Operačná stavová zložitosť operácii zjednotenie, prienik a zreťazenie na unárnych automatoch s polovicou stavov koncových

Analýza a návrh riešenia diplomovej práce

Autor práce: Bc. Petra Plšková
Vedúci diplomovej práce: RNDr. Juraj Šebej, PhD
Pracovisko: ÚINF

Úvod

V oblasti teórie formálnych jazykov a teórie automatov zohráva skúmanie stavovej zložitosti dôležitú úlohu pri porozumení výpočtovej sily a efektívnosti rôznych výpočtových modelov. Alternujúce automaty, variant nedeterministických konečných automatov (NFA), sú silnými nástrojmi na modelovanie a analýzu zložitých rozhodovacích procesov v počítačových systémoch. Ich schopnosť vyjadriť existenčné aj univerzálne vetvenie má za následok rozsiahlu využiteľnosť v praxi. Jedným príkladom využitia týchto automatov je verfikácia konkurentných a distribuovaných systémov. Častým učebnicovým príkladom využitia alternujúcich automatov je určenie optimálnej stratégie pre jedného hráča v hre s dvoma hráčmi.

V tejto práci nebudeme priamo študovať stavovú zložitosť operácii pre alternujúce automaty. Namiesto toho sa zameriame na deterministické konečné automaty s polovicou stavov koncových. Na určenie stavovej zložitosti operácii pre alternujúce automaty nám bude slúžiť nasledujúce tvrdenie.

Veta (Fellah, Jürgensen, Yu 1990)
Nech L je jazyk rozpoznateľný n stavovým alternujúcim konečným automatom (AFA), potom jazyk L^R je rozpoznateľný 2^n stavovým deterministickým konečným automatom (DFA) s polovicou stavov koncových.

V práci budeme skúmať špeciálny prípad, kedy automaty pracujú iba nad jednoprvkovou, unárnou, abecedou. V minulosti bolo viackrát preukázané, že stavová zložitosť operácii nad unárnou abecedou pre rôzne typy automatov sa správa inak ako je tomu vo všeobecnom prípade pre viacprvkovú abecedu.

1 Základné pojmy a doterajšie poznatky

Pred návrhom riešenia uvedieme niekoľko základných pojmov potrebných k pochopeniu práce a zhrnieme doterajšie poznatky. Najprv definujeme už spomínaný alternujúci automat.

Definícia (Miyano and Hayashi ((1984)))
Alternujúci konečný automatom (skrátene AFA) nazveme $A = (Q_\exists, Q_\forall, \Sigma, \delta, q_0, F)$, *označme* $Q := Q_\exists \cup Q_\forall$, *pričom:*

- $Q_\exists$ *je konečná množina existenciálnych stavov,*
- $Q_\forall$ *je konečná množina všeobecných stavou,*
- Σ *je konečná neprázdna množina vstupných symbolov (abeceda)*
- $\delta : Q \times \Sigma \to \mathscr{P}(Q)$ *prechodová funkcia*
- $q_0 \in Q$ *počiatočný stav*
- $F \subseteq Q$ *množina koncových/prijímajúcich stavov (final states)*

Poznamenajme, že slovo $w = w_1 w_2 ... w_n \in \Sigma^*$ je akceptované (prijímané) konečným alternujúcim automatom A, pokiaľ existuje postupnosť stavov $(q_0, q_1, ..., q_n)$ taká, že $\delta(q_{i-1}, w_i) = q_i$ pre $i \in \{1, ..., n\}$ a zároveň pokiaľ $q_i \in Q_\forall$, tak každá z ciest z q_0 do q_i musí reprezentovať práve podslovo $w_1 ... w_i$. Pre existenciálny stav stačí, že existuje jedna takáto cesta. Teda nedeterministické konečné automaty sú špeciálnym prípadom alternujúcich automatov, ktorých stavy sú iba existenciálne. Je dokázané, obdobne ako v prípade ekvivalencie medzi DFA a NFA, že aj jazyky prijímané alternujúcimi automatmi sú ekvivalentne jazykom prijímanými deterministickými automatmi.

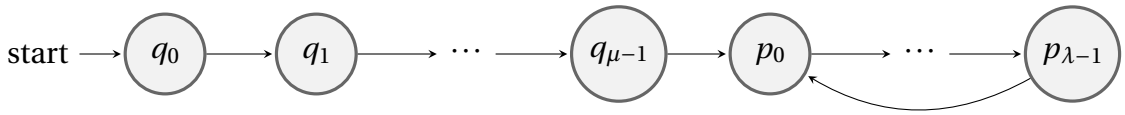
Veta
Nech A *je* n*-stavový alternujúci konečný automat alebo boolovský automat. Potom existuje* 2^{2^n} *stavový deterministický konečný automat, ktorý akceptuje jazyk* $L(A)$.

Druhým základným pojmom, ktorý je potrebné definovať, je stavová zložitosť operácií. Špeciálne definujme tento pojem pre binárne operácie, keďže tie sú cielom tejto práce. Analogicky by znela definícia pre ľubovoľnú r-árnu operáciu, $r \in \mathbb{N}$.

