jeho abeceda bola len trojprvková  $\Sigma = \{0, 1, B\}$ . Uvedomme si, že samotný kód používa len symboly 0,1 - je nad dvojprvkovou abecedou  $\Sigma_{bin} = \{0, 1\}$ . Preto aj UTS má svoj kód, ktorý môže byť vstupom do UTS. Navyše, existuje usporiadanie na  $\Sigma_{bin}^*$  a má zmysel hovoriť o  $i$ -tom reťazci  $x_i$ , a teda aj  $i$ -tom TS, resp.  $i$ -tom kóde TS, či kóde  $i$ -teho TS.

$$\Sigma_{bin}^* = 0, 1, 00, 01, 10, 11, 000, \dots$$

$$x_7 = ? \quad 000101011 = x_?$$

Zrejme nie každý reťazec zo  $\Sigma^*$  je zmysluplným kódom TS tak, ako sme ho popísali. Zistiť to však nie je problém. Prípadne sa môžeme dohodnúť, že reťazec, ktorý nie je správnym kódom TS, budeme považovať za kód takého TS, ktorý rozpoznáva prázdny jazyk; potom je  $i$ -ty reťazec kódom  $i$ -teho TS.

**Úloha:** Napíšte program v ľubovoľnom programovacom jazyku, ktorý overí, či vstupný reťazec je syntakticky korektným kódom nejakého TS. Skúste ten istý problém riešiť na TS.

**Úloha:** Napíšte program (v ľubovoľnom programovacom jazyku), ktorý bude simulovať UTS. Zo vstupu načíta reťazec  $###kod###w$  a na výstup vypíše výstup zakódovaného TS.

**Úloha:** Zamyslite sa nad usporiadaním PASCALovských programov.

## 4.2 Rozhodnuteľné a nerozhodnuteľné problémy

Doteraz ste sa zrejme venovali takým problémom, ktoré sa dali riešiť. Dá sa ukázať, že existujú problémy, ktoré sú neriešiteľné. Dôkaz je jednoduchý:

$\Rightarrow$  Ak existuje riešenie, je napísané v nejakom jazyku; nezáleží na tom, či je to program, matematická formula alebo text. Vo všetkých prípadoch sa jedná o *reťazec*

<sup>2</sup>pamätáme si ho v riadiacej jednotke

nad konečnou abecedou. Keďže konečnú abecedu môžeme kódovať binárnymi reťazcami fixnej dĺžky, môžeme každému reťazcu *text* nad konečnou abecedou priradiť binárny reťazec  $\tau$ . No a teraz už nie je problém vnímať  $\tau$  ako binárny zápis čísla  $t$ .

$$\text{text} \rightsquigarrow \tau \rightsquigarrow t$$

Z toho dostávame, že *algoritmov/riešeni/dôkazov/.. je toľko, ako prirodzených čísel*.

$\Rightarrow$  Na druhej strane problémov je aspoň toľko, ako reálnych čísel. Všimnime si len jednu konkrétnu množinu problémov, a to rozpoznávanie formálnych jazykov. Formálnym jazykom môžeme rozumieť ľubovoľnú množinu reťazcov  $L$  nad konečnou abecedou  $\Sigma$ ;  $L \subseteq \Sigma^*$ . Nech  $a^n$  označuje reťazec pozostávajúci z  $n$  symbolov  $a$ . Potom

$$L = \{aab, aaabb, \dots, a^n b^{n-1} \mid n \in \mathbb{N}\}$$

je jazykom nad abecedou  $\Sigma = \{a, b\}$ . My sa obmedzíme na jazyky nad abecedou  $\{0, 1\}$ .

Ukážeme, že jazykov je toľko, ako reálnych čísel v intervale  $\langle 0, 1 \rangle$ .

