Teoretická informatika, FIT, VUT Brno

2. domácí úloha

David Mihola
xmihol00

26. listopadu 2023

1 Náležitost jazyka L_{prime} do $\mathcal{L}_2$

Bezkontextové gramatiky a zásobníkové automaty zřejmě nemají takovou sílu, aby dokázaly generovat, respektive přijímat prvočísla. Následující sekce dokazuje, že jazyk L_{prime} není bezkontextový.

1.1 Důkaz $L_{prime} \notin \mathcal{L}_2$

Pro důkaz nebezkontextovosti je vhodné použít obměněnou implikaci s *pumping lemma* pro $\mathcal{L}_2$ a na základě ní provést přímý důkaz[1]. Obměněná implikace pro jazyk L_{prime} vypadá následovně:
$(\forall k \in \mathbb{N}^+ : \exists z \in \{a,b\}^* : z \in L_{prime} \wedge |z| \geq k \wedge \forall u,v,w,x,y \in \{a,b\}^* : uvwxy = z \wedge |vx| > 0 \wedge |vwx| \leq k \Rightarrow \exists i \in \mathbb{N} : uv^iwx^iy \notin L_{prime}) \Rightarrow L_{prime} \notin \mathcal{L}_2$.
Samotný důkaz je poté veden v následující posloupnosti kroků:

1. Zvolení $z = a^p$, kde $p = nextPrime(k)$. Funkce $nextPrime$ vrací prvočíslo větší nebo rovno k. Eukleidova věta o prvočíslech říká, že prvočísel je nekonečně mnoho, takže tato funkce je definovaná pro libovolně velké k.

2. Všechna možná rozdělení z na u, v, w, x, y za použití parametrů $o, q, r, s, t \in \mathbb{N} \wedge o+q+r+s+t = p \wedge q+s > 0 \ \wedge q+r+s \leq k$ budou vypadat následovně:

   - $u = a^o$,
   - $v = a^q$,
   - $w = a^r$,
   - $x = a^s$,
   - $y = a^t$.

3. Pro volbu $i = p+1$[2] slovo $uv^{p+1}wx^{p+1}y$ má délku $l = o + q \cdot (p+1) + r + s \cdot (p+1) + t = o + q + r + s + t + q \cdot p + s \cdot p = p + q \cdot p + s \cdot p = p \cdot (1 + q + s)$.

4. Délka slova l je zřejmě beze zbytku dělitelná i jinými čísly než 1 a l, a to např. číslem p, pro které platí $l > p > 1$, protože $q + s > 0$.

5. Slovo $uv^{p+1}wx^{p+1}y \notin L_{prime}$.

6. Platí levá strana implikace, tudíž, aby byla zachována platnost celého výroku, musí platit i její pravá strana a tím pádem $L_{prime} \notin \mathcal{L}_2$. □

[1] Obměnou dojde k znegování obou stran implikace. Důkaz má tedy podobný průběh jako důkaz sporem vycházející z původní implikace.
[2] Volba i byla inspirována http://www.cs.rpi.edu/courses/fall00/modcomp3/sol6.pdf.

2 Rozhodnutelnost jazyka L_{BKG}

Zadaný jazyk L_{BKG} je (c) nerozhodnutelný a není ani částečně rozhodnutelný. Pro důkaz použijme redukci ze známého jazyka inkluze bezkontextových gramatik $L_{\subseteq BKG} = \{\langle G_1\rangle\#\langle G_2\rangle \mid L(G_1) \subseteq L(G_2) \land G_1, G_2 \text{ jsou bezkontextové gramatiky}\}$, který není ani částečně rozhodnutelný. Dokážeme, že $L_{\subseteq BKG} \leq L_{BKG}$.

2.1 Důkaz

Redukční funkce bude ve tvaru $\sigma : \{0, 1, \#\}^* \rightarrow \{0, 1, \#\}^*$.
Pokud je vstup redukční funkce řetězec ve tvaru $\langle G_1\rangle\#\langle G_2\rangle$, kde $\langle G_1\rangle$ a $\langle G_2\rangle$ jsou validní kódy bezkontextových gramatik[3], pak jej funkce zobrazí na $\langle M\rangle\#\langle G_1\rangle$, kde:

- G_1 pouze zkopíruje,
- Turingův stroj M naprogramuje tak, aby obsahoval zásobníkový automat přijímající jazyk $L(G_2)$, např. za použití algoritmu převodu bezkontextové gramatiky na ZA analýzou shora dolů, a aby TS M přijal svůj vstup právě tehdy, když jej přijme i ZA[4], jehož simulaci TS M spustí nad svým vstupem.