Definícia
Nech $\square$ *je binárna operácia nad triedou automatov* X. *Stavovou zložitosťou operácie* $\square$ *je funkcia* $sc_\square(m,n)$ *taká, že:*

- *pre každý* m*-stavový* X*-automat* A *a* n*-stavový* X*-automat* B *existuje* $sc_\square(m,n)$ *- stavový* X*-automat pre* $L(A) \circ L(B)$,
- *pre každé* $m \in \mathbb{N}$, $n \in \mathbb{N}$ *existuje* m*-stavový konečný* A *a* n*-stavový* X *- automat* B *taký, že každý* X*-automat pre* $L(A) \circ L(B)$ *má minimálne* $sc_\square(m,n)$ *stavov.*

Vo všeobecnosti stavová zložitosť operácii pre operácie nad unárnymi deterministickými konečnými automatmi je dobre preskúmaná. Je zrejmé, že graf ľubovoľného unárneho DFA má nasledujúci tvar až na množinu koncových stavov.



Teda každý graf unárneho automatu možno disjunktne rozdeliť na cestu dĺžky $\mu \in \mathbb{N}$ a kružnicu dĺžky $\lambda \in \mathbb{N}$. Zároveň μ a λ určujú unárny automat jednoznačne až na množinu koncových stavov. Koncové stavy možno reprezentovať ako binárny reťazec dĺžky $\mu + \lambda$, kde jednička na i-tom mieste značí, že i-tý stav je koncový.

Nasledujúce tvrdenia a ich dôkazy sú uvedené v práci od Pighizzini and Shallit ((2002))

Veta (Stavová zložitosť prieniku a zjednotenia unárnych DFA)
Nech graf unárneho DFA A pozostáva z cesty dĺžky μ_A a kružnice dĺžky λ_A a nech DFA B pozostáva z cesty dĺžky μ_B a kružnice dĺžky λ_B. Potom prienik a tiež zjednotenie sú pre jazyky $L(A)$ a $L(B)$ sú akceptované DFA dĺžky s kružnicou dĺžky $nsn(\lambda_A,\lambda_B)$ a cestou dĺžky $\max\{\mu_A,\mu_B\}$.

Uvedená veta hovorí však o stavovej zložitosti pre automaty s fixnou dĺžkou cesty a kružnice. Naším cieľom je určiť stavovú zložitosť pre všeobecný počet stavov pôvodných automatov. V tom nám napomôžeme nasledujúca lemma.

Lemma
Funkcia $f(\lambda_A,\lambda_B) = \max\{n-\lambda_A, m-\lambda_B\} + nsn(\lambda_A,\lambda_B)$ nadobúda pre pevne zvolené $m,n \in \mathbb{N}$ svoje maximum na $\{1,...,n\} \times \{1,...,m\}$ práve vtedy, keď

$$NSD(\lambda_A,\lambda_B) = 1$$

Zrejme teda pre stavovú zložitosť prienika a zjednotenia platí:

$$sc_\cap(m,n) = \max_{\lambda_A \in \{1,...,n\}, \lambda_B \in \{1,...,m\}} (\max\{n-\lambda_A, m-\lambda_B\} + nsn(\lambda_A,\lambda_B))$$

Pokiaľ sa obmedzíme na 2^m-stavový DFA A a 2^n-stavový DFA B, kde $m,n \in \mathbb{N}$, $m \geq n$, a má platiť, že $NSD(\lambda_A,\lambda_B) = 1$, tak maximum dosiahneme pre $\lambda_A = 2^m - 1, \lambda_B = 2^n$. Celkovo

$$sc_\cap(2^m,2^n) = 1 + 2^n(2^m-1) \qquad m \geq n$$
$$sc_\cup(2^m,2^n) = 1 + 2^n(2^m-1) \qquad m \geq n$$

To, že stavová zložitosť prieniku sa rovná stavovej zložitosti zjednotenia, vyplýva z následujúceho pozorovania pre DFA A a DFA B.

$$L(A) \cup L(B) = (L(A)^C \cap L(B)^C)^C$$

Avšak mi sa budeme zaoberať triedou unárnych DFA s polovicou stavou koncových, teda stavová zložitosť operácii na týchto automatoch môže byť nižšia.

Rovnako poznáme stavovú zložitosť pre konkatenáciu nad unárnymi deterministickými konečnými automatmi. Tvrdenia boli formulované pre nasledujúce špeciálne prípady.

Veta
Nech graf unárneho DFA A pozostáva z cesty dĺžky μ_A a kružnice dĺžky λ_A a nech DFA B pozostáva z cesty dĺžky μ_B a kružnice dĺžky λ_B, pričom $\mu_A,\mu_B \geq 2,\lambda_A,\lambda_B \geq 2$. Potom jazyk $L(A)L(B)$ je prijímaný automatov s kružnicou dĺžky $nsn(\lambda_A,\lambda_B)$ a cestou dĺžky $\mu_A + \mu_B + nsn(\lambda_A,\lambda_B) - 1$.