Každému binárnemu reťazcu  $\tau$  vieme priradiť jeho poradové číslo  $i_\tau$  v postupnosti  $\Sigma_{bin}$ . Potom reálnemu číslu  $\alpha = 0.\alpha_1\alpha_2\dots$ ,  $0 < \alpha < 1$  priradíme najprv číslo  $\beta$ ,  $0 < \beta < 1$  tak, že  $\alpha_i$  nahradíme binárnym zápisom  $bin(\alpha_i)$  čísla  $\alpha_i$ .

$$0.23 \rightsquigarrow 0.\underbrace{0010}_{2}\underbrace{0011}_{3} \quad 0.451 \rightsquigarrow 0.\underbrace{0100}_{4}\underbrace{0101}_{5}\underbrace{0001}_{1}$$

$\beta = 0.\beta_1\beta_2\dots$  Potom

$$\begin{aligned} \beta &\rightsquigarrow L_\beta = \{\tau \in \{0, 1\}^*, \beta_{i_\tau} = 1\} \\ L \subseteq \{0, 1\}^* &\rightsquigarrow \beta = 0.\beta_1\beta_2\dots, \quad \beta_i = 1 \Leftrightarrow \tau_i \in L \end{aligned}$$

$\Rightarrow$  Keďže prirodzených čísel je menej ako reálnych, existujú problémy, pre ktoré neexistuje riešenie.

Na programovaní ste písali programy na riešenie rôznorodých problémov, na M ste skúmali uzavretosť, resp. neuzavretosť triedy objektov na niektoré operácie, ... Je tu však množstvo iných otázok, napr:

- je napísaný program korektný?
- je výpočet daného programu na konkrétnom vstupe konečný?
- ...

Toto sú otázky, na ktoré sa ťažko odpovedá. Chceli by sme nájsť algoritmy, ktoré nám odpoveď vypočítajú. Pre niektoré z uvedených problémov sa však hľadaný algoritmus nedarí nájsť. Existuje vôbec? Dostávame sa k problémom rozhodnuteľnosti.

*rozhodovací  
problém*

**Definícia 4.1** *Problém, ktorý zahrňuje nekonečný počet prípadov a v každom prípade je odpoveď buď áno alebo nie, je rozhodovací problém.*

*rozhodnuteľný  
problém*

**Definícia 4.2** *Rozhodovací problém je rozhodnuteľný, ak existuje algoritmus, ktorý pre daný vhodný kód ľubovoľného prípadu dá správnu odpoveď pre tento prípad. Ak algoritmus neexistuje, je problém nerozhodnuteľný.*

Pri kódovaní treba dávať pozor, aby nami predpokladané kódovanie existovalo; aby sme ho efektívne vedeli vyrábať a používať.

Ak chceme dokázať, že problém je rozhodnuteľný, napíšeme algoritmus, ktorý ho rieši. Ako ale dokázať, že problém nie je rozhodnuteľný, ale je nerozhodnuteľný? Musíme vyargumentovať, že neexistuje TS na jeho riešenie. Inými slovami, neexistuje TS, ktorý na každom vstupe zastane a správne odpovie; každý potenciálny kandidát sa na nejakom vstupe musí "pomýliť".

Ako prvý nerozhodnuteľný problém uvažujme jazyk  $L_{DIAG} \in \Sigma_{bin}^*$

$$L_{DIAG} = \{x_i \mid T_i \text{ neakceptuje } x_i\}$$

Pripomeňme si, že každý reťazec núl a jednotiek možno chápať ako vstupné slovo, ale tiež ako kód TS. A tiež to, že keď zafixujeme postupnosť reťazcov nad abecedou  $\{0, 1\}$ , tak má zmysel hovoriť o  $i$ -tom vstupe  $x_i$ , resp.  $i$ -tom TS  $T_i$ , ktorý zodpovedá  $i$ -temu slovu v tejto postupnosti (chápanom ako jeho kód).

