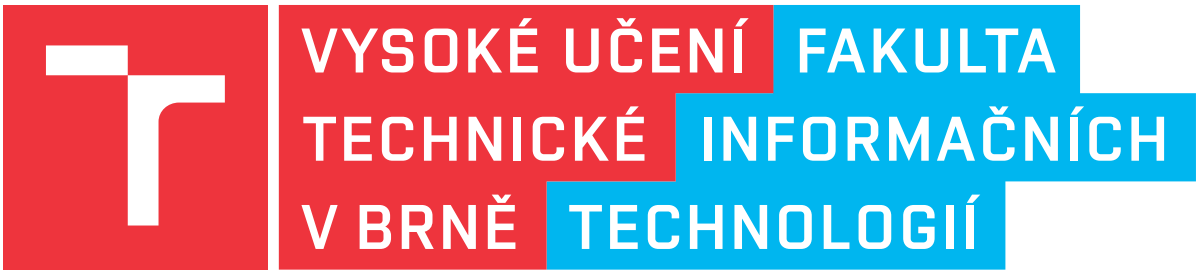

Teoretická informatika (TIN)

2. domácí úloha

v Brně
18. prosince 2019

Václav Martinka
xmarti76

1. úkol **Uvažujte jazyk L. Dokažte, že jazyk je bezkontextový**

$$L = \{w \in \{a,b\}^* | \#_a(w) = \#_b(w)\}$$

a) Navrhněte gramatiku G tak, že $L = L(G)$
$G = (\{S\}, \{a,b\}, \{S \to SS|aSb|S \to bSa|S \to \epsilon\}, S)$

b) Pomocí indukce dokažte, že $L = L(G)$
1) Musíme dokázat, že platí $L \subseteq L(G)$

- Pro slovo délky 0:
  $S \Rightarrow \epsilon \ \wedge \ |\epsilon| = 0 \ \wedge \ \epsilon \in L$ **PLATÍ**
- Předpokládejme $\forall w \in L, |w| \leq i : S \Rightarrow^* w$
- Pro slova délky $i+2$:
  $w' = \{\{a,b\}^* | \#_a(w') = \#_b(w') = \frac{i}{2} + 1\} \ \Rightarrow \ w' \in L \ \wedge \ |w'| = i+2$
  $w' = axb|bxa \ kde \ x = \{\{a,b\}^* | \#_a(x) = \#_b(x) = \frac{i}{2}\} \ \wedge \ S \Rightarrow^* x \ \wedge \ |x| = i$
- Pak tedy:
  $S \Rightarrow aSb|bSa \Rightarrow^* axb|bxa = \{\{a,b\}^* | \#_a(x) = \#_b(x) = \frac{i+2}{2}\}$ kde
  $x = \{\{a,b\}^* | \#_a(x) = \#_b(x) = \frac{i}{2}\}$ **PLATÍ**

2) Dále musíme dokázat i $L(G) \subseteq L$

- Pro slovo délky 0:
  $S \Rightarrow \epsilon \ \wedge \ |\epsilon| = 0 \ \wedge \ \epsilon \in L$ **PLATÍ**
- Předpokládejme $\forall w \in L, |w| \leq i : S \Rightarrow^* w$
- Pro slova délky $i+2$:
  $S \Rightarrow aSb|bSa \Rightarrow aw'b|bw'a$
  nebo
  $S \Rightarrow SS \Rightarrow aSbS|bSaS|SaSb|SbSa$ dále viz předchozí řádek
- Pokud $S \Rightarrow^* w'$ pak dle indukčního předpokladu platí:
  $w' \in L$, tedy $w' = \{\{a,b\}^* | \#_a(x) = \#_b(x) = \frac{i}{2}\}$
- $aw'b|bw'a = \{\{a,b\}^* | \#_a(x) = \#_b(x) = \frac{i+2}{2}\} \in L$ **PLATÍ**

2. úkol Dokažte, zda je problém prázdnosti jazyka L^P rozhodnutelný

Idea

Z definice jazyka $L^P(M)$ vyplývá, že TS M může při přijetí řetězce použít pouze pravidla ve tvaru $(q_1, a) \to (q_2, R)$ nebo $(q_1, a) \to (q_2, a)$. Pokud existuje posloupnost takových pravidel, která vede do koncového stavu, musí nutně platit, že $L^P(M) \neq \emptyset$

Vstup: Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$

Výstup: *True* pokud je jazyk $L^P(M)$ neprázdný, jinak *false*.

