VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

http://fit.vut.cz

Projektová praxe 2021/2022

Využití opakujících se podstruktur pro efektivní reprezentaci automatů

Bc. Michal Šedý

Abstrakt

Nedeterministické konečné automaty (NKA) jsou široce využívány napříč téměř všemi odvětvími počítačové vědy, například při reprezentaci regulárních vvýrazů, monitorování vysokorychlostních sítí, v abstraktním regulárním model chackingu, verifikaci programů manipulujícími s řetězci nebo v rozhodovacích procedůrách WS1S a WS2S logiky. NKA jsou dokonce využívány v bioinformatice pro vyhledávání sekvencí nukleotidových kyselin v DNA. Základní technikou snižující nároky na výpočetní zdroje (maměť, čas nebo množství hardwarových komponentů) při práci s NKA je minimalizace. Nejznámějšími minimalizačními metodami jsou slučování stavů a prořezávání hran přechodů. Přestože kombinací těchto dvou metod lze dosáhnou snížení velikosti automatů až o 50 %, tak ve výsledných automatech se mohou stále nacházet duplicitní sekvence přechodů. Dokonce existují automaty, které zmíněné postupy vůbec nedokáží minimalizovat. Tato práce prezentuje nový způsob minimalizace automatů, jehož základem je informovaný převod NKA na nedeterministický zásobníkový automat (NZA). Při převodu jsou vyhledávány nejpodobnější části automatů (procedury), které mohou být reprezentovány pouze jednou. Tento algoritmus nově umožňuje redukovat doposud neminimalizovatelné automaty a také vylepšit výsledky ostatních minimalizací. Princip převodu NKA na NZA lze připodobnit k převodu čistě sekvenčního programu na program s vzájemně komunikujícími procedurami.

Klíčová slova: nedeterministický konečný automat — nedeterministický zásobníkový automat — minimalizace — simulace

xsedym02@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

Nedeterministické konečné automaty (NKA) prezentovali Michael Rabin a Dana Scott v [12]. Oproti své deterministické variantě se vyznačují schopností přechodu do více následujících stavů na základě stejného přijatého znak. To umožňuje NKA kompaktněji reprezentovat daný jazyk. Na druhou stranu tato vlastnost působí minimalizaci NKA složitou. I přes obtížnou minimalizaci jsou NKA široce využívány napříč téměř všemi odvětvími počítačové vědy, například při reprezentaci regulárních výrazů, pro monitorování vysokorychlostních sítí [13], v abstraktním regulárním model checkingu [5], verifikaci programů manipulujícími s řetězci [2] nebo v rozhodovacích procedurách WS1S a WS2S logiky [9, 8]. NKA jsou dokonce využívány v bioinformatice pro vyhledávání sekvencí nukleotidových kyselin v DNA [3].

Základní technikou snižující nároky na výpočetní zdroje (paměť, čas nebo množství hardwarových komponentů) při práci s NKA je právě minimalizace. Nejznámější technikou minimalizace je slučování stavů [4, 6, 11], které vyhledává dva jazykově ekvivalentní stavy a ty následně sloučí v jeden. Dalším přístupem je prořezávaní hran přechodů [6, 7], které na základě jazykové inkluze dvou stavů odstraní přechod jazykově slabšího stavu, protože má jistotu, že funkce tohoto odstraněného přechodu je již zastoupena jazykově silnějším stavem. Opakem k prořezávání hran přechodů je jejich přidávání (saturace) [4, 7], kde právě nově přidané hrany mohou umožnit další slučování stavů nebo odstraňování jiných hran.

Přestože kombinace zmíněných minimalizačních metod redukuje u většiny automatů jejich velikost

až o 50 %, tak ve výsledných automatech stále existují duplicitní sekvence přechodů. Existují také typy automatů, které nejsou dosavadními postupy vůbec minimalizovatelné. Do této skupiny patří automaty s lineární strukturou (bez větvení) nebo automaty reprezentující slova s daným infixem a stejným prefixem i sufixem (např. *abba, cbbc*). V těchto případech nemůže slučování stavů, prořezávání či přidávání přechodů automat nijak minimalizovat. Důvodem je, že tyto redukční přístupy pracují na základě jazykových inkluzí a tyto typy automatů žádnou inkluzi neobsahují. Ve struktuře zmíněných automatů se vyskytují pouze úseky podobných přechodů (např. *bb* z předchozího příkladu).