**Veta 4.1** Jazyk  $L_{DIAG}$  je nerozhodnuteľný.

*nerozhodnuteľnosť*

$L_{DIAG}$

**Dôkaz:** Nerozhodnuteľnosť jazyka  $L_{DIAG}$  ukážeme sporom. Predpokladajme, že existuje TS  $T$ , ktorý jazyk  $L_{DIAG}$  rozpoznáva. Nech tento TS je  $i$ -ty v kódovaní TS, teda vlastne  $T_i$ . Zoberme slovo  $x_i$  a dajme ho na vstup TS  $T = T_i$ . Aká je situácia?

$$\left. \begin{array}{l} x_i \in L(T_i) \Rightarrow x_i \notin L_{DIAG} = L(T_i) \\ x_i \notin L(T_i) \Rightarrow x_i \in L_{DIAG} = L(T_i) \end{array} \right\} x_i \in L(T_i) \Leftrightarrow x_i \notin L(T_i)$$

V oboch prípadoch máme spor, preto bol predpoklad o existencii TS, ktorý rozpoznáva  $L_{DIAG}$  nesprávny. □

Použitému jazyku sa niekedy hovorí diagonalizačný a metóde dôkazu *diagonalizácia*. Je tomu tak (aj) preto, že keď spravíme tabuľku, ktorej jedným indexom je slovo, druhým indexom stroj a hodnota

$$\text{tab}(\text{Index-slova}, \text{Index-stroja}) = 1 \Leftrightarrow \text{stroj akceptuje slovo}$$

tak (diagonalizačný) jazyk  $L_{DIAG}$  vznikne tak, že hodnoty na diagonále zmeníme.

$$x_i \in L(T_i) \Leftrightarrow \text{tab}(i, i) = 0$$

**Úloha:** Metódu diagonalizácie môžeme použiť napríklad aj pri argumentácii o existencii funkcie, ktorá sa nedá vypočítať žiadnym PSCALovským programom. Skúste.

V okamihu, keď už máme nejaký nerozhodnuteľný problém, môžeme ho využiť pri dôkaze nerozhodnuteľnosti iného problému. Ide vlastne o dôkaz sporom, ktorý je zachytený v nasledujúcej schéme.

*metóda redukcie*

Nech

**N** je známy nerozhodnuteľný problém

**K** problém - kandidát na to, aby bol nerozhodnuteľný

Postupujeme sporom:

1. Predpokladajme, že problém  $K$  je rozhodnuteľný a  $\mathcal{A}$  je algoritmus, ktorý ho rozhoduje.
2. Skonstruujeme algoritmus  $\mathcal{B}$ , ktorý za predpokladu existencie  $\mathcal{A}$  rieši  $N$ .
3. Keďže  $N$  je nerozhodnuteľný, dostaneme spor s predpokladom existencie  $\mathcal{A}$ . Preto je  $K$  nerozhodnuteľný.

Uvedená schéma je vlastne **metódou redukcie**, keď sa neriešiteľnosť jedného problému získa redukciami na iný problém. Formálne:

*rekurzívna redukovateľnosť* **Definícia 4.3** *Nech  $L_1 \subseteq \Sigma_1^*$ ,  $L_2 \subseteq \Sigma_2^*$  sú jazyky. Hovoríme, že jazyk  $L_1$  je rekurzívne redukovateľný na jazyk  $L_2$ ,  $\mathbf{L}_1 \leq_{\mathbf{R}} \mathbf{L}_2$ , ak*

$$L_2 \in \mathcal{L}_R \implies L_1 \in \mathcal{L}_R$$

Neformálne  $L_1 \leq_R L_2$  vyjadruje fakt, že jazyk  $L_2$  je z hľadiska algoritmickej riešiteľnosti aspoň tak ťažký, ako jazyk  $L_1$ .