Pro zbylé, tj. nevalidní, vstupy, redukční funkce vrátí řetězec $\langle M_x\rangle\#\langle G_x\rangle$, kde $L(M_x) = \emptyset$ a $L(G_x) = \{a\}$.
Zřejmě platí:

$$\langle G_1\rangle\#\langle G_2\rangle \in L_{\subseteq BKG} \Rightarrow L(G_1) \subseteq L(G_2) \Rightarrow L(G_1) \subseteq L(M) \Rightarrow \langle M\rangle\#\langle G_1\rangle \in L_{BKG}$$

$$\langle G_1\rangle\#\langle G_2\rangle \notin L_{\subseteq BKG} \Rightarrow L(G_2) \subset L(G_1) \Rightarrow L(M) \subset L(G_1) \Rightarrow \langle M\rangle\#\langle G_1\rangle \notin L_{BKG}$$

$$\langle G_1\rangle\#\langle G_2\rangle \in L_{\subseteq BKG} \iff \sigma(\langle G_1\rangle\#\langle G_2\rangle) \in L_{BKG}$$

O redukční funkci tedy můžeme říci, že zachovává příslušnost. Dále lze říci, že redukční funkce je totální, rekurzivně vyčíslitelná funkce, protože provádí pouze syntaktickou analýzu a algoritmický převod BKG na ZA, což je možné pro libovolnou BKG. □

[3]To lze poznat z tvaru pravidel syntaktickou analýzou.
[4]Cyklení ZA zajisté lze detekovat, protože jsme si na přednáškách ukázali, že cyklení lze detekovat i u lineárně omezeného automatu, který má větší výpočetní sílu.

3 Kardinalita množin - binární strom

Následující sekce obsahují důkazy kardinality množin zadaných problémů. Oba důkazy jsou inspirované paradoxem známým jako Hilbertův hotel[5].

3.1 Důkaz $|V| = |\mathbb{N}|$

Definujme $\forall i \in \mathbb{N} : V_i = \{v \in V \mid$ délka cesty mezi r a v, na které se libovolný vrchol nachází maximálně jednou, je rovna $i\}$ a $\forall k \in \mathbb{N} : prime(k)$ vrací k-té prvočíslo[6]. Eukleidova věta o prvočíslech říká, že prvočísel je nekonečně mnoho. Bijekce mezi V a podmnožinou $\mathbb{N}$ lze tedy najít tak, že vrcholu $v_j \in V_i$, kde $j \in \mathbb{N}^+$, je přiřazeno číslo $prime(i)^j$.
Ačkoliv výše uvedený text dokazuje[7], že množina V je spočetně nekonečná, text přímo neodpovídá požadavku ze zadání, a to nalezení bijekce mezi V a $\mathbb{N}$. Bijekce mezi V a celou množinou $\mathbb{N}$ lze definovat tak, že vrcholu v_j, kde j značí jeho index v seřazených posloupnostech vrcholů množin V_i, je přiřazeno číslo $2^i + j - 1$[8]. $\square$

3.2 Důkaz $|T_{all}| > |\mathbb{N}|$

Definujeme funkci $binLsbFirst : \mathbb{N} \rightarrow b_0b_1b_2...$, kde $b_0b_1b_2...$ je binarní kód s *least significant bit* zapsaným vlevo[9]. Dále pro $\forall i \in \mathbb{N} : binLsbFirst(i) = b_{i0}b_{i1}b_{i2}...$ definujme funkci obarvení stromu $T_i = \{(0, color(b_{i0})), (1, color(b_{i1})), (2, color(b_{i2})), ...\} \in T_{all}$, kde $color(1) = red$ a $color(0) = black$. Nakonec vytvořme *level order* průchodem binárního stromu seřazenou posloupnost vrcholů $V_{levelOrder}$[10].
Nyní sestavme nekonečnou matici M, o které předpokládáme, že obsahuje všechna možná obarvení stromu:

