2014P.1

1

L – zbiór w nad {0, 1} takich że wśród ostatnich 9 liter w jest parzyście wiele “1”.

T: Indeks L = 512.

Dowód, że ≥ 512:

Załóżmy, że DFA A rozpoznający L ma < 512 stanów. Wtedy istnieją dwa różne 9-literowe słowa z {0, 1}*, które skończą w tym samym stanie. Weźmy ostatni indeks, na którym się różnią i dopiszmy same jedynki na końcu, tak żeby zaczynały się od tego indeksu. Jedno będzie miało parzystą ilość jedynek, inne nie, a A da taką samą odpowiedź dla obu – sprzeczność.

A z twierdzenia o indeksie? (To by od razu dało potwierdzenie).

Chcemy pokazać, że liczba klas abstrakcji ~L to 512.
Konkretnej – jeżeli słowa różnią się gdzieś na 9-literowym prefiksie, to są w różnych klasach abstrakcji. W sumie to jest dowód jak poprzednio, ale z tw. daje nam od razu, że tyle wystarczy.

2005.1

1

T: L regularny $\Rightarrow$ $L/2 = \{ w \in \{0, 1\}^* \mid ww \in L \}$ regularny

Załóżmy, że L regularny. Wtedy istnieje DFA (Q_A, $\sum$, ∂_A, $q0_A$, F_A) rozpoznający L – weźmy taki automat. (Niech $n = |Q_A|$). Pokażemy, że L/2 też jest regularny, konstruując DFA rozpoznający L/2.

Idea: chcemy zacząć od $q0_A$, po wczytaniu w wylądować w stanie $qx \in Q_A$ i ocenić, czy po wczytaniu w jeszcze raz zaczynając z qx wylądowaliśmy w jakimś stanie $qy \in F_A$. Nie wiemy, jaki będzie ten qx, więc chcemy "odtworzyć" trasę w z każdego możliwego stanu – dlatego każdy stan z Q_B będzie n-krotką stanów z Q_A, a stan początkowy $q0_B$ – uporządkowanymi stanami z Q_A. Wtedy po wczytaniu w znajdziemy się w stanie $q = (qx, \dots)$. Chcemy ocenić, czy jakbyśmy startowali ("drugi raz") z qx, to znaleźlibyśmy się $qy \in F_A$ – w tym celu patrzymy na q[x] i sprawdzamy, czy należy on do F_A (w sensie takie stany już są akceptujące).

Niech:

- $Q_B = Q_A^{|Q_A|}$
- $\partial_B((q1, q2, \dots, qn), a) = (\partial_A(q1, a), \partial_A(q2, a), \dots, \partial_A(qn, a))$
- $q0_B = (q0_A, q1_A, \dots, qn_A)$
- $F_B = \{ q \in Q_B \mid q[q[0]] \in F_A \}$

Wtedy DFA rozpoznający L/2 to (Q_B, $\sum$, ∂_B, $q0_B$, F_B).

2

$\sum = \{a, b, A, B\}$
$L = \{ w \in \sum^* \mid (|wa| + |wA| = |wb| + |wB|) \land (|wa| + |wb| = |wA| = |wB|) \}$

Na logikę nie, bo PDA musiałby mieć dwa stosy.

Jeśli L byłby CFL, to $L \cap L(a^*b^*A^*B^*)$ też.
Pokażemy, że nie jest, używając lematu o pompowaniu dla CFL.
Niech p – stała z lematu, weźmy $w' = a^p b^p A^p B^p$. Wiemy, że $w' \in L$.

Niech $w' = uvwxy$ tak że $|vwx| \le p$, $|vx| \ge 1$.
Gdzie byśmy nie trafili, zawsze coś zepsujemy.

3

Light:
S → SIMPLEXPR | COMPLEXPR
EXPR → (COMPLEXPR) | SIMPLEXPR
COMPLEXPR → EXPR ⇔ EXPR
SIMPLEXPR → 0 | 1

Hard:
S → SIMPLEXPR_T | COMPLEXPR_T
EXPR_T → (COMPLEXPR_T) | SIMPLEXPR_T
EXPR_F → (COMPLEXPR_F) | SIMPLEXPR_F
COMPLEXPR_T → EXPR_T ⇔ EXPR_T | EXPR_F ⇔ EXPR_F
COMPLEXPR_F → EXPR_T ⇔ EXPR_F | EXPR_F ⇔ EXPR_T
SIMPLEXPR_T → 1
SIMPLEXPR_F → 0

2005.2

2

T: SUBCFL = { (G1, G2) | L(G1) ⊆ L(G2) } nie jest rekurencyjny.