Pre aplikovanie spomínanej schémy potrebujeme nájsť vhodný nerozhodnuteľný problém  $N$ , ktorého riešenie by nám pomohlo vyriešiť náš problém  $K$ . Ako však využiť algoritmus jedného problému pre riešenie iného problému? Jedným zo spôsobov je redukcia/transformácia jedného problému na druhý. Formálnejšie:

*many-one redukovateľnosť* **Definícia 4.4** *Nech  $L_1 \subseteq \Sigma_1^*$ ,  $L_2 \subseteq \Sigma_2^*$  sú jazyky. Hovoríme, že jazyk  $L_1$  je (many-one) redukovateľný na jazyk  $L_2$ ,  $\mathbf{L}_1 \leq_{\mathbf{m}} \mathbf{L}_2$ , ak existuje TS  $M$ , ktorý počíta zobrazenie*

$$f_M : \Sigma_1^* \rightarrow \Sigma_2^*$$

*také, že*

$$\forall x \in \Sigma_1^* \quad x \in L_1 \Leftrightarrow f_M(x) \in L_2$$

*Funkciu  $f_m$  zvykneme hovoriť redukcia  $L_1$  na  $L_2$ .*

O vzťahu spomenutých redukcí hovorí nasledujúca lema, dôkaz ktorej necháme ako cvičenie.

**Lema 4.2** *Nech  $L_1 \subseteq \Sigma_1^*$ ,  $L_2 \subseteq \Sigma_2^*$  sú jazyky.*

$$\text{ak } L_1 \leq_R L_2 \quad \text{tak } L_1 \leq_m L_2$$

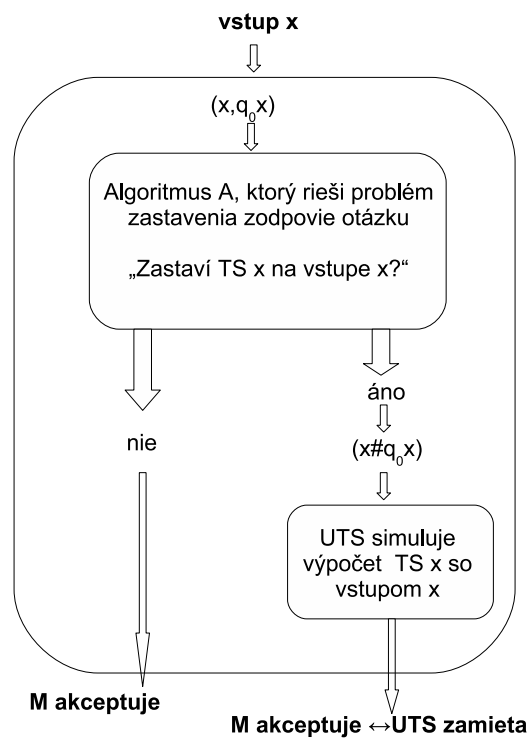
A teraz sa vráťme k dôkazom nerozhodnuteľnosti.

### 4.2.1 Zastavenie TS

*problém zastavenia* **Definícia 4.5** *Problém zastavenia: pre daný kód TS  $T$  a konfiguráciu  $C$  určiť, či  $T$  zastane pri výpočte štartujúcim z konfigurácie  $C$ .*

Ukážeme, že tento problém je nerozhodnuteľný. Napriek tomu, že je tento problém formulovaný ako problém zastavenia TS, zrejme nie je ťažké si predstaviť, že k danému programu v nejakom programovacom jazyku by sme napísali TS, ktorý by simuloval jeho činnosť. Potom sa problém zastavenia TS stáva problémom toho, či daný program vôbec niekedy skončí. Neznamená to nutne, že z toho priamo vyplýva nerozhodnuteľnosť problému zastavenia programu, ale určite si uvedomíme, že je to ťažký problém. A to je problém zastavenia zrejme ľahší, ako požadovať odpoveď na otázku, či je daný program korektný...