Práce popisuje minimalizační algoritmus, který využívá podobné sekvence přechodů k redukci velikosti automatu. Prezentovaná metoda pracuje na základě informovaného převodu nedeterministického konečného automatu na nedeterministický zásobníkový automat (NZA). V NZA lze podobné úseky, nahradit jednou procedurou, která na základě symbolu uloženého na zásobníku rozpozná, ve které větvi původního automatu se výpočet nachází. Cílem úspěšného převodu je nahrazení takových sekvencí přechodu, že úspora způsobena jejich redukcí bude převyšovat nad režii zásobnikových operací. Při použití tohoto postupu minimalizace se lze, pro dostatečně velké automaty se slovy obsahujícími stejný prefix a sufix, přiblížit až 100% redukci velikosti.

Převod NKA na NZA lze připodobnit k převodu čistě sekvenčního programu na program využívající mezi sebou navzájem komunikující procedury. Podobně jako v NZA je i v programu využit zásobník volání pro uložení informace o větvi, ve které se program při volání nachází a kam se má výpočet vrátit po skončení procedury.

2. Základní pojmy

Tato sekce definuje základní teoretické pojmy využívané v minimalizačním algoritmu, jakými jsou nedeterministický konečný automat a nedeterministický zásobníkový automat. A dále zavádí upravenou definici relace simulace, která bere v potaz pouze přechody do vzdálenosti $k \in \mathbb{N}$ od zkoumaného stavu a nerozlišuje mezi koncovými a nekoncovými stavy.

2.1 Nedeterministický konečný automat

Nedeterministický konečný automat (NKA) je pětice $M = (Q, \Sigma, \delta, i, F)$, kde:

- Q je konečná množina stavů,
- Σ je abeceda,
- $\delta : Q \times \Sigma \to 2^Q$ je přechodová funkce,
- $i \in Q$ je počáteční stav a
- $F \subseteq Q$ je množina koncových stavů.

Přechodovou funkci δ lze také zobecnit pro množinu symbolů. Mějme $q \in Q$ a $A \subseteq \Sigma$, pak platí, že $\delta(q, A) = \bigcup_{a \in A} \delta(q, a)$.

K přechodové funkcí definujeme také inverzní funkci δ^{-1}, kde platí: $q \in \delta^{-1}(r, a) \iff r \in \delta(q, a)$, pro $a \in \Sigma$ a $q, r \in Q$.

Konfigurace

Konfigurace NKA je uspořádaná dvojice $C \in Q \times \Sigma^*$. $(q, w) \in C$ značí, že se řízení automatu nachází ve stavu q a na vstupu zbývá nezpracovaný řetězec w.

Přechod automatu

Přechod automatu je binární relace $\vdash \subseteq C \times C$, která je definována takto: $(q, w) \vdash (r, w') \iff w = aw' \wedge r \in \delta(q, a)$, pro $q, r \in Q$, $w, w' \in \Sigma^*$, $a \in \Sigma$.

Jazyk

Dopředný jazyk stavu $q \in Q$ je množina $\overrightarrow{L}(q) = \{w \in \Sigma^* \mid (q, w) \vdash (f, \varepsilon)$, kde $f \in F\}$.
Zpětný jazyk stavu $q \in Q$ je množina $\overleftarrow{L}(q) = \{w \in \Sigma^* \mid (i, w) \vdash (q, \varepsilon)\}$.
Jazyk automatu M je množina $L(M) = \overrightarrow{L}(i)$.

Abeceda stavu

Pro NKA dále zaveďme *abecedu stavu*, která je jednoznačně určena funkcí $\sigma : Q \to 2^\Sigma$, kde platí $a \in \sigma(q) \iff \exists r \in Q : r \in \delta(q, a)$, pro $a \in \Sigma$.