Veta
Nech graf unárneho DFA A pozostáva z cesty dĺžky μ_A a kružnice dĺžky λ_A a nech DFA B pozostáva z cesty dĺžky μ_B a kružnice dĺžky λ_B, pričom $NSD(\lambda_A,\lambda_B) = 1$. Potom jazyk $L(A)L(B)$ je prijímaný automatov s cestou dĺžky 1 a kružnicou dĺžky $\lambda_A\lambda_B + \mu_A + \mu_B - 1$.

Všimneme si, že napriek tomu, že konkatenácia automatov je nesymetrická operácia, t.j. vo všeobecnosti $AB \neq BA$, stavová zložitosť konkatenácie je symetrická.

Obdobne ako v predchádzajúcom prípade, maximum dosiahneme ak $NSD(\lambda_A,\lambda_B) = 1$. Špeciálne pokiaľ počet stavov DFA A a DFA B sú mocniny dvojky, platí:

$$sc.(2^m,2^n) = 1 + 2^n(2^m - 1) \qquad m \geq n$$

Čo je opäť rovnaký výsledok ako pri prieniku a zjednotení. Avšak obmedzením sa na triedu DFA s polovicou stavov koncových môže byť tento výsledok značne menší.

Čo sa týka stavovej zložitosti konkatenácie nad AFA pre ľubovoľnú abecedu Fellah et al. ((1990)) publikovali nasledujúce tvrdenie.

Veta (Stavová zložitosť konkatenácie pre AFA)
Nech A je m-stavový AFA a B je n-stavový AFA. Potom jazyk $L(A)L(B)$ je prijímaný alternujúcim automatom s $2^m + n + 1$ stavmi.

2 Impremetácia

Základným krokom pre skúmanie stavovej zložitosti v tejto práci bolo zostavenie programu, ktorého výstupom by boli jednotlivé operácie prieniku, zjednotenia a konkatenácie pre ľubovoľné dva deterministické konečné automaty nad unárnou abecedou.

Prv, než jednotlivé funkcie implementujeme, je potrebné zvoliť vhodnú reprezentáciu pre unárne automaty. Ako už bolo uvedené vyššie, unárny automat je určený jednoznačne svojou „cestou“, „kružnicou“ a množinou koncových stavov.

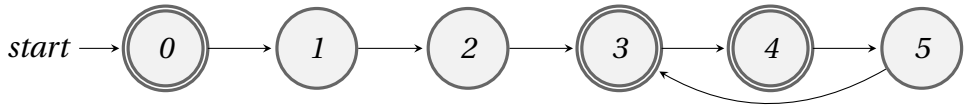
Reprezentuje teda unárny automat ako trojicu (premenná *tuple*) **(n,k,F)**, kde:

- **n** značí počet stavov automatu
- **k** index stavu, do ktorého vedú práve dva prechody
- **F** boolovská postupnosť (premenná *list*) dĺžky n, kde *True* na i-tom mieste značí, že i-tý stav je prijímajúci, v opačnom prípade je na danom mieste *False*

Teda dĺžka kružnice bude $n-k+1$ a dĺžka cesty bude $k-1$.

Pre lepšiu ilustráciu uveďme príklad.

Príklad
Nech je daný nasledujúci unárny DFA zobrazený nižšie.



Potom jedno reprezentáciou v počítači bude:

(5,3,[True,False,False,True,True,False])

Druhým podstatným problémom, ktoré je potrebné vyriešiť, je spôsob, akým budeme generovať všetky možné dvojice automatov, prvý s počtom stavov m, druhý s počtom stavov n, pre nejaké pevne zvolené $m,n \in \mathbb{N}$ párne.

Zrejme stačí prechádzať vnoreným cyklom cez všetky možnosti.

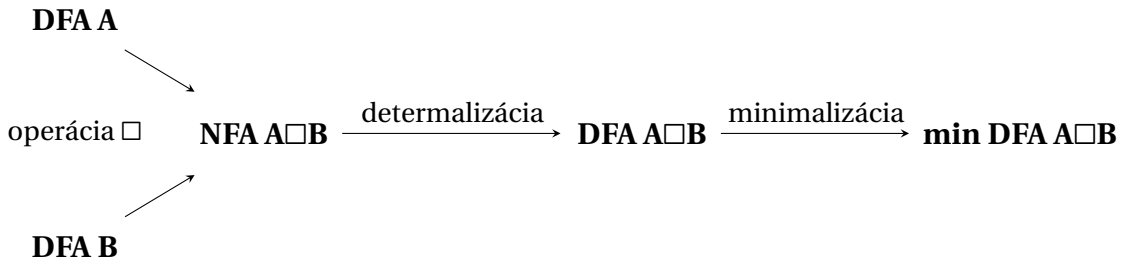
```
for k_A = 0,...,m−1 do
    for k_B = 0,...,n−1 do
        for f_A ∈ F_m do
            for f_B ∈ F_n do
                Operation((n,k_A,f_A),(m,k_B,f_B))
            end
        end
    end
end
```

V algoritme uvádzame množiny F_m a F_n všetkých možných postupností F pre koncové stavy dĺžky m a n. Tie generujeme rekurzívne pomocou nasledujúceho algoritmu.