Wskazówka 1: PCP ≤ SUBCFL.
Wskazówka 2: język słów które nie są palindromami (NPAL) jest CFL.

Dają nam instancję PCP = (g[], h[]).
Chcemy pokazać, że przy pomocy SUB CFL da się rozstrzygnąć PCP.

Możemy skonstruować gramatyki generujące:
(Nie tak, ale poprawnie jest trudniej).

G → ε | g[0] G 0 | … | g[n] G n
H → ε | h[0] H 0 | … | h[n] H n

Ok, to jest wersja specjalna L(G) = ∑*, tj. ∑* ⊆ L(G) ⇔ L(G) = ∑*.

Dobra, bez redukcji, po prostu:
Załóżmy, że SUBCFL jest rozstrzygalny. Wtedy potrafilibyśmy skonstruować algorytm ϕ rozstrzygający PCP:

- wczytaj g[], h[]
- skonstruuj CFG G:

  S → # | g[0] S h[0]^R | g[1] S h[1]^R | … | g[n] S h[n]^R
- zwróć negację SUBCFL(G, NPAL)

Musimy pokazać, że:

1. PCP(g[], h[]) ma rozwiązanie $\Rightarrow \phi$(g[], h[]) zwraca 1
2. PCP(g[], h[]) nie ma rozwiązania $\Rightarrow \phi$(g[], h[]) zwraca 0

Dowód 1.: Załóżmy, że PCP(g[], h[]) ma rozwiązanie. Wtedy istnieje ciąg i[] indeksów taki że g[i0] g[i1] … g[im] = h[i0] h[i1] … h[im]. Wtedy g[i0] g[i1] … g[im] # h[im]^R … h[i1]^R h[i0]^R jest palindromem, więc CFG G potrafi wygenerować palindrom, więc L(G) nie jest podzbiorem L(NPAL). SUBCFL zwróci 0, więc ϕ – 1.

Dowód 2.: Analogicznie.

3

Problem słów semiThue(∑, R, w1, w2) pozostaje nierozstrzygalny dla semiprocesów Thuego, gdzie każda R jest postaci:

- a → w, gdzie a ∈ ∑, w ∈ ∑*
- ab → ε, gdzie a, b ∈ ∑

Gdyby były tylko produkcje postaci 1., to byłby rozstrzygalny (dowód – równoważność CFG). Tak jest w zasadzie równoważny zad. 127 (zbiór 2020).

2005.3

1

Co jest instancją poprawnie zaplanowanej sesji?
Dane:

- lista przedmiotów P = [p1, p2, …, pn]
- rodzina zbiorów przedmiotów, które nie mogą być razem NP = {{p1, p2}, {p1, p3}, …, {}}

Wynik:

- długość minimalnego ciągu zbiorów przedmiotów takiego że
    - żaden z jego elementów nie jest podzbiorem żadnego zbioru z NP
    - razem dają P bez powtarzania (bo to funkcja, więc musi być jednoznaczna).
      [{p1, p2}, {p3, p4, p7}, {p5}, …]

Obserwacja: kolejność zbiorów w wyniku nie ma znaczenia.
Chcemy pokazać, że jeżeli mamy algorytm, który to rozstrzyga w PTIME, to P = NP.
Czyli musimy wskazać NP-zupełny problem, który da się rozwiązać tym algorytmem.

2

Co tak naprawdę znaczy, że P1, P2, …, Pk "siedzą przy jednym stole"?
Niech T(p, t) oznacza, że ambasador państwa p siedzi przy stole t, $t \in N$.