2.2 Nedeterministický zásobníkový automat

Nedeterministický zásobníkový automat (NZA) je sedmice $M = (Q, \Sigma_\varepsilon, \Gamma_\varepsilon, \delta, i, S, F)$, kde:

- Q je konečná množina stavů,
- Σ_ε je vstupní abeceda[1],
- Γ_ε je zásobníková abeceda,
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to 2^{Q \times \Gamma_\varepsilon}$ je přechodová funkce,
- $i \in Q$ je počáteční stav,
- $Z \in \Gamma$ je počáteční zásobníkový symbol a
- $F \subseteq Q$ je množina koncových stavů.

Podobně jako pro NKA lze definovat přechodovou funkci také pro množinu symbolů $A \subseteq \Sigma$, nebo $B \subseteq \Gamma_\varepsilon$.

Pro inverzní přechodovou funkci δ^{-1} v NZA platí: $(q, \beta) \in \delta^{-1}(r, a, \gamma) \iff (r, \gamma) \in \delta(q, a, \beta)$, kde $q, r \in Q$, $a \in \Sigma_\varepsilon$, $\beta, \gamma \in \Gamma_\varepsilon$. Povšimněte si, že u inverzní funkce se zamění vkládaný a vyjímaný zásobníkový symbol.

[1] $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$

Konfigurace

Konfigurace NZA je uspořádaná trojice $C \in Q \times \Sigma^* \times \Gamma^*$. $(q,w,\beta) \in C$ značí, že se řízení automatu nachází ve stavu q, na vstupu zbývá nezpracovaný řetězec w a zásobník obsahuje řetězec β (vrchol zásobníku je vlevo).

Přechod automatu

Přechod automatu je binární relace $\vdash \subseteq C \times C$, která je definována takto: $(q,w,\beta) \vdash (r,w',\beta') \iff w = aw' \wedge \beta = X\alpha \wedge \beta' = Y\alpha \wedge (r,Y) \in \delta(q,a,X)$, pro $q,r \in Q$, $w,w' \in \Sigma^*$, $a \in \Sigma_\varepsilon$, $X,Y \in \Gamma_\varepsilon$.

2.3 Simulace

Pro popis vztahu mezi stavy s podobnou sekvencí přechodů je použita modifikovaná *relace simulace* [10, 1]. Oproti původní relaci zkoumá modifikovaná relace simulace (*k-aproxsim*) pouze přechody do vzdálenosti $k \in \mathbb{N}$ od zkoumaného stavu. Dále *k-aproxsim* nerozlišuje mezi koncovými a nekoncovými stavy.

Simulace

Simulace na NKA M je kvaziuspořádání $\preceq \subseteq Q \times Q$. Nechť $a \in \Sigma$. Relace $p \preceq q$ existuje pouze tehdy, když platí $r \in F \implies q \in F$ a pro každé $r' \in \delta(r,a)$ existuje $q' \in \delta(q,a)$, pro které dále musí platit $r' \preceq q'$.

K-aproxsim

K-aproxsim na NKA M je kvaziuspořádání $\precsim_k \subseteq Q \times Q$, pro které platí: pokud je $k = 0$, pak $\precsim_0 = Q \times Q$, jinak, $r \precsim_k q \iff \forall r' \in \delta(r,a)\ \exists q' \in \delta(q,a) : r' \precsim_{k-1} q'$, kde $a \in \Sigma$.

3. Minimalizační techniky

Tato sekce uvádí v současnosti nejpoužívanější minimalizační metody, jakými jsou: slučování stavů, prořezávání přechodů a přidávání přechodů.

3.1 Slučování stavů

Dva stavy p a q mohou být sloučeny v jeden, pokud je splněna alespoň jedna z následujících podmínek:

- $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q) \wedge \overleftarrow{L}(q) \subseteq \overleftarrow{L}(p)$,
- $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overrightarrow{L}(q) \subseteq \overrightarrow{L}(p)$, nebo
- $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q) \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$.

3.2 Prořezávání přechodů

Přechod $pa \to q$[2] může být odstraněn, pokud je splněna jedna z následujících podmínek:

- $\exists ra \to q \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$
- $\exists qa \to p \wedge \overleftarrow{L}(r) \subseteq \overleftarrow{L}(q)$ nebo
- $\exists r'a \to p' \wedge \overleftarrow{L}(r) \subseteq \overleftarrow{L}(r') \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(p')$.

