1.

Riešenie

(a) Nech G_D je gramatika generujúca Dyckov jazyk (definícia 4.29[1]) nad jednou dvojicou zátvoriek $\{[,]\}$, definovaná ako $G_D = (S_D, \{[,]\}, P_D, S_D)$. Množina pravidiel P_D obsahuje pravidlá:
$S_D \to \varepsilon \mid S_D S_D \mid [S_D]$
Aby každé neprázdne slovo bolo možné zapísať v tvare $[u]v$, kde $u, v \in L$ (a teda $S_D \underset{G_D}{\Longrightarrow}^* u$ a $S_D \underset{G_D}{\Longrightarrow}^* v$), je zrejmé, že v derivácii musí byť použité pravidlo obsahujúce terminál (t.j. pravidlo $S_D \underset{G_D}{\Longrightarrow} [S_D]$). Ak teda máme deriváciu s pravidlom $S_D \underset{G_D}{\Longrightarrow} [S_D]$ a následne na vnútorne S_D použijeme pravidlo $S_D \underset{G_D}{\Longrightarrow} \varepsilon$, čiže $S_D \underset{G_D}{\Longrightarrow} [S_D] \underset{G_D}{\Longrightarrow} [\ ]$, získavame najmenší možný prefix slova w, ktorý patrí do L, a tým je „[ ]“.

Majme ľavú deriváciu. Postupne prepisujeme vždy najľavejší neterminál. Kvôli neprázdnosti musíme použiť aspoň raz pravidlo $S_D \to [S_D]$. Uvažujme teda jeho prvé použitie.

Pravidlo mohlo byť použité hneď v rámci prepisu najľavejšieho neterminálu. V prípade, že toto použitie nebolo v rámci prepisu najľavejšieho neterminálu, tak pred najľavejšou zátvorkou [vo w je ε, keďže sa pred najľavejšou zátvorkou [mohli použiť len pravidlá $S_D \underset{G_D}{\Longrightarrow} S_D S_D$ a $S_D \underset{G_D}{\Longrightarrow} \varepsilon$.

Za najľavejšou zátvorkou [môžeme S_D ľubovolne prepísať pomocou pravidiel z G_D a teda časť (označme ako u) medzi najľavejšou zátvorkou [a k nej sa viažúcou zátvorkou] musí patriť do L. $S_D \underset{G_D}{\Longrightarrow}^* u, u \in L$.

Za zmienenou zátvorkou] je možné v rámci prepisu možných ďalších neterminálov použiť ľubovolné pravidlá z G_D a teda aj táto časť (označme ako v) musí patriť do L. $S_D \underset{G_D}{\Longrightarrow}^* v, v \in L$.

Všetky neprázdne reťazce patriace do L je teda možné zapísať v tvare $[u]v$, kde $u, v \in L$.

(b) Báza pre $i = 0$:
Pre prípad $i = 0$ máme $\#_{[}(w) = 0$, čiže $w = \varepsilon$. Z gramatík je zrejmé, že existujú derivácie $S_D \underset{G_D}{\Longrightarrow}^* \varepsilon$ a $S \underset{G}{\Rightarrow}^* \varepsilon$ a teda $w \in L$ a taktiež $w \in L(G)$.

Indukčný krok pre $i > 0$:
Predpoklad: Pre prípad $\#_{[}(w) = j \wedge j > 0 \wedge j < i$ platí, že $w \in L(G)$.

Z tvrdenia z podúlohy (a) vieme, že všetky neprázdne reťazce patriace do L je možné zapísať v tvare $[u]v$, kde $u, v \in L$. Nech teda máme u, v také, že platí $\#_{[}(u) + \#_{[}(v) = \#_{[}(w) - 1$. Potom vieme, že $u, v \in L(G)$, čiže $S \underset{G}{\Rightarrow}^* u$ a $S \underset{G}{\Rightarrow}^* v$. Z gramatiky G vidíme, že existuje derivácia $S \underset{G}{\Rightarrow} [S]S \underset{G}{\Rightarrow}^* [u]v$, kde $[u]v$ označme ako w. Vidíme, že $\forall w \in L \wedge \#_{[}(w) = i : w = [u]v \wedge u, v \in L \wedge u, v \in L(G) \to [u]v \in L(G) \to w \in L(G)$. A teda, $\forall w \in L : w \in L(G)$. Tvrdenie $L \subseteq L(G)$ platí.