Mamy warunki postaci (dla każdego $t \in N$):

- $T(p1, t) \wedge T(p2, t) \wedge \ldots \wedge T(pk, t) \Rightarrow T(q1, t) \vee \ldots \vee T(ql, t)$
- $\sim(T(p1, t) \wedge \ldots \wedge T(pk, t))$
- $T(p, t) \Rightarrow \sim T(p, t')$, dla każdego $t' \neq t$

Odpowiadają im klauzule:

- $\sim T(p1, t) \vee \sim T(p2, t) \vee \ldots \vee T(q1, t) \vee \ldots \vee T(ql, t)$
- $\sim T(p1, t) \vee \ldots \vee \sim T(pk, t)$
- $\sim T(p, t) \vee \sim T(p, t')$, dla każdego $t' \neq t$

No i to się da zapisać w CNFie, ale "literałami" są formuły T.
Czy da się je zapisać samymi zmiennymi? Jeżeli ilość stołów da się z góry ograniczyć, to pewnie tak – i pewnie da się. Wtedy mamy formułę logiczną, które da się zapisać w CNF, więc na pewno da się znaleźć rozwiązanie w PSPACE (min-max). A jak pokazać, że nie da się lepiej? Redukcja z QBF, że niby dowolną formułę w CNF da się przedstawić jako instancję protokołu bankietów dyplomatycznych? Tylko czemu odpowiadają te stoły, że ich jest nie wiadomo ile?

3

L3 = { w ∈ {a, b}* | |w|a ≤ |w|b ≤ 2|w|a }

T: L3 jest CFL.

Warunek znaczy tyle, że na każdą literę ‘b’ ma przypadać 1 lub 2 litery ‘a’.
Możemy skonstruować PDA A rozpoznający L3. Idea jest taka, żeby na stosie mieć różnicę między ilością wczytanych ‘a’ i ‘b’ i niedeterministycznie rozstrzygać, czy spełnia ona proporcję 1 ≤ |w|b / |w|a ≤ 2. Jeśli wczytujemy najpierw ‘a’, potem ‘b’, to:

- na każde wczytane ‘a’ wrzucamy na stos ‘a’;
- po każdym wczytaniu ‘b’ zdejmujemy ze stosu 1 lub 2 symbole.

Nie możemy mieć nic poniżej dna stosu, więc jest problem, jeśli wczytamy ‘b’ przy pustym stosie. W tym momencie musimy zakodować negatywną ilość symboli ‘a’ innym symbolem ‘-a’; w dowolnym momencie na stosie będzie tylko 1 rodzaj symbolu. Automat akceptuje słowo, jeśli w danym momencie stos jest pusty.

Opis A:

Taśma	Stos	Zachowanie
a	a | ε	Wrzuć ‘a’ na stos
a	-a	Zdejmij symbol
b	-a | ε	Niedeterministycznie wrzuć 1 lub 2 razy ‘-a’ na stos
b	a	Niedeterministycznie zdejmij 1 lub 2 symbole

4

Lv = { a^i b^j c^k d^l | i = k ∧ j = l }

T: Lv nie jest CFL.

Skorzystamy z lematu o pompowaniu dla CFL, niech p będzie stałą z lematu. Weźmy słowo s = a^p b^p c^p d^p; wiemy, że s ∈ Lv. s Z lematu s =uvwxy, gdzie:

- |vwx| ≤ p
- |vx| ≥ 1

Gdzie by nie wypadło vwx w słowie s, zawsze coś zepsujemy, usuwając litery (rozpatrywanie przypadków).

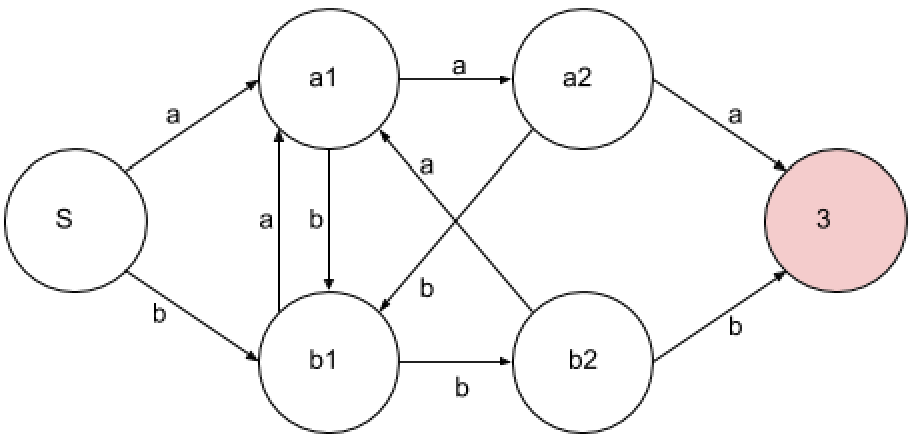
1999P.1

1

L = { w ∈ {a, b}* | w nie zawiera ‘aaa’ ani ‘bbb’ }

To jest przecięcie takich niezawierających jednego z nich.