[2] Namísto $q \in \delta(p,a)$ můžeme psát $pa \to q$.

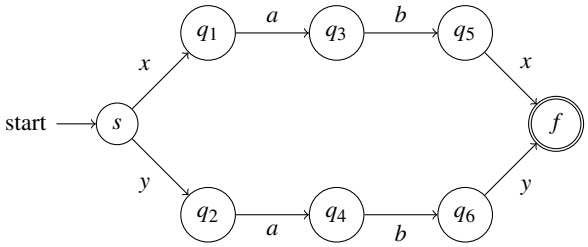
3.3 Přidávání přechodů

Základní myšlenka přidávání přechodů (nebo-li saturace) je analogií k prořezávání přechodů. Přechod $pa \to q$ může být do automatu přidán pokud je splněna alespoň jedna z následujících podmínek:

- $\exists qa \to r \wedge \overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$ nebo
- $\exists pa \to q \wedge \overrightarrow{L}(r) \subseteq \overrightarrow{L}(q)$.

3.4 Limitace

Lze vidět, že všechny minimalizační techniky jsou založeny na jazykových inkluzích (mohou být určeny na základě simulace). Pak v případě lineárních automatů (bez větvení), nebo automatů slov se stejným prefixem a sufixem nemohou být tyto metody plně využity, protože v těchto automatech existuje minimum stavů v netriviální jazykové inkluzi.



Obrázek 1. Příklad automatu přijímajícího slovo s infixem ab, a kde prefix i sufix musí být x nebo y.

Z obrázku 1 lze vidět, že v automatu nelze provést žádnou minimalizaci založenou na jazykových inkluzích (až na triviální se zde žádné nevyskytují). Přesto si lze všimnout, že se infixová část slova (zbytečně) opakuje. Hlavní motivací práce je fakt, že se v automatech tohoto a podobných typů vyskytují dlouhé opakující se sekvence přechodů, které by mohly být reprezentovány pouze jednou. V dostatečně velkých automatech, obsahujících dlouhé podobné sekvence přechodu, může tato náhrada za jednu proceduru přinést redukci velikosti přechodové funkce blížící se limitně až 100 %.

4. Převod NKA na NZA

Převod původního NKA na NZA s procedurami sestává z pěti částí. Nejdříve je spočítaná $asim_n$ pro $n = 1,\ldots,k$, kde $p \in asim_n(q) \iff q \precsim_n p$. ($asim_n$ je spočítaná také pro reverzní automat. Pokud se při iteraci redukčního algoritmu vybere prvek z reverzní $asim_n$, pak je redukce prováděná na reverzním automatu.) Pro původní automat je vytvořen odpovídající NZA. Pro všechny stavy z nejsilnější $asim_k(p)$ a jejich následníky do vzdálenosti k je vytvořena procedura, na kterou se původní stavy přemapují. Všechny přemapované stavy jsou následně odstraněny z automatu i z $asim_n$ pro $n = 1,\ldots,k$. Znovu je vybraná

nejsilnější $asim_k(p)$. Pokud je $asim_k = \emptyset$, pak se pokračuje s $asim_{k-1}$. Po provedení všech přemapování je zjednodušena zásobníková abeceda pomocí α-*redukce* a množství přechodů je sníženo ε-*redukcí*.

4.1 Výpočet $asim_k$

Relace $asim_k \subseteq Q \times 2^Q$ pro $k \in \mathbb{N}$ je definována následovně: $p \in asim_k(q) \iff q \precsim_k p$. Pro výpočet je využit modifikovaný algoritmus RefinedSimilarity [10]. Před uvedením algoritmu definujme množinu předchůdců stavů $S \subseteq Q$ jako $anc(S) = \bigcup_{q \in S} \delta^{-1}(q, \Sigma)$. Dále definujme pomocnou relaci $prevasim_k$, kde množina $prevasim(q) \supseteq asim_k(q)$ je množina předpokládaných kandidátů pro $asim_k(q)$.