$$\begin{array}{ccccc} & v_0 & v_1 & v_2 & \cdots \\ T_0 & T_{all}(0,0) & T_{all}(0,1) & T_{all}(0,2) & \cdots \\ T_1 & T_{all}(1,0) & T_{all}(1,1) & T_{all}(1,2) & \cdots \\ T_2 & T_{all}(2,0) & T_{all}(2,1) & T_{all}(2,2) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array} \equiv \begin{array}{ccccc} & v_0 & v_1 & v_2 & \cdots \\ T_0 & black & black & black & \cdots \\ T_1 & \color{red}{red} & black & black & \cdots \\ T_2 & black & \color{red}{red} & black & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

Pro tuto matici platí, že i-tý řádek odpovídá obarvení stromu daného funkcí T_i a j-tý sloupec odpovídá obarvení j-tého vrcholu z $V_{levelOrder}$ ve všech stromech. Výraz $T_{all}(i, j)$ pak zřejmě přiřazuje barvu j-tému vrcholu v i-tém obarvení stromu.

[5] https://youtu.be/OxGsU8oIWjY?si=NB2LbEcDWHLxoiRn

[6] $prime(0) = 2$, $prime(1) = 3$, $prime(2) = 5$ atd.

[7] Důkaz by nebylo nutné měnit pro libovolný n-arní strom, kde $n \in \mathbb{N}^+$.

[8] Zřejmě i tento důkaz lze zobecnit na n-ární strom, přiřazení čísel je v tomto případě dáno výrazem $n^i + j - 1$ pro $n \in \mathbb{N} \setminus \{0, 1\}$. V tomto případě ale také není dodržena bijekce mezi V a celou množinou $\mathbb{N}$ pro $n > 2$.

[9] $binLsbFirst(0) = 000...$, $binLsbFirst(1) = 100...$, $binLsbFirst(2) = 010...$, $binLsbFirst(7) = 111...$ atd.

[10] I takto by bylo možné najít bijekci mezi V a $\mathbb{N}$.

Nyní sestavme funkci obarvení stromu $T_{missing} = \{(i, invertColor(T_{all}(i,i))) \mid \forall i \in \mathbb{N}\}$, kde $invertColor(red) = black$ a $invertColor(black) = red$. Obarvení stromu definované touto funkcí se jistě liší od všech obarvení stromu v matici M barvou alespoň jednoho vrcholu, a to vrcholu ležícího na hlavní diagonále.
V předchozím odstavci jsme dospěli ke sporu, že matice M obsahuje všechna možná obarvení stromu. Z toho plyne, že množina všech obarvení stromu T_{all} je nespočetně nekonečná. $\square$

4 Algoritmus a výpočet množiny ${}_aN_a$

Algoritmus pro výpočet množinu ${}_aN_a$ sestavíme za pomocí:

- množiny N_t obsahující neterminální symboly, ze kterých lze vygenerovat řetězec terminálních symbolů,
- množiny ${}_aN$ obsahující neterminální symboly, ze kterých lze vygenerovat řetězec terminálních symbolů začínající symbolem a,
- množiny N_a obsahující neterminální symboly, ze kterých lze vygenerovat řetězec terminálních symbolů končící symbolem a

a množiny N_ε.
Následně provedeme výpočet aplikováním algoritmu na zadaný problém.

4.1 Definice algoritmu

Nejprve definujeme algoritmy pro výpočet pomocných množin N_t, ${}_aN$ a N_a. Výstupy těchto algoritmů následně použijeme pro definici algoritmu pevného bodu pro výpočet množiny ${}_aN_a$.

4.1.1 Definice algoritmů pro pomocné množiny

Algoritmy pro výpočet pomocných množin také definujeme jako algoritmy pevného bodu.