[1] http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/TIN-studijni-text.pdf

2.

Riešenie

Predpokladajme, že jazyk L_{primes} je bezkontextový jazyk. Potom podľa vety 4.19[2] o Pumping lemme pre BJ platí:

$$\exists k\ > 0 : \forall w \in L_{primes} : |w| \geq k\ \Rightarrow \exists u, v, w, x, y \in \Sigma^* :$$
$$w = uvwxy \wedge\ vx \neq \varepsilon\ \wedge |vwx| \leq k\ \wedge \forall i \geq 0 : uv^i wx^i y \in L_{primes}$$

Nech prvočíslo p je také, že platí $p \geq k$.

Výber reťazca w:
$w = a^p, w \in L_{primes}$
$|w| = p, p \geq k$
$w = a^p = uvwxy$, kde $v = a^r, x = a^s, r + s \leq k, r + s > 0$ a teda $vx \neq \varepsilon$

Iterácia $i = p + 1$:
$uv^{(1+p)}wx^{(1+p)}y \in L_{primes}$
$a^{(p-r-s)+(1+p)r+(1+p)s} \in L_{primes}$
$a^{p+pr+ps} \in L_{primes}$
$a^{p(1+r+s)} \in L_{primes}$ - čo je ale spor, keďže $p(1+r+s)$ nie je prvočíslo, $a^{p(1+r+s)} \notin L_{primes}$

Predpoklad, že jazyk L_{primes} je bezkontextový, neplatí. Jazyk L_{primes} nie je bezkontextový.

[2] http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/TIN-studijni-text.pdf

3.

Riešenie

(a) Skúmaný problém je možné charakterizovať jazykom:
$AF = \{\langle M \rangle \mid M$ je TS, ktorý prijme aspoň jeden reťazec z jazyka $Affine\} \subseteq \{0,1\}^*$
Problém členstva je charakterizovaný jazykom:
$MP = \{\langle M \rangle \# \langle w \rangle \mid M$ je TS taký, že $w \in L(M)\} \subseteq \{0,1,\#\}^*$

Zostavíme redukciu σ: $\{0,1,\#\}^* \to \{0,1\}^*$ z jazyka MP na AF. TS, ktorý implementuje σ, priradí každému vstupu $x \in \{0,1,\#\}^*$ reťazec $\langle Mx \rangle$, kde Mx je TS, ktorý na vstupe $y \in \{0,1\}^*$ pracuje nasledovne:

- Mx vymaže svoj vstup y.
- Zapíše na pásku reťazec x.
- Mx posúdi, či $x = x1\#x2$ pre $x1$, ktoré je kódom TS a $x2$, ktoré je kódom jeho vstupu. Ak nie, odmietne.
- Inak Mx simuluje činnosť TS s kódom $x1$ ma reťazci s kódom $x2$. Ak prijal, Mx prijme tiež, ak odmietol, Mx odmietne tiež alebo ak cyklí, Mx cyklí tiež.

TS $M\sigma$ je možné implementovať úplným TS. Konečne tento TS vypíše kód Mx, ktorý sa skladá zo 4 komponent, ktoré odpovedajú vyššie uvedeným krokom. Tri z nich sú pritom konzistentné, nezávisia na x - konkrétne 1. zmazanie pásky, 2. test na dobré sformovanie inštancie MP, 3. simuláciu daného TS na danom vstupe (pomocou UTS). TS implementujúci tieto kroky, ktoré evidentne existujú, je možné pripraviť dopredu a $M\sigma$ len vypíše kód spolu s kódom na predanie riadenia. Zostavá vygenerovať kód TS, ktorý zapíše na pásku dané x. Toto generovanie je už jednoduché.