bool_list (n,bools,ones,zeros,result)

```
if ones + zeros = n then
    vlož bools do zoznamu result
end
if ones < n/2 then
    bool_list(n, bools + [True], ones + 1, zeros, result)
end
if ones < n/2 then
    bool_list(n, bools + [False], ones, zeros + 1, result)
end
return result
```

Po vyriešení problému generovania všetkých možných automatov môžeme prejsť na implementáciu operácii. Pamätajme ale na to, že po aplikovaní danej operácie na dva deterministické automaty nám vo všeobecnosti vznikne nedeterministický automat. Na ten môžeme následne aplikovať známy algoritmus determinizácie. Výsledný automat však ešte nemusí byť minimálny, je potrebné ho ešte zredukovať. Uvedenú úvahu zhrnieme následujúcim grafom nižšie.



Rozoberme najprv prípad operácie prieniku dvoch DFA. Poznamenajme, že aj v prípade DFA s polovicou stavov koncových platí

$$L(A) \cup L(B) = (L(A)^C \cap L(B)^C)^C$$

Uvedené pozorovanie zdôvodníme. Nech A,B sú dva konečné deterministické automaty s polovicou stavov koncových, zrejme $L(A)^C$ aj $L(B)^C$ sú jazyky prijímané deterministickými automatmi s polovicou stavov koncových, keďže doplnok k DFA je automat, ktorý má množinu koncových stavov práve opačnú, t.j. stavy, ktoré boli koncové sú u doplnku nekoncové a naopak. Obdobne aj doplnok prieniku dvoch DFA s polovicou stavov koncových je výsledkom prieniku nejaký dvoch DFA s polovicou stavov koncových.

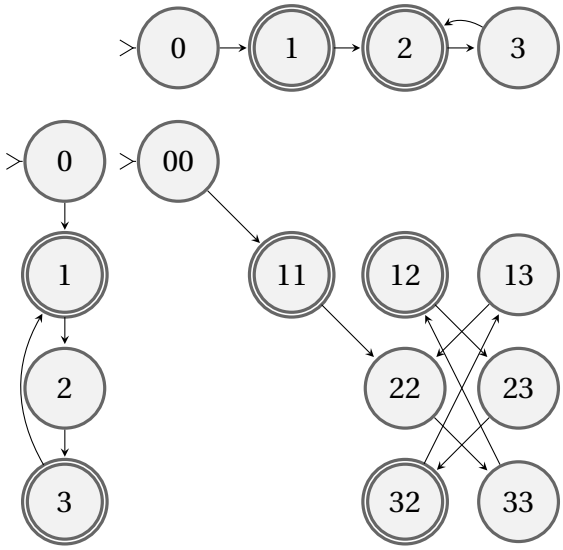
Vďaka tomuto pozorovaniu platí, že stačí implementovať len funkciu pre prienik, výsledok stavovej zložitosti pre zjednotenie bude zhodný s prípadom pre prienik.

Nasledujúci algoritmus pre prienik je implementovaný v Pythone. Počíta štandartným spôsobom kartézsky súčin automatov A a B s počtom stavov po poradí m a n, pričom výsledok uchováva ako maticu typu $m \times n$ a priebežne počíta zoznam koncových stavov. Prvky matice sú buď 0, pokiaľ na danej pozícii sa nenachádza stav patriaci do kartézskeho súčinu, alebo prirodzené čísla od 1 po $k \in \mathbb{N}$, kde k je počet stavov vo výslednom prieniku. Algoritmus je založený na postupnom dopĺňaní čísel $l \in \{1,...,k\}$ na pozície (i,j) v matici, ktoré značia poradie stavu vo výslednom prieniku. Algoritmus končí, pokiaľ by ďalším stavov mal byť už existujúci stav v kartézskom súčine.

```
def intersection(A, B):
    # INPUT: unary DFA A,B both represented as a tuple (number_of_states, loop_state, bool_list_of_finals)
    # OUTPUT: unary DFA AnB represented as a tuple (number_of_states, loop_state, bool_list_of_finals)
    state_label = 0
    cartesian_product_matrix = np.zeros((A[0], B[0]))
    intersection_finals = []
    diag_range = min(A[0], B[0]) - 1
    for i in range(diag_range + 1):
        state_label += 1
        cartesian_product_matrix[i, i] = state_label
        intersection_finals.append(A[2][i] and B[2][i])
    i, j = diag_range, diag_range
    while True:
        if i < (A[0] - 1):
            i += 1
        else:
            i = A[1]
        if j < (B[0] - 1):
            j += 1
        else:
            j = B[1]
        if cartesian_product_matrix[i, j] > 0:
            break
        else:
            state_label += 1
            cartesian_product_matrix[i, j] = state_label
            intersection_finals.append(A[2][i] and B[2][j])
    return (int(state_label), int(cartesian_product_matrix[i, j] - 1), intersection_finals)
```