Algoritmus 1: Výpočet $asim_k$

Input: NKA $M = (Q, \Sigma, \delta, i, F)$, $k \in \mathbb{N}$
Output: $asim_k(q)$

```
1  forall q ∈ Q do
2      asim_k^1(q) = {r ∈ Q | σ(q) ⊆ σ(r)}
       /* σ(q) je abeceda stavu q */
3  prevasim_k^1 = Q × {Q}
4  i = 1
5  while i ≤ k do
6      asim_k^i = asim_k^{i-1}
7      forall q ∈ Q do
8          forall a ∈ σ(q) do
9              s_by_a = {s ∈ Q | ∃r ∈ Q : r ∈ δ(q,a)}
10             acn_pasim = anc(prevasim_k^{i-1} ∩ s_by_a, a)
11             anc_asim = anc(asim_k^{i-1} ∩ s_by_a, a)
12             remove = anc_pasim \ anc_asim
13             forall s ∈ δ^{-1}(q,a) do
14                 asim_k^i(s) = asim_k^i(s) \ remove
15     prevasim_k^i = asim_k^{i-1}
16     i = i + 1
17 return asim_k^k
```

V první smyčce for algoritmus inicializuje relaci $asim_k^1$, která odpovídá $\precsim_1$. Oproti původnímu algoritmu počítajícího simulaci zde není rozlišováno mezi koncovými a nekoncovými stavy. Základní myšlenkou algoritmu je, že s každou další iterací cyklu while se vzdálenost na kterou působí relace $asim_k$ prodlužuje, dokud nedosáhne požadované vzdálenosti k.

4.2 Reprezentace NKA pomocí NZA

Před zahájením vyhledávání procedur je původní NKA přepsán do tvaru NZA bez jakékoliv změny vnitřní struktury.

Mějme dán NKA $M = (Q, \Sigma, \delta, i, F)$. Automat M převedeme na NZA $Z = (Q, \Sigma, \{\bot\}, \delta', \{\bot\}, i, F)$, kde $(p, \varepsilon) \in \delta'(q, a, \varepsilon) \iff p \in \delta(q, a)$, kde $p, q \in Q$, $a \in \Sigma$.

4.3 Vytvoření procedury

Mějme dán NZA $Z = (Q, \Sigma, \Gamma, \delta, i, S, F)$ a původní NKA $M = (Q_M, \Sigma_M, \delta_M, I_M, F_M)$. Definujme bijekci $cname : Q \to Q$, kde $cname(q) = r$ udává, že se stav q přemapoval na stav r. $cname$ je inicializován jako identita.

Procedury se vytváří postupně od nejsilnější dvojice $(p, S) \in asim_k$, kde $p \in Q$ se nazývá *primární stav* a $(S \setminus \{p\}) \subseteq Q$ množina *sekundárních stavů*. Nejprve je vytvořena *kostra procedury* na základě množiny $follow_k(p) \subseteq Q$, která obsahuje následníky primárního stavu p v automatu M do vzdálenosti k. Následně jsou pro každý sekundární stav $s \in (S \setminus \{q\})$ přemapovány všechny stavy z $follow_k(s)$ na tuto kostru.

Stavy do vzdálenosti k

Množina stavů do vzdálenosti $k \in \mathbb{N}$ od stavu $q \in Q_M$ je definována jako $follow_k(q) = \{s \in Q_M \mid (q, w) \vdash^n (s, w'),\ 0 \leq n \leq k,\ w, w' \in \Sigma_M^*\}$.

Síla relace $aim_k(q)$

Síla relace $pow(aim_k(q)) \in \mathbb{N}$ je definována následovně: $pow(aim_k(q)) = |\{(r, a, s) \mid s \in \delta_M(r, a),\ r, s \in follow_k(q),\ a \in \Sigma_M\}|$.