Možné jazyky TS Mx:

- $L(Mx) = \emptyset \Leftrightarrow x$ nie je správne sformovaná inštancia MP alebo $x = x1\#x2$ a TS s kódom $x1$ odmietne reťazec $x2$ alebo cyklí
- $L(Mx) = \Sigma^* \Leftrightarrow x$ je správne sformovaná inštancia MP, kde $x = x1\#x2$ a TS s kódom $x1$ prijme reťazec $x2$

Ukážme, že σ zachováva členstvo:
$\langle Mx \rangle \in AF \Leftrightarrow L(Mx) = \Sigma^* \Leftrightarrow x = x1\#x2$, kde $x1$ je kód TS, ktorý prijme vstup s kódom $x2 \Leftrightarrow x \in MP$

(b) Idea dôkazu:
K čiastočnému rozhodnutiu uvedeného problému môžeme zostrojiť TS M', ktorý na svojej páske simuluje beh TS M pre jednotlivé vstupné reťazce z jazyka $Affine$, napr. v lexikografickom usporiadaní. Je teda potrebné skontrolovať všetky vygenerované reťazce z Σ^* a zistiť, či sú alebo nie sú z jazyka $Affine$, a to tak, že TS M' overí rovnicu z definície jazyka $Affine$ pre všetky reťazce. Na reťazcoch, ktoré patria do jazyka $Affine$, simuluje beh TS M. TS M' ale nemôže len systematicky vygenerovať vstupy M a na každom z nich nechať neobmedzene bežať M - v takomto prípade hrozí zacyklenie. Namiesto toho M' môže mať rozbehnutú simuláciu M pre jednotlivé vstupné reťazce. Jednotlivé konfigurácie pásky sú vhodným spôsobom oddelené. M' môže vždy prejsť všetky aktuálne rozbehnuté simulácie a na každej urobiť jeden krok. Ak jedna z nich vedie k prijatiu, prijme. Inak pridá páskovú konfiguráciu pre ďalší reťazec a tento postup opakuje. TS M' teda prijme, ak $L(M)$ obsahuje aspoň jeden reťazec z jazyka $Affine$, inak neskončí.

4.

Riešenie

(a) Pri prevode z TS M na P_M uvažujeme TS M s $\Gamma = \{0, 1, \Delta\}$ z dôvodu jednoduchšieho popisu reprezentácie symbolov z páskovej abecedy. Z tvrdenia v dôkaze vety 6.1.1[3] vieme, že v prípade TS, ktorý používa viacej symbolov, tieto symboly je možné zakódovať do istej postupnosti symbolov z $\{0, 1, \Delta\}$ a zostrojiť ďalší TS, ktorý simuluje TS M.

Nech $w \in L(M)$ potom $x0$ bude definované ako $x0 = f_{rationalc}(reversal(w)\Delta\#)$. Hlava TS bude reprezentovať pozíciu desatinnej bodky. Symbol # bude slúžiť ako zarážka, čo nám umožní ošetriť možnosť prepadnutia hlavy TS, a preto na začiatku programu musíme urobiť jeden posun R, aby sme posunuli za #.

Keďže máme štyri symboly $0, 1, \Delta, \#$, bude potrebné ich zakódovať dvoma bitmi. Kódovacia funkcia $f_{rationalc}$ na jednotlivých symboloch jej vstupu funguje nasledovne:

- Δ zakóduj na 00
- # zakóduj na 10
- 0 zakóduj na 01
- 1 zakóduj na 11

(A) Reprezentácia posunov L, R
L:
x `*=` 2;
x `*=` 2;

R:
x `/=` 2;
x `/=` 2;

Treba dodať, že inštrukcie musíme urobiť dvakrát preto, lebo sme jeden symbol zakódovali na dva bity.

(B) Reprezentácia zápisov
operácia zápisu 1 (11):
x `/=` 2;
⌈ `odd(x)` ⌉;
x `*=` 2;
⌈ `odd(x)` ⌉;

operácia zápisu 0 (01):
x `/=` 2;
⌊ `even(x)` ⌋;

[3] http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/TIN-studijni-text.pdf

x `*=` 2`;`
⌈ `odd(x)` ⌉`;`

operácia zápisu Δ (00):
x `/=` 2`;`
⌊ `even(x)` ⌋`;`
x `*=` 2`;`
⌊ `even(x)` ⌋`;`

(C) Reprezentácia stavov
Každý stav reprezentujeme riadkom v programe a tento riadok nazveme podľa odpovedajúceho stavu. Koncový stav (napr. $r1$) reprezentuje riadok $r1$ s príkazom `return` 1 a nekoncový stav reprezentuje riadok s príkazom `return` 0.