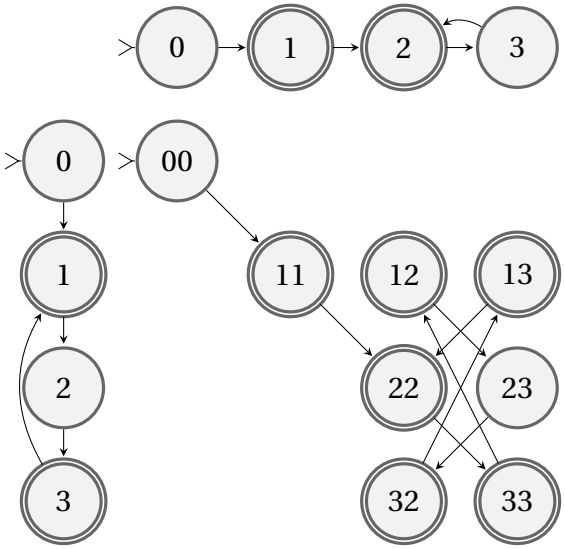
Obr. 1: Implementácia algoritmu prieniku v Pythone

Vzhľadom k uvedenej implementácii nie je potrebné determinizovať výsledný automat.



Príklad prieniku pre DFA s polovicou stavov koncových

Analogicky by fungoval algoritmus pre zjednotenie dvoch DFA. Jediným rozdielom by bolo výtvaranie koncových stavov, kedy miesto operácie *and* by sme využili operáciu *or*.



Príklad zjednotenia pre DFA s polovicou stavov koncových

Znenie vety z kapitoly 1 pre stavovú zložitosť prieniku a zjednotenia unárnych DFA nám umožňuje spočítať prienik rýchlejšie ako v predchádzajúcom algoritme. Priamo z vety dostávame n, počet stavov automatu pre prienik, a k, stav do ktorého vedú dva prechody. Ostáva určiť postupnosť koncových stavov F. Tú spočítame ako *bitový and* reťazcov s_1 a s_2, kde každý z reťazcov predstavuje zreťazenie postupnosti koncových stavov pre cestu a s-násobného zreťazenia postupnosti koncových stavov pre kružnice, kde s je najmenší spoločný násobok dĺžok kružníc.

intersection_fast $((n_A,k_A,F_A),(n_B,k_B,F_B))$

$s \leftarrow nsn(n_A - k_A, n_B - k_B)$
$n \leftarrow s + max(n_A, n_B)$
$s_1 = F_A[: k_A] + n * F_A[k_A :]$
$s_2 = F_B[: k_B] + n * F_B[k_B :]$
$s_1 = s_1[: n]$
$s_2 = s_2[: n]$
return $(n, max(k_A, k_B), s_1 \ \& \ s_2)$

Obdobnú myšlienku by bolo možné využiť aj pri konkatenácii, avšak problémom je v tomto prípade nejasnosť pri vytváraní postupnosti koncových stavov. Konkatenáciu vytvárame tak, že z koncových stavov prvého automatu vedieme prechod do druhého stavu druhého automatu. Výsledok nakoniec zdeterminizujeme. Uvedieme algoritmus, ktorými súčasne počíta konkatenáciu a jej determinizáciu. Stavy nového automatu po konkatenáciu budú reprezentované pomocou premennej *množina*, čo zároveň uľahčuje overenie, či novo vzniknutý stav bol už v minulosti vytvorený.

```
def concatenation(A,B):
    lst_of_states = []
    concatenation_finals = []
    if A[2][0]: #the first state in automata A is final
        new_state = set()
        new_state.add(0)
        lst_of_states.append(new_state)
        concatenation_finals.append(True)
    else:
        i = 0
        while not A[2][i]:
            new_state = set()
            new_state.add(i)
            lst_of_states.append(new_state)
            concatenation_finals.append(False)
            i+=1
    while True:
        new_state = set()
        is_final = False
        for x in lst_of_states[len(lst_of_states)-1]:
            if x//A[0] == 0:
                if x<(A[0]-1):
                    new_state.add(x+1)
                    if A[2][x+1]:
                        is_final = True
                else:
                    new_state.add(A[1])
                    if A[2][A[1]]:
                        is_final = True
                if A[2][x]:
                    new_state.add(A[0]+1)
                    if B[2][1]:
                        is_final = True
            else:
                if x<(A[0]+B[0]-1):
                    new_state.add(x+1)
                    if B[2][x+1-A[0]]:
                        is_final = True
                else:
                    new_state.add(A[0]+B[1])
                    if B[2][B[1]]:
                        is_final = True
        if new_state in lst_of_states:
            break
        else:
            lst_of_states.append(new_state)
            concatenation_finals.append(is_final)
    return (len(lst_of_states),lst_of_states.index(new_state),concatenation_finals)
```

Obr. 2: Implementácia algoritmu konkatenácie v Pythone

Typicky sa pri procese minimalizácie používa Hopcroftov algoritmus pracujúci v čase $O(n \log n)$. Uvedieme si jeho pseudokód.