*nerozhodnuteľnosť zastavenia* **Veta 4.3** *Problém zastavenia TS je nerozhodnuteľný.*



Obrázok 4.1: Rozpoznávanie  $L_{DIAG}$  pomocou algoritmu, ktorý rieši problém zastavenia

**Dôkaz:** Pri dôkaze využijeme nerozhodnuteľnosť jazyka  $L_{DIAG}$ ,

$$L_{DIAG} = \{x_i \mid T_i \text{ neakceptuje } x_i\}$$

ktorý použijeme podľa vysvetlenej schémy:

1. Predpokladajme, že je problém zastavenia TS rozhodnuteľný,  $\mathcal{A}$  je TS, ktorý ho rozhoduje.
2. Uvažujme nasledovný algoritmus/TS  $M$  na rozpoznávanie  $L_{DIAG}$ :
  - $M$  vytvorí počiatočnú konfiguráciu  $C = q_0x$  a chápe  $x$  ako kód TS. Vytvorí vstup pre problém zastavenia

$$\left( \underbrace{x}_{\text{kód TS}}, \underbrace{q_0x}_{\text{počiatočná konfigurácia}} \right)$$

- $M$  odovzdá riadenie TS  $\mathcal{A}$ , ktorý rozhodne, či TS  $x$  so vstupom  $x$  zastane alebo nie
- ak TS  $x$  zastane, tak  $M$  odovzdá riadenie UTS, ktorý odsimuluje výpočet TS  $x$  na vstupe  $x$  a odpovie naopak
- ak TS  $x$  na vstupe  $x$  nezastane, tak  $M$  akceptuje.

Ak vieme, že jazyk  $L_{DIAG}$  sa nedá rozpoznávať, bol predpoklad o existencii TS  $\mathcal{A}$  rozhodujúceho problém zastavenia nesprávny.  $\square$

### 4.2.2 Postov korešpondenčný problém

**Definícia 4.6** *Nech  $\Sigma$  je konečná abeceda,  $A$  a  $B$  sú dva zoznamy slov zo  $\Sigma^*$ , PKP pričom oba majú rovnaký počet slov.*

$$\mathbf{A} = w_1, w_2, \dots, w_n$$

$$\mathbf{B} = x_1, x_2, \dots, x_n$$

*Hovoríme, že Postov korešpondenčný problém (PKP, presnejšie inštancia/prípád PKP) má riešenie, ak existuje postupnosť indexov  $i_1, \dots, i_m$  taká, že*

$$w_{i_1} w_{i_2} \dots w_{i_m} = x_{i_1} x_{i_2} \dots x_{i_m}$$

*V takom prípade je postupnosť  $i_1, \dots, i_m$  riešením tohto prípadu PKP.*

*Ak vyžadujeme, aby  $i_1 = 1$ , ide o modifikovaný Postov korešpondenčný problém.*

*Nech  $(A, B)$  je vstup do PKP. Fakt, že tento prípad má riešenie zvykneme označovať  $(A, B) \in \text{PKP}$ . Analogicky pre MPKP.*

Nasledujú dva príklady konkrétnych vstupov do PKP. Prvý je príkladom PKP, ktorý riešenie má.

#### Príklad 4.1

$i$	$w_i$	$x_i$
1	1	111
2	10111	10
3	10	0

#### Riešenie

2,      1,    1,    3  
 10111   1    1    10  
 10      111   111   0

A teraz príklad prípadu PKP, ktorý riešenie nemá.

#### Príklad 4.2

$i$	$w_i$	$x_i$
1	10	101
2	011	11
3	101	011

#### Riešenie

*Aby  $x_{i_1}, w_{i_1}$  mohli byť začiatkom riešenia, musí byť jeden druhému prefixom. Preto do úvahy pripadá len  $i_1 = 1$ . Takto dostaneme*

*A: 10*

*B: 101*

*Ak chceme správne pokračovať, musí byť  $i_2 = 3$ . Avšak potom*

*A: 10101*

*B: 101011*

*Opäť je zoznam  $A$  kratší a jediné možné pokračovanie túto situáciu zachováva. Preto tento prípad PKP riešenie nemá.*

Uvedené dve verzie Postovho korešpondenčného problému sa z hľadiska definície líšia len minimálne. Čo vieme povedať o riešení týchto problémov? Líšia sa z hľadiska zložitosti? Uvidíme, že nie. Najprv ukážeme, že PKP je rozhodnuteľný práve vtedy ak je rozhodnuteľný MPKP. Následne potom redukciami na problém zastavenia ukážeme, že MPKP (a teda aj PKP!) je nerozhodnuteľný.