(D) Reprezentácia zmeny stavov
Ukážme si princíp zmeny stavu, napr. na stav $r0$:
`if` x `%` 2 `==` 1 `goto` $r0$`;`
`if` x `%` 2 `==` 0 `goto` $r0$`;`

Vidíme, že tok riadenia programu sa vždy presunie práve na riadok $r0$.

(E) Detekcia symbolu pod hlavou
Ukážme si princíp pre detekciu symbolu 0 (pre ďalšie symboly je princíp podobný):
`if` x `%` 2 `==` 0 `goto` l_0`;`
x `/=` 2`;`
`if` x `%` 2 `==` 1 `goto` l_{11}`;`
⟨⟨ v tomto mieste je už jasné, že ide o bity 01, a teda pod hlavou bola detegovaná 0 ⟩⟩

(F) Prípad prepadnutia hlavy
Symbol # je zarážkou, vďaka ktorej tento prípad ošetríme. Pri každom posune L podľa postupu uvedeného v (D) musíme overiť, či je pod hlavou symbol #. Ak áno, musíme program korektne ukončiť s návratovou hodnotou 0, t.j. pri detekcii symbolu # vykonať príkaz `return` 0.

(G) Prípad nedefinovaného prechodu
Nedefinované prechody reprezentujeme skokom na riadok s príkazom `return` 0, t.j. ako keby tam bol prechod do nekoncového stavu.

(b) Pre w platí, že ak program P s počiatočnou hodnotou x_0 skončí s návratovou hodnotou 1, tak $w = reversal(x_0)$. Keďže x_0 je prirodzené číslo, jeho otočením získame pozíciu desatinnej bodky hneď naľavo. Myšlienkou je, že desatinnú bodku bude reprezentovať hlava TS. V prvom kroku TS M_P presunie hlavu z Δ na prvý znak reťazca. Hlava TS M_P bude teda priamo na pozícii desatinnej bodky. Príkazy v programe je potrebné nejakým spôsobom reprezentovať v TS M_P.

(A) Reprezentácia príkazov

⌈ `odd(x)` ⌉	- operácia zápisu 1
⌊ `even(x)` ⌋	- operácia zápisu 0
x `*=` 2	- operácia posunu doľava (L)
x `/=` 2	- operácia posunu doprava (R)
`if` x `%` 2 `==` b `goto` n	- zmena stavu
`return` 0	- stav q_X
`return` 1	- stav q_F

Pre každý príkaz v programe budeme mať stav v TS. Poradie príkazu v rámci programu bude súčasťou názvu odpovedajúceho stavu. Treba dodať, že ak by sme mali viacero príkazov `return` 1 v programe, mali by sme viacero koncových stavov. V definícii 5.1.1[3] pre TS sa avšak hovorí o jednom koncovom stave s dodatkom, že existujú alternatívne definície TS, ktoré hovoria o množine koncových stavov. Tento „problém“ vyriešime tak, že v takomto prípade vieme vytvoriť taký TS, kde z týchto pôvodne koncových stavov urobíme nekoncové a vytvoríme z nich prechod do jedného koncového stavu.

(B) Práca so symbolmi Δ na páske

Je zrejmé, že môže dôjsť k prípadu prepadnutia hlavy, ak sa dostaneme na Δ. Tento prípad je potrebné ošetriť a umožniť pokračovanie simulácie programu. Problém môžeme vyriešiť tak, že pri každom posune doľava skontrolujeme, či pod hlavou nie je Δ. Ak je pod hlavou Δ, je nutné urobiť nasledovné kroky: $R_\Delta S_R L_\Delta 0$ (t.j. posunúť sa doprava na najbližší symbol Δ, posunúť pásku doprava, posunúť sa doľava na najbližší symbol Δ, zapísať 0). Pri každom posune doprava tiež skontrolujeme, či pod hlavou nie je Δ. Ak áno, zapíšeme 0.