Automat:



Minimalna liczba stanów: 6?

Z tw. o indeksie; kl. abstrakcji – sufiksy:

1. ε
2. 1x ‘a’
3. 1x ‘b’
4. 2x ‘a’
5. 2x ‘b’
6. 3x ‘a’ lub 3x ‘b’

Trzeba pokazać, że te kl. abstrakcji są rozłączne.

2

Dla CFG G1, G2 problem INF-INT(G1, G2) definiujemy: czy L(G1) ∩ L(G2) jest nieskończony?

T: INF-INT jest nierozstrzygalny.

Pokażemy, że PCP ≤ INF-INT.
Dają nam instancję PCP(g[], h[]), niech to będzie nad alfabetem ∑.

Tworzymy CFG G1:

$S \to \# \mid g[1]\ S\ h[1]^R \mid \ldots \mid g[n]\ S\ h[n]^R$

I drugą G2 do generowania palindromów:

$S \to \# \mid a\ S\ a$ dla każdego $a \in \Sigma$

I mamy instancję INF-INT(G1, G2).

Dowód poprawności:

1. PCP ma rozwiązanie ⇒ L(G1) ∩ L(G2) jest nieskończony.
   Jeżeli PCP ma rozwiązanie, to z G1 można wygenerować palindrom, a potem cyklicznie kolejny i kolejny dokładając kolejne w środku, tj. $wSw^R \to wwSw^Rw^R \to \ldots$, więc w L(G1) jest nieskończona ilość palindromów, więc przecięcie z CFL palindromów jest też nieskończone.
2. PCP nie ma rozwiązania ⇒ L(D1) ∩ L(G2) jest skończony.
   Jeżeli PCP nie ma rozwiązania, to z G1 nie wygenerujemy żadnego palindromu, więc przecięcie z CFL palindromów będzie puste, a zatem skończone.

2001P.1

1

Język słów nad {a, h}, które nie zawierają ‘haha’.

Minimalna liczba stanów: 5 (z tw. o indeksie).

2

L – język słów, których każdy prefiks ma przynajmniej tyle samo ‘b’ co ‘a’.

Intuicja: każde ‘a’ musi być poprzedzone przynajmniej jednym ‘b’.

$S \to T \mid TS$

$T \to bTa \mid bT \mid \varepsilon$

3

L – język z 2.

Czy $L \cap L^R$ jest bezkontekstowy?

Każdy prefiks i każdy sufiks ma mieć przynajmniej tyle ‘b’ co ‘a’.

Czyli każde ‘a’ musi mieć przynajmniej jedno ‘b’ przed i za sobą.

Tak, można pewnie stworzyć PDA, albo taką gramatykę:

$S \to TL\ TR \mid TL\ S\ TR$
$TL \to b\ TL\ a \mid b\ TL \mid \varepsilon$
$TR \to a\ TR\ b \mid TR\ b \mid \varepsilon$

1999.2

2

$A \subseteq N \times N$ jest rekurencyjny.
$B = \{ n \mid$ istnieje nieskończenie wiele m takich, że $(n, m) \in A \}$

Czy B jest r.e.?
Raczej nie, formułę da się zapisać jako: $\forall m0\ \exists m > m0\ (n, m0) \in A \Rightarrow (n, m) \in A$

Przykład: $(a, b) \in A$ jeśli ϕ_a NIE zatrzymuje się po b krokach.
Wtedy przy pomocy B można by semirozstrzygać K', co daje sprzeczność.

2014.2

1

f – częściowa funkcja rekurencyjna, taka że:

- dla każdego $j \in n$ istnieje $i \in N$ takie że $f(i) = j$,
- dla każdych $i, j \in N$ $i < j \Rightarrow f(i) < f(j)$.

Czyli wychodzi na to, że f to prawie identyczność, tyle że "z dziurami", tj.:

x	0 1 2 3 4 5 6 7
f(x)	0 1 – – 2 3 – 4