Algoritmus 2: Vytvoření kostry procedury

Input: NZA $Z = (Q, \Sigma, \Gamma, \delta, i, S, F)$, $p \in Q$, $follow_k(p)$, $cname$
Output: NZA $Z' = (Q', \Sigma, \Gamma', \delta', i, S, F)$, $cname'$

```
1  follow = follow_k(p) ∩ Q
2  Q' = Q
3  Γ' = Γ ∪ follow
4  δ' = δ
5  cname' = cname
6  forall q ∈ follow_k(p) do
7      t = newState()                          // t ∉ Q'
8      Q' = Q' ∪ {t}
9      cname'(q) = t
10 forall q ∈ follow do
11     inside = {(r,a) | (r,β) ∈ δ(q,a,Γ), r ∈ follow}
12     forall (r,a) ∈ inside do
13         δ' = δ' ∪ {(cname(q), a, q) → (cname(r), r)}
14     in = {(r,a,β) | (r,β) ∈ δ^{-1}(q,a,Γ), r ∈ Q \ follow}
15     forall (r,a,β) ∈ trans_in do
16         δ' = δ' ∪ {(r,a,β) → (cname(q), q)}
17     out = {(r,a,β) | (r,β) ∈ δ(q,a,Γ), r ∈ Q \ follow}
18     forall (p,a,β) ∈ out do
19         δ' = δ' ∪ {(cname(q), a, q) → (r, β)}
20 return Z', cname'
```

Pro vytvoření kostry procedury stavu $p \in Q$ do vzdálenosti $k \in \mathbb{N}$ je zapotřebí pro každý stav $r \in follow_k(p)$ vygenerovat nový, na který se stav r přemapuje. Odpovídající dvojice stavů obsahuje zobrazení $cname$. Přechody vstupující do stavu $cname(r)$ vkládají na zásobník znak r a přechody vycházející ze stavu $cname(r)$ vyjímají ze zásobníků symbol r. Lze vidět,

že při přechodu ze stavu $r_1 \in follow_k(p)$ do stavu $r_2 \in follow_k(p)$ bude znak r_1 vyjmut ze zásobníku a r_2 na zásobník vložen.

Algoritmus 3: Mapování sekundárních stavů

Input: NZA $Z = (Q,\Sigma,\Gamma,\delta,i,S,F)$, $follow_k(s)$, $k \in \mathbb{N}$, *cname*
Output: NZA $Z' = (Q,\Sigma,\Gamma',\delta',i,S,F)$, *cname*$'$

```
1  follow = follow_k(s) ∩ Q
2  Γ' = Γ
3  δ' = δ
4  cname' = cname
5  mapped = ∅
6  queue = {(p,s,k,False)}                    // fronta
7  while queue ≠ ∅ do
8      (p,s,i,visible) = queue.pop()
9      if s ∈ mapped then
10         continue
11     if s = cname(s) then      // s nikdo nemapoval
12         visible = True
13         cname'(s) = cname'(p)
14         Γ' = Γ' ∪ {s}
15     mapped = mapped ∪ {s}
16     if visible ∧ i > 0 then
           /* s nikdo nemapoval. Existuje
              následník v proceduře.          */
17         inside = {(r,a,β) | (r,β) ∈ δ(s,a,Γ), r ∈ follow}
18         forall (q,a,β) ∈ inside do
19             if r = cname(r) then
20                 δ' = δ' ∪ {(cname(s),a,s) →
                     (cname(r),r)}
21             else
22                 δ' = δ' ∪ {(cname(s),a,s) →
                     (cname(r),β)}
23         in = {(r,a,β) | (r,β) ∈ δ^-1(s,a,γ), r ∈ Q \ follow}
24         forall (r,a,β) ∈ in do
25             δ' = δ' ∪ {(r,a,β) → (cname(s),s)}
26         out = {(r,a,β) | (r,β) ∈ δ(s,a,γ), r ∈ Q \ follow}
27         forall (r,a,β) ∈ out do
28             δ' = δ' ∪ {(cname(s),a,s) → (r,β)}
29     if i > 0 then
           /* Zařaď do fronty dvojice
              ekvivalentních následníků.      */
30         next = {(r,a) | (r,β) ∈ δ(s,a,γ), r ∈ follow}
31         forall (r,a) ∈ next do
32             if visible ∧ r ≠ cname(r) then
33                 continue
34             if δ'(p,a,Γ) ∩ asim_{i-1}(q) = ∅ then
35                 continue
36             next_p = pick_one(δ'(p,a,Γ) ∩ asim_{i-1}(q))
37             queue = queue.put(r,next_p,visible,i-1)
38 return Z', cname'
```