Hopcroftov algoritmus

```
P ← {F, Q \ F}
W ← {F, Q \ F}
while W ≠ ∅ do
    zvol a odstráň A ∈ W
    for c ∈ Σ do
        X ← {q ∈ Q | δ(q,c) = p, p ∈ A}
        for Y ∈ P do
            if X ∩ Y ≠ ∅ & Y \ X ≠ ∅ then
                odstráň Y ∈ P
                vlož X ∩ Y a Y \ X do P
                if Y ∈ W then
                    odstráň Y ∈ P
                    vlož X ∩ Y a Y \ X do P
                end
                else
                    if |X ∩ Y| ≤ |Y \ X| then
                        vlož X ∩ Y do W
                    end
                    else
                        vlož Y \ X do W
                    end
                end
            end
        end
    end
end
```

Hopcroftov algoritmus sa používa pre DFA nad ľubovoľnou abecedou. Avšak vzhľadom k tvaru DFA (viď graf DFA v kapitole 1) nad unárnou abecedou sa problém minimalizácie automatu značne zjednoduchší. Pighizzini and Shallit ((2002)) uviedli nasledujúce pozorovanie.

Veta (Nutná a postačujúca podmienka minimalizácie unárneho DFA)
Nech graf unárneho DFA A pozostáva z cesty dĺžky λ a kružnice dĺžky μ. Nech $p_0,...,p_{\lambda-1}$ sú po poradí stavy na kružnici a $q_0,...,q_{\mu-1}$ sú po poradí stavy na ceste. Potom A je minimálny práve vtedy, keď:

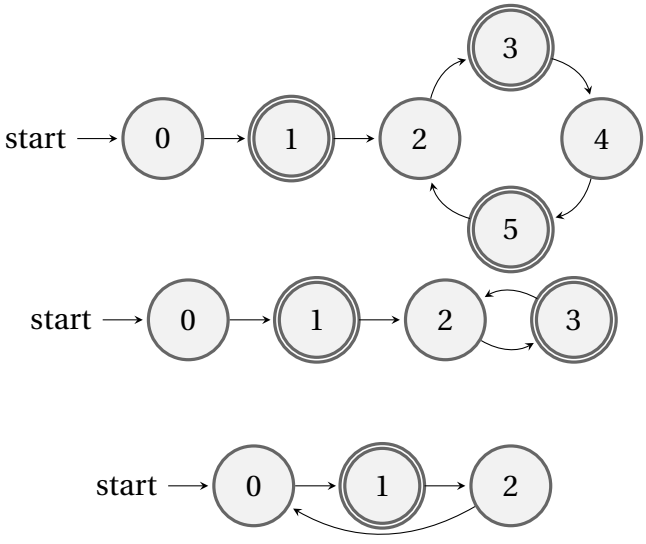
- *pre ľubovoľný maximálny vlastný deliteľ d čísla μ existuje $h \in \mathbb{N}$, $0 < h < X$, taký, že $p_h \in F$ práve vtedy, keď $p_{(h+d) mod \lambda} \notin F$*
- *$p_{\lambda-1} \in F$ práve vtedy, keď $q_{\mu-1} \notin F$*

Vzhľadom k uvedenému tvrdeniu nie je náročné zostaviť algoritmus pre minimalizáciu unárneho DFA. Stačí overiť, že reťazec koncových stavov prislúchajúcich kružnici nie je periodická, t.j. že ju nemožno rozdeliť na niekoľko rovnako dlhých zhodných disjunktných podporeťazcov. Pokiaľ áno, namiesto kružnice budeme mať len daný podreťazec. Na druhej strane overíme, či $k-1$-vý stav a posledný stav sú oba koncové alebo nekoncové. Pokiaľ áno, zahodíme posledný stav a prechodová funkcia bude viesť z predposledného stavu do $k-1$-vého stavu.

Minimalizácia unárneho DFA

```
cesta ← zoznam prvých k − 1 znakov z F
kružnica ← zvyšné znaky z F
for d ∈ VlastneDelitele(length(kruznica)) do
    rozdeľ kružnicu na disjunktné dieliky dĺžky d
    if diely sú rovnaké then
        kruznica ← jeden dielik
    end
end
while cesta[−1] = kruznica[−1] do
    kruznica ← cesta[-1] + kruznica[:-1]
    cesta ← cesta[:-1]
end
return (length(cesta)+length(kruznica), length(cesta), cesta+kruznica)
```

Pre lepšiu predstavivosť priebehu minimalizácie ilustrujme nasledujúci príklad.



Postupná minimalizácia unárneho DFA

3 Výsledky

Pomocou algoritmom uvedených v kapitole 2 sa nám podarilo vygenerovať všetky možné dvojice unárnych automatov s počtom stavov nanajvýš 10 a určiť minimálny automat pre ich prienik, zjednotenie a konkatenáciu. Následne stavovú zložitosť pre dané m a n a danú operáciu sme určili maximálne počet stavov pre výsledné minimálne automaty.