Metoda:

Krok 1) $X^0 = \{q_0\},\ Q' = Q \setminus \{q_0\}$
Pozn.: *X^i bude obsahovat dostupné stavy v iteraci i. Q' obsahuje zatím nenavštívené stavy, aby se zabránilo cyklení.*

Krok 2) $\forall q_1 \in X^i\ \forall q_2 \in Q'\ \forall a \in \Gamma : (q_2, x \in \{a, R\}) \in \delta(q_1, a) \Leftrightarrow \{q_2\} \in X^{i+1} \wedge Q' = Q' \setminus \{q_2\}$
Pozn.: *Pro všechny stavy q_1 z množiny dostupných stavů X^i najdeme všechny dosažitelné stavy přijetím libovolného symbolu a přechodu doprava nebo zápisem stejného symbolu. Pokud jsou tyto nové stavy obsaženy v množině zatím nenavštívených stavů Q', odebereme je a zároveň vložíme do množiny X^{i+1}.*

Krok 3) Pokud $X^{i+1} = \emptyset$, vrať *false*.
Pozn.: *Pokud je množina X^{i+1} prázdná, znamená to, že již není kam iterovat, tudíž koncových stav q_F není dostupný.*

Krok 4) $\forall q \in X^{i+1} : q \neq q_F \Rightarrow$ *goto* Krok 2
Pozn.: *Pokud žádný ze stavů q v množině X^{i+1} není koncový, vrátíme se na krok 2.*

Krok 5) Vrať *true*.
Pozn.: *Množina X^{i+1} obsahuje koncový stav q_F, ten je tudíž dostupný pouze pomocí pravidel s posuvem doprava a L^P tudíž musí být neprázdný.*

Podařilo se nám sestavit algoritmus, který tento problém rozhoduje. Algoritmus pracuje v iteracích nad konečnou množinou stavů, tudíž musí v konečném čase vrátit výsledek.

3. úkol Dokažte pomocí redukce, že L_{42} je nerozhodnutelný

Důkaz provedeme redukcí z *co-HP*.

Jazyk charakterizující *co-HP* problém je

$$co - HP = \{\langle M\rangle\#\langle w\rangle | M \textit{ je TS takový, že na } w \textit{ nezastaví}\}$$

Navrhneme redukci $\sigma : \{0,1,\#\}^* \to \{0,1\}^*$ redukující *co-HP* na L_{42}

Redukce σ přiřadí každému řetězci $x \in \{0,1,\#\}^*$ kód TS M_x, který pracuje následovně:

1. M_x smaže svůj vstup w
2. M_x zapíše na vstupní pásku řetězec x, který má uložený v konečném stavovém řízení
3. M_s ověří, zda x má strukturu $x_1\#x_2$, kde x_1 je kód TS M_{x_1} a x_2 je kód řetězce w_{x_2}
4. Pokud x nemá patřičnou strukturu, **odmítne**
5. M_x odsimuluje TS s kódem M_{x_1} na vstupu s kódem w_{x_2}, pokud M_{x_1} zastaví **přijme**, jinak cyklí

σ lze implementovat úplným TS M_σ, který pro vstup x vyprodukuje kód TS M_x, který se sestává ze 4 komponent:

1. komponenta, která maže vstupní pásku
2. M_σ vypíše kód TS, který zapíše na vstup x
3. M_σ vypíše kód TS, který na vstupu ověří, zda se jedná o platnou instanci *co-HP* a pokud ne, **přijme**
4. M_σ vypíše kód TS, který spustí univerzální TS na TS_{x_1} a vstupu x_2

M_σ zajistí sekvenční předávání řízení

Zkoumejme jazyk TS M_x:

a) $L(M_x) = \emptyset \Leftrightarrow x$ má správnou strukturu a TS s kódem x_1 na vstupu s kódem x_2 zastaví

b) $L(M_x) = \Sigma^* \Leftrightarrow (x$ nemá strukturu $x_1\#x_2$ pro kód TS x_1 a kód vstupu $x_2)$ nebo $(x$ má správnou strukturu, ale TS s kódem x_1 na vstupu s kódem x_2 nezastaví)

Nyní již snadno ukážeme, že σ zachová členství v jazyce:

$$\forall x \in \{0,1,\#\}^* : \sigma(x) = \langle M_x\rangle \in L_4 2 \Leftrightarrow L(M_x) = \Sigma^* \Leftrightarrow \textit{bod } b) \Leftrightarrow x \in \textit{co-HP}$$

4. úkol Dokažte, že programovací jazyk `Karel@TIN` je Turingovsky úplný

a) Pro každý TS M lze sestrojit program P_M

Nejdříve si vytvoříme dva jednoduché podprogramy `jdiKeSroubku` a `sbirejSroubky`:

- `jdiKeSroubku` bude podprogram, kdy robot půjde po přímce, dokud nenarazí na šroubek nebo konkrétní počet šroubků.
- `sbirejSroubky` postupně sebere všechny šroubky z políčka, pokud tam nějaké jsou. Tento podprogram **ne**vytvoříme pomocí skoků, tudíž sice bude robot omezen na konečný počet šroubků (daný délkou podprogramu), ale bude schopný si ve stavovém řízení uložit jejich počet, na konci počítání je tam opět všechny vrátit a provést skok do odpovídající části programu.

Důkaz provedeme návrhem algoritmu pro převod TS na program. Program P_M bude simulovat TS M, musí být tedy schopen uchovat konfiguraci TS a simulovat jeho přechody.