Mapování větve sekundárního stavu $s \in Q$ je oproti vytváření kostry komplikovanější. Buď primární stav *prim*, pak může nastat situace, že pro dva různé sekundární stavy $r_1, r_2 \in asim_k(prim)$, mají jejich množiny následníků $follow_k(r_1)$ a $follow_k(r_2)$ neprázdný průnik. Tedy může existovat stav, který je zahrnut ve více než jedné mapované sekvenci. Duplicitní mapování jednoho stavu může vést ke zvýšení počtu přechodů automatu, a proto není povoleno.

Procedura využívá čtveřice $(p,s,i,visible)$, kde $p \in follow_k(prim)$ je stav z primární větve a stav $s \in follow_k(r)$ je jemu odpovídající stav ze sekundární, kde $s \in asim_i(p)$. Kladná hodnota i udává aktuální výšku zanoření v sekundární větvi. Pokud hodnota i klesne na 0, mapování se zastaví. Informaci, zda stav s nebyl doposud přemapován nese proměnná $visible \in \{\textbf{True}, \textbf{False}\}$. Pokud již byl počátek sekundární větve přemapován v rámci jiné sekvence, zůstává $visible =$ **False** a žádné mapování se pro dvojici stavů s,p neprovádí. V opačném případě je mapování podobné vytváření kostry procedury: přemapují se vnitřní (mezi stavy procedury), vstupní (vstupující do stavů procedury) a výstupní (vystupující ze stavů procedury) přechody. Po přemapování dvojice stavů s,p jsou do fronty dvojic dalších stavů zařazeny stavy s', p', kde $s' \in follow_k(r)$ a $p' \in follow_k(prim)$. Pokud již stav s nemá následníky, pak není dvojice s',p' generovaná, jinak $s' \in \delta(s,a,\Gamma)$, $p' \in \delta(p,a,\Gamma)$, $s' \in asim_{i-1}(p')$.

4.4 Počáteční stav a symbol na zásobníku

Pokud je počáteční stav původního automatu přemapován do procedury (bude následně smazán), musí být ustanoven nový počáteční stav a s ním i zásobníkový symbol. Mějme NZA $Z = (Q,\Sigma,\Gamma,\delta,i,S,F)$ a bijektivní zobrazení *cname*, které je výsledkem mapovacího algoritmu. Pak automat s novým počátečním symbolem je NZA $Z' = (Q,\Sigma,\Gamma \cup \{i\},\delta,cname(i),i,F)$.

4.5 Koncové stavy

Při mapování koncových stavů je zapotřebí řešit podobný problém jako u mapování počátečních stavů. Zde však existují dva přístupy. Buď jsou pomocí epsilon přechodů svedeny všechny koncové stavy do jednoho centrálního koncového stavu, nebo bude automat přijímat přechodem do koncového stavu a zároveň na zásobníku musí být symbol daný relací s koncovým stavem. V implementovaném algoritmu je využit druhý přístup.

Centrální koncový stav

Mějme NZA $Z = (Q,\Sigma,\Gamma,\delta,i,S,F)$ a bijektivní zobrazení *cname*, které je výsledkem mapovacího algoritmu. Pak automat s centrálním koncovým stavem je NZA $Z' = (Q',\Sigma,\Gamma,\delta',i,S,\{fin\})$, kde $Q' = Q \cup \{fin\} \wedge fin \notin Q$,

$$\delta'(p,a,\beta) = \begin{cases} \delta(p,a,\beta) & p \notin F \vee cname(p) \neq p, \\ \delta(p,a,\beta) \cup \{(fin,\varepsilon)\} & jinak. \end{cases}$$

Koncový stav a koncový symbol

Mějme NZA $Z = (Q,\Sigma,\Gamma,\delta,i,S,F)$ a bijektivní zobrazení *cname*, které je výsledkem mapovacího algo-

ritmu. Pak automat přijímající v koncovém stavu se symbolem definovaným relací $\Phi \subseteq F \times \Gamma$ je osmice $Z' = (Q, \Sigma, \Gamma, \delta, i, S, F, \Phi)$, kde:

$$\Phi(f) = \begin{cases} S & cname(f) = f, \\ cname(f) & jinak. \end{cases}$$

Jazyk automatu Z' je $L(Z') = \{w \in \Sigma^* \mid (i, w, Z) \vdash^* (f, \varepsilon, \Phi(f)),\ kde\ f \in F\}$.