Výsledky sú uvedené v nasledujúcich tabuľkách.

m \ n	2	4	6	8	10
2	3	5	11	15	19
4	5	13	21	29	37
6	11	21	31	43	46
8	15	29	43	57	73
10	19	37	46	73	91

Tabuľka 1: Stavová zložitosť prieniku a zjednotenia

m \ n	2	4	6	8	10
2	3	5	7	9	11
4	5	8	12	12	18
6	7	10	15	20	21
8	9	12	18	24	30
10	11	14	21	29	35

Tabuľka 2: Stavová zložitosť konkatenácie

Z tabuľky 1 si všimneme, že stavové zložitosti práve odpovedajú hodnotám funkcie

$$sc_{\cap}(m,n) = \max_{\lambda_A \in \{1,\ldots,n\}, \lambda_B \in \{1,\ldots,m\}} (\max\{n-\lambda_A, m-\lambda_B\} + nsn(\lambda_A,\lambda_B))$$

ako je uvedené v kapitole 1. To znamená, že pre $(m,n) \notin \{(6,10),(10,6)\}$ existujú automaty $A = (m,1,F_A)$ a $B = (n,0,F_B)$, ktoré sa po operácii prieniku už nezminimalizujú. Pokiaľ $(m,n) \in \{(6,10),(10,6)\}$, tak sú to automaty typu $A = (m,1,F_A)$ a $B = (n,1,F_B)$.

Uveďme niekoľko príkladov pre automaty s maximálnou stavovou zložitosťou.

A			B			A∩B			reduct(A∩B)			
n	k	F	n	k	F	n	k	F	n	k	F
2	0	10	2	1	01	3	1	001	3	1	001
2	0	10	4	1	1100	7	1	1000100	7	1	1000100
2	0	10	6	1	111000	11	1	10100010000	11	1	10100010000
2	0	10	8	1	11110000	15	1	101000001010000	15	1	101000001010000
2	0	10	10	1	1111100000	19	1	1010100000101000000	19	1	1010100000101000000
4	0	1100	4	1	1100	13	1	1100100000000	13	1	1100100000000
4	0	1100	6	1	111000	21	1	11000000000100011000	21	1	11000000000100011000
4	0	1100	8	1	11110000	29	1	11000000110000001100000010000	29	1	11000000110000001100000010000
4	0	1100	10	1	1111100000	37	1	11001000000011000000110000001100000000	37	1	11001000000011000000110000001100000000
6	0	111000	6	1	111000	31	1	11100011000010000000000000010000	31	1	11100011000010000000000000010000
6	0	111000	8	1	11110000	43	1	11100000100000000000000010000011000011110000	43	1	11100000100000000000000010000011000011110000
6	1	111000	10	1	1111100000	46	1	11100000000110000000011000000001000001000000000	46	1	11100000000110000000011000000001000001000000000
8	0	11110000	8	1	11110000	57	1	11110000111000001100000010000000000000000000100...	57	1	11110000111000001100000010000000000000000000100...
8	0	11110000	10	1	1111100000	73	1	11110000001100000001000000000000000000000100000...	73	1	11110000001100000001000000000000000000000100000...
10	0	1111100000	10	1	1111100000	91	1	11111000001111000000111000000011000000001000000...	91	1	11111000001111000000111000000011000000001000000...

DFA s maximálnou stavovou zložitosťou pre prienik

		A			B			AB			reduct(AB)			how much it reduces
		n	k	F	n	k	F	n	k	F	n	k	F	
(A, n)	(B, n)													
4	**2**	4	3	1010	2	0	01	5	3		11110	5	3	11110	0
	4	4	1	1010	4	2	1001	10	7		1011010111	8	7	10110101	2
	6	4	0	1001	6	3	110001	15	11		110111011101111	12	11	110111011101	3
	8	4	1	1010	8	6	11010001	14	11		11110111110111	12	11	111101111101	2
	10	4	1	1010	10	6	1101100001	20	17		11111110110111110111	18	17	111111101101111101	2
6	**2**	6	5	101010	2	0	01	7	5		1111110	7	5	1111110	0
	4	6	1	101010	4	2	1001	14	9		10111101011111	10	9	1011110101	4
	6	6	1	110010	6	3	110001	19	14		11101111111011111	15	14	111011111111101	4
	8	6	1	100011	8	4	11100001	24	19		11101111011110111101111	20	19	111011110111101111	4
	10	6	3	110001	10	6	1101100001	23	20		11111110111111110110111	21	20	111111101111111101101	2
8	**2**	8	7	10101010	2	0	01	9	7		111111110	9	7	111111110	0
	4	8	1	10101010	4	2	1001	18	11		101111101011111111	12	11	101111101101	6
	6	8	1	10010011	6	3	110001	24	17		11011111101111101111111	18	17	110111110111111101	6
	8	8	3	11101000	8	4	11100001	28	23		11111111111111111101111011111	24	23	111111111111111111011101	4
	10	8	2	11000110	10	5	1111000001	35	29		11111111111111110111110111101111111	30	29	111111111111111101111101111101	5
10	**2**	10	9	1010101010	2	0	01	11	9		11111111110	11	9	11111111110	0
	4	10	1	1001011010	4	1	1010	15	6		101101111011111	14	5	10110011101111	1
	6	10	2	1100100101	6	3	110001	28	20		11101111111111111111011111111	21	20	111011111111111111101	7
	8	10	2	1100010101	8	2	11000101	35	11		11100011111111111111111111101111111	29	5	11100011111111111111111111101	6
	10	10	3	1110001010	10	5	1111000001	41	34		11111111111101111111111111111111101111111	35	34	11111111111101111111111111111111101	6

DFA s maximálnou stavovou zložitosťou pre konkatenáciu

V tabuľke pre konkatenáciu uvádzame aj či a o koľko stavov sa daná konkatenácia zredukovala (zminimalizovala). Pokiaľ jeden z automatov má len 2 stavy, výsledná konkatenácia sa neredukuje, inak áno.