Konfigurace TS se skládá z aktuálního stavu, obsahu pásky a pozice hlavy na pásce. Tyto informace lze uchovat v programu následovně:

- **páska** může obsahovat symboly $\{0, 1, \Delta\}$, ty budeme kódovat takto:
  - Δ jako žádný šroubek
  - `1` jako 1 šroubek
  - `0` jako 2 šroubky

  a uchováme je v řadě za sebou na jednom řádku matice (např. na řádku s indexem 0)
- **pozici hlavy** lze zapsat pomocí jednoho šroubku, který se bude přesouvat na řádku č. 1, kde pozice šroubku $(i, 1)$ bude značit hlavu na pozici i
- **Stavy** Q lze očíslovat indexy $0 - \mathbb{N}$ a využít řádek matice č. 2, kde pole ve sloupci i bude představovat stav q_i. Na tomto řádku se bude nacházet právě jeden šroubek, který bude symbolizovat aktuální stav. Tedy TS bude ve stavu q_5 tehdy, když bude na poli (5,2) šroubek.

Počáteční konfigurace bude tedy na řádku 0 obsahovat zakódovanou vstupní pásku začínající sloupcem 0. Dále jeden šroubek na pozici $(0, 1)$ značící pozici čtecí hlavy na indexu 0, jeden šroubek na $(0, 2)$ značící počáteční stav q_0 a po třech šroubcích na pozicích $(-1, 0)$, $(-1, 1)$ a $(-1, 2)$ sloužící jako zarážky.

S využitím `jdiKeSroubku` a `sbirejSroubky` lze navrhnout následující program:

1. Karel začne na prvním znaku vstupní pásky (tedy na pozici $(0, 0)$).
2. Postupně odebere šroubky a ve stavovém řízení si uloží jejich počet.
3. Vrátí šroubky zpět a na základě jejich počtu provede skok v programu do jedné ze tří sekcí (každá ze sekcí realizuje přechodovou funkci pro daný symbol).
4. Karel se přesune na pozici $(0, 2)$, (lze realizovat díky zarážce).
5. Postupně se prochází řádek 2 a hledá se šroubek (index pozice odpovídá indexu aktuálního stavu).

6. V programu je kódováno co se má provést, tedy změna indexu stavu (realizován jako n kroků a umístění šroubku) a buď posun čtecí hlavy o 1 doprava/doleva (je nutné hlídat překročení zarážky) nebo změna symbolu na pozici čtecí hlavy.

7. Karel se vrátí zpět na aktuální pozici na pásce a pokračuje bodem 2.

V případě, že TS z aktuálního stavu pomocí načteného symbolu přechází do koncového stav, program vrátí 1. Pokud pro daný symbol neexistuje následující stav, vrací 0.

a) Pro každý program P_M lze sestrojit TS M

Pro jednoduchost navrhneme více TS, které posléze spojíme v jeden vícepáskový.

- Dvojrozměrnou matici lze zakódovat následovně: $m1^x0m1^y01^s00$, kde x, y představují souřadnice pole a s počet šroubků na tomto poli a m je buď prázdné a nebo obsahuje symbol − pro záporné indexy. Pokud nějaké pole na pásce není, tak obsahuje 0 šroubků.
  Lze sestavit dvoupáskový TS A, který na první pásce dostane vstup ve tvaru $< op > m1^x0m1^y$ který vyhledá pole (x, y) na druhé pásce a na základě kódu operace *op* (viz dále) zjistí zda obsahuje šroubek (a zapíše na první pásku *True* nebo *False*), popř. šroubek odebere nebo přidá. Pokud pole neexistuje, a má se na něj přidat šroubek, tak ho vytvoří na konci druhé pásky.

- Aktuální pozici robota lze kódovat následovně: $m1^x0m1^y01^d0$, kde x, y jsou souřadnice a d nabývá hodnotu 1-4 v závislosti na směru pohledu robota a m opět obsahuje znaménko.
  Opět lze sestavit dvoupáskový TS B, který na první pásce dostane vstup ve tvaru $\langle op \rangle$ a na druhé pásce patřičně modifikuje stav robota - tedy otočí ho či udělá krok.

- Příkazy programu P na pásku TS kódujeme jako jedno písmeno $\langle op \rangle$ (`drop-screw` jako a, `lift-screw` jako b, ...). Hodnoty parametrů b a n se budou kódovat unárně jako posloupnost jedniček zapsané na pásce přímo za daným písmenem.
  Navrhneme dvoupáskový TS C, který na první pásce dostane program kódovaný výše uvedeným postupem a druhá páska bude sloužit pro dočasné uložení unárně kódovaných parametrů.

- Pomocí těchto tří TS můžeme realizovat TS M, který přečte a jeden příkaz (TS C) a pak zapíše vhodný vstup na pásku TS A nebo TS B (popř. obou). Skok v programu se provádí přesunem hlavy TS C o n polí (pro jednoduchost si unárně kódované číslo n vykopíruje na pomocnou pásku), kdy je nutné přeskočit pole obsahující 1 (unárně kódované parametry).

Pokud TS C dekóduje příkaz `return` zjistí, zda následuje právě jedna 1. Pokud je jich více (nebo méně), odmítne. V opačném případě daný vstup přijme.