**Fakt 4.4** *Ak by bol rozhodnuteľný MPKP, tak by bol rozhodnuteľný aj PKP.* *MPKP  $\rightarrow$  PKP*

**Dôkaz:** Majme inštanciu PKP danú zoznamami  $A, B$

$$A = w_1, w_2, \dots, w_n$$

$$B = x_1, x_2, \dots, x_n$$

Predpokladajme, že existuje algoritmus  $\mathcal{A}$  na riešenie MPKP. Ľahko vidno, že PKP má riešenie práve vtedy, ak ho má niektoré z  $n$  prípadov *modifikovaného* MPKP, ktoré vzniknú shiftami zoznamov  $A, B$ . Nech sú to  $PKP_1, PKP_2, \dots, PKP_n$ , kde  $PKP_i = (A_i, B_i)$ ,

$$A_i = w_i, \dots, w_n, w_1, \dots, w_{i-1}$$

$$B_i = x_i, \dots, x_n, x_1, \dots, x_{i-1}$$

Rozhodnuteľnosť PKP sme takto zredukovali na nanajviš  $n$  volaní algoritmu  $\mathcal{A}$ .

□

**Veta 4.5** *Ak by bol rozhodnuteľný PKP, tak by bol rozhodnuteľný aj MPKP.* *PKP  $\rightarrow$  MPKP*

**Dôkaz:** Aby sme pri riešení MPKP mohli využiť algoritmus, ktorý rieši PKP, budeme postupovať takto:

Nech  $(A, B)$  je prípad MPKP

$$A = w_1, w_2, \dots, w_n$$

$$B = x_1, x_2, \dots, x_n$$

Zostrojíme (k nemu) taký prípad  $(A', B')$  PKP, pre ktorý bude platiť

$$(A, B) \in MPKP \Leftrightarrow (A', B') \in PKP$$

Konštrukcia PKP  $(A', B')$

Nech  $\Sigma$  je najmenšia abeceda obsahujúca všetky symboly zo zoznamov  $A, B$  a nech  $\$, \# \notin \Sigma$ . Uvažujme nasledujúce homomorfizmy:

$$h_L(a) = \$a$$

$$h_R(a) = a\$$$

Potom hľadané zoznamy sú:

	$w'_i$	$x'_i$
1	$\$h_R(w_1)$	$h_L(x_1)$
$i+1$	$h_R(w_i)$	$h_L(x_i)$
$n+2$	$\S$	$\$\S$

PKP má riešenie  $\Leftrightarrow$  MPKP má riešenie

Nech  $1, i_1, i_2, \dots, i_m$  je riešenie MPKP  $(A, B)$ . Potom  $1, i_1+1, i_2+1, \dots, i_m+1, n+2$  *MPKP  $\rightarrow$  PKP* je riešenie PKP  $(A', B')$ .

Nech  $i_1, i_2, \dots, i_r$  je riešenie PKP  $(A', B')$ . Potom  $i_1 = 1$  a  $i_r = n+2$ . Nech  $j$  *PKP  $\rightarrow$  MPKP* je najmenší taký index, že  $i_j = n+2$ . Potom zrejme  $i_1, i_2, \dots, i_j$  je tiež riešenie PKP  $(A', B')$ . Riešenie MPKP  $(A, B)$  potom je  $1, i_2-1, i_3-1, \dots, i_{j-1}-1$ . □

**Veta 4.6** *Modifikovaný Postov korešpondenčný problém je nerozhodnuteľný.*

*nerozhodnuteľnosť  
MPKP*

**Dôkaz:** Pri dôkaze využijeme nerozhodnuteľnosť problému zastavenia — ak by bol rozhodnuteľný MPKP, tak by bol rozhodnuteľný aj problém zastavenia TS.