Pri analyzovaní stavovej zložitosti konkatenácie pre unárne DFA s polovicou stavov koncových si je potrebné uvedomiť, že okrem algoritmu minimalizácie môže stavovú zložitosť oproti všeobecne unárnym DFA ovplyvniť aj podmienka pre polovicu stavov koncových, keďže z koncových stavov prvého automatu vedieme prechod do druhého stavu druhého automatu.

4 Formulácia hypotézy

Ako môžeme nahliadnúť v tabuľke pre prienik na predchádzajúcej strane, stavová zložitosť pre unárne DFA s polovicou stavov koncových je rovnaká ako pre všeobecne unárne DFA. Na dokázanie, že uvedené pozorovanie platí aj pokiaľ jeden z automatov bude mať počet stavov viac ako 10, postačí nájsť svedkov A a B, pre ktoré budeme vedieť určiť presný tvar prieniku. Vzhľadom k tabuľke formulujeme nasledujúcu hypotézu.

Hypotéza
Nech je daný unárny DFA $A = (n,0,1^{\frac{n}{2}}0^{\frac{n}{2}})$ *a DFA* $B = (m,1,1^{\frac{m}{2}}0^{\frac{m}{2}})$, $m,n \in \mathbb{N}$ *sú mocniny dvojky,* $n \leq m$. *Potom minimálny automat akceptujúci jazyk* $L(A) \cap L(B)$ *bude mať* $mn - n + 1$ *stavov. Špeciálne tento automat bude pozostávať z cesty dĺžky* 1 *a kružnice dĺžky* $(m-1)n$.

Dôkaz. Načrtneme ideu dôkazu. Graf automatu A je kružnicou dĺžky n, graf automatu B možno disjunktne rozdeliť na cestu dĺžky 1 a kružnicu dĺžky $m-1$. Podľa kapitoly 1 potom bez ohľadu na charakter koncových stavov je stavová zložitosť prieniku

$$sc_{\cap}(m,n) = nsn(m-1,n) + max\{0,1\} = n(m-1)+1 = mn - n + 1$$

keďže $m-1$ a n sú nesúdelné. Ostáva ukázať, že vzniknutý DFA prieniku A a B sa nezredukuje. Na určenie postupnosti koncových stavov potrebujeme vzhľadom k myšlienke algoritmu *intersection_fast* spočítať bitový AND pre nasledujúce regulárne výrazy dĺžky $n(m-1)+1$:

$$(1^{\frac{n}{2}}0^{\frac{n}{2}})^{m-1}1$$
$$1(1^{\frac{m}{2}-1}0^{\frac{m}{2}})^{n}$$

Keďže cesta vo výslednom grafe bude dĺžky $max\{0,1\} = 1$, tak z výsledku odoberieme prvý bit. Ostáva ukázať, že výsledný výraz nebude periodický.

Platí $(1^{\frac{n}{2}}0^{\frac{n}{2}})^{m-1}1 = 1(1^{\frac{n}{2}-1}0^{\frac{n}{2}}1)^{m-1}$. Teda stačí overiť neperiodickosť bitového ANDu pre výrazy:

$$(1^{\frac{n}{2}-1}0^{\frac{n}{2}}1)^{m-1}$$
$$(1^{\frac{m}{2}-1}0^{\frac{m}{2}})^{n}$$

Záver

Nasledujúcimi cieľmi pre túto prácu je dokončenie dôkazu hypotézy uvedenej v kapitole 4, čím by mala byť otázka stavovej zložitosti pre prienik a zjednotenie unárnych konečných automatov s polovicou stavov koncových úplne vyriešená. Ostáva analýza operácie konkatenácie pre spomínanú triedu automatov. Analyzovaním výsledkov z pomocného programu sa pokúsime nájsť nejaké závislosti pre jednotlivé parametre. Na záver, pokiaľ uvedené metódy nebudú vracať nové výsledky, bude našou snahou optimalizovať implementovaný kód a vygenerovať automaty a operácie na nich pre viacstavové automaty.

Literatúra

A. Fellah, H. Jürgensen, and S.Yu. Constructions for alternating finite automata. *International Journal of Computer Mathematics*, 35(1-4):117–132, 1990. doi: 10.1080/00207169008803893.

S. Miyano and T. Hayashi. Alternating finite automata on -words. *Theoretical Computer Science*, 32(3):321–330, 1984. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(84)90049-5.

G. Pighizzini and J. Shallit. Unary language operations, state complexity and jacobsthal's function. *International Journal of Foundations of Computer Science*, 13(01):145–159, 2002.