Algoritmus pevného bodu pro výpočet N_t

procedure $N_t(N, P, N_\varepsilon)$
    $N_0 \leftarrow \emptyset$
    $i \leftarrow 0$
    do
        $i \leftarrow i + 1$
        $N_i \leftarrow \{A \in N \mid \exists(A \to \alpha) \in P \land \alpha \in (N_{i-1} \cup \Sigma)^*\}$
    while $N_i \neq N_{i-1}$
    return N_i
end procedure

Algoritmus pevného bodu pro výpočet ${}_aN$

procedure ${}_aN(N, P, N_\varepsilon, N_t)$
  $N_0 \leftarrow \emptyset$
  $i \leftarrow 0$
  do
    $i \leftarrow i + 1$
    $N_i \leftarrow \{A \in N \mid \exists (A \rightarrow \alpha) \in P \land \alpha \in N_\varepsilon^*(N_{i-1} \cup \{a\})(N_t \cup \Sigma)^*\}$
  while $N_i \neq N_{i-1}$
  return N_i
end procedure

Algoritmus pevného bodu pro výpočet N_a

procedure $N_a(N, P, N_\varepsilon, N_t)$
  $N_0 \leftarrow \emptyset$
  $i \leftarrow 0$
  do
    $i \leftarrow i + 1$
    $N_i \leftarrow \{A \in N \mid \exists (A \rightarrow \alpha) \in P \land \alpha \in (N_t \cup \Sigma)^*(N_{i-1} \cup \{a\})N_\varepsilon^*\}$
  while $N_i \neq N_{i-1}$
  return N_i
end procedure

4.1.2 Algoritmus pevného bodu pro výpočet ${}_aN_a$

procedure ${}_aN_a(N, P, N_\varepsilon, N_t, {}_aN, N_a)$
  $N_0 \leftarrow \{A \in N \mid \exists (A \rightarrow \alpha) \in P \land \alpha \in N_\varepsilon^*({}_aN \cup \{a\})(N_t \cup \Sigma)^*(\{a\} \cup N_a)N_\varepsilon^*\}$
  $i \leftarrow 0$
  do
    $i \leftarrow i + 1$
    $N_i \leftarrow \{A \in N \mid \exists (A \rightarrow \alpha) \in P \land \alpha \in N_\varepsilon^* N_{i-1} N_\varepsilon^*\} \cup N_{i-1}$
  while $N_i \neq N_{i-1}$
  return N_i
end procedure

4.2 Aplikace algoritmu na zadaný problém

Zřejmě $N_\varepsilon = \{Y\}$. Dále vypočteme v potřebném pořadí pomocné množiny dle algoritmů uvedených výše. Pomocné množiny následně využijeme pro výpočet požadované množiny ${}_aN_a$.

4.2.1 Výpočet pomocných množin

Následují průběhy výpočtů pomocných množin dle definovaných algoritmů výše.

Výpočet N_t

1: $N_0 \leftarrow \emptyset$
2: $N_1 \leftarrow \{Y\}$
3: $N_2 \leftarrow \{Y, U\}$
4: $N_3 \leftarrow \{Y, U, W\}$
5: $N_4 \leftarrow \{Y, U, W, S\}$
6: $N_t = N_4 = N_5 \leftarrow \{Y, U, W, S\}$

Výpočet ${}_aN$

1: $N_0 \leftarrow \emptyset$
2: $N_1 \leftarrow \{S, W\}$
3: ${}_aN = N_1 = N_2 \leftarrow \{S, W\}$

Výpočet N_a

1: $N_0 \leftarrow \emptyset$
2: $N_1 \leftarrow \{S, U\}$
3: $N_2 \leftarrow \{S, U, W\}$
4: $N_a = N_2 = N_3 \leftarrow \{S, U, W\}$

4.2.2 Výpočet ${}_aN_a$

Zadaná gramatika neobsahuje jednoduchá pravidla, ve kterých by se projevila situace[11], kdy by v N_0 nebyly již všechny odpovídající neterminální symboly. Průběh výpočtu tedy bude následující:

1: $N_0 \leftarrow \{A \in N \mid \exists(A \rightarrow \alpha) \in P \land \alpha \in \{Y\}^*(\{S, W\} \cup \{a\})(\{Y, U, W, S\} \cup \Sigma)^*(\{a\} \cup \{S, U, W\})\{Y\}^*\} = \{S, W\}$
2: ${}_aN_a = N_0 = N_1 \leftarrow \emptyset \cup \{S, W\}$

a získáváme ${}_aN_a = \{S, W\}$.

[11] Např. po přidání pravidla $S' \rightarrow S$ by to tak již nebylo.