K danému TS  $T$  a vstupnému slovu  $w$  — teda vstupu do problému zastavenia — skonštruujeme prípad MPKP. Pritom skonštruovaný prípad MPKP bude mať tú vlastnosť, že má riešenie práve vtedy, ak sa TS  $T$  vo výpočte z počiatočnej konfigurácie so vstupom  $w$  zastaví.

Zoznamy MPKP preto budeme konštruovať tak, aby sme mali možnosť "simulovať" výpočet TS  $T$  na slove  $w$ .

- Nech TS  $T = (K, \Sigma, \Gamma, \delta, q_0, F)$ . Predpokladáme, že  $\forall q \in F \forall a \in \Sigma$  je  $\delta(q, a)$  nedefinované; po dosiahnutí akceptujúceho stavu sa TS zastaví.
- Konfiguráciu  $(q, \alpha, i)$  budeme reprezentovať reťazcom  $\alpha_1 q \alpha_2$ , kde  $|\alpha_1| = i - 1$  ô stav vkladáme pred symbol snímaný hlavou TS.
- Ak  $q_0 w, \alpha_1 q_1 \beta_1, \alpha_2 q_2 \beta_2, \dots, \alpha_k q_k \beta_k$  bude možným výpočtom,  $q_k \in F$ , tak začiatok každého riešenia prípadu MPKP bude

$$\#q_0 w \# \alpha_1 q_1 \beta_1 \# \alpha_2 q_2 \beta_2 \# \dots \# \alpha_k q_k \beta_k \#; \# \notin K \cup \Gamma$$

Na zozname B simulujeme výpočet TS  $T$ , zoznamom A sme o jeden krok pozadu. Takto na základe konfigurácie  $C_i$  v zozname A máme možnosť budovať konfiguráciu  $C_{i+1}$  v zozname B.

*čiasť* **Označenie:** Hovoríme, že  $(x, y)$  je *čiasť* riešenie MPKP, ak  $x$  je prefixom  $y$  a  $x, y$  sú potenciálnym začiatkom riešenia prípadu MPKP.

*vyšok* Ak  $xz = y$ , potom hovoríme, že  $z$  je *vyšok* čiastočného riešenia  $x, y$ .

	Zoznam A	Zoznam B	
<i>vytváranie pásiky skupina 1</i>	# X #	# $q_0 w$ # X #	$X \in \Gamma - B$
<i>simulácia T vytváranie úseku v okolí hlavy skupina 2</i>	qX ZqX q# Zq#	Yp pZY Yp# pZY#	$\delta(q, X) = (p, Y, R)$ $\delta(q, X) = (p, Y, L)$ $\delta(q, B) = (p, B, R)$ $\delta(q, B) = (p, Y, L)$
<i>A postupne "dobíha" B skupina 3</i>	XqY Xq# #qY	q q# #q	$q \in F, X, Y \in \Gamma - \{B\}$
<i>záver-skupina 4</i>	q##	#	$q \in F$

Matematicou indukciou sa ukáže:

Nech  $q_0w, \alpha_1q_1\beta_1, \alpha_2q_2\beta_2, \dots, \alpha_kq_k\beta_k$  je platná postupnosť konfigurácií taká, že neobsahuje koncový stav. Potom existuje čiastočné riešenie

$$(x,y) = (\#q_0w\#\alpha_1q_1\beta_1\#\alpha_2q_2\beta_2\#\dots\#\alpha_{k-1}q_{k-1}\beta_{k-1}\#, \\ \#q_0w\#\alpha_1q_1\beta_1\#\alpha_2q_2\beta_2\#\dots\#\alpha_kq_k\beta_k\#)$$

Od okamihu, keď sa v zozname B objaví koncový stav, vhodným výberom indexov skupiny 1 a 3 a na záver skupiny 4 dotiahneme riešenie do konca.  $\square$