4.6 α-redukce

Po provedení všech možných přemapování může být velikost zásobníkové abecedy rovna počtu stavů původního automatu. Její velikost je možné snížit α-redukcí popsanou algoritmem 4.

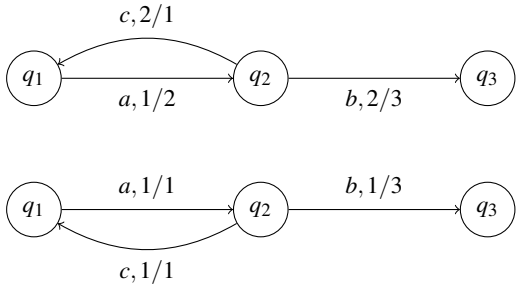
Algoritmus 4: α-redukce

Input: NZA $Z = (Q, \Sigma, \Gamma, \delta, i, F)$
Output: NZA $rename \subseteq \Gamma \times \Gamma$

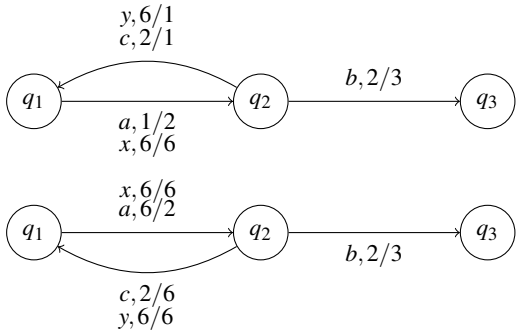
```
1  rename = {(β,β) | β ∈ Γ}
2  stack = I                                   // zásobník
3  close = ∅
4  while stack ≠ ∅ do
5      s = stack.pop()
6      if s ∈ close then
7          continue
8      close = closed ∪ {s}
9      pop_in_by_push = {(γ,β) | (q,γ) ∈ δ^-1(s,Σ,γ)}
10     pops_out = {β | (q,γ) ∈ δ(s,Σ,β)}
11     forall pop ∈ pops_out do
12         push_in = pop
13         pops_in = ⋃_{β ∈ pop_in_by_push(pop)} rename(β)
14         pops_out = ⋃_{β ∈ pops_out(pop)} rename(β)
15         if rename(pop) ∈ pops_in then
16             continue
17         possible = pops_in \ pops_out
18         if possible ≠ ∅ then
19             rename(rename(pop)) = pict_one(possible)
20     stack = stack ∪ succ(s)
21 return rename
```

Symbol na zásobníku rozlišuje, ve kterém stavu původního NKA se výpočet nachází. Je zřejmé, že pokud není využito zanořování procedur, pak dvě procedury, které na sebe nenavazují přímo mohou sdílet část zásobníkové abecedy. Místo využívání zásobníku k rozlišování stavů, můžeme také použít zásobník pouze k rozlišování, ve které větvi původního NKA se výpočet nachází. Toto zjednodušení významně ovlivní velikost použité zásobníkové abecedy.

Pro správnou funkci algoritmu 4 musí být ve vstupním NZA použit každý zásobníkový symbol pouze s jedním stavem (to je zajištěno korektním mapováním). V průběhu algoritmu platí, že pokud byl symbol $\beta \in \Gamma$ vložen na zásobník na vstupním přechodu do stavu $q \in Q$, pak musí být také vyjmut na alespoň jednom přechodu vycházejícím ze stavu q. Algoritmus prostupuje automatem do šířky a provádí substituci zásobníkových symbolů. Substituce $\alpha \in \Gamma$ za $\beta \in \Gamma$ u stavu $q \in Q$ je validní, pokud není symbol β vyjímán ze zásobníku na žádné výstupní hraně stavu q (může být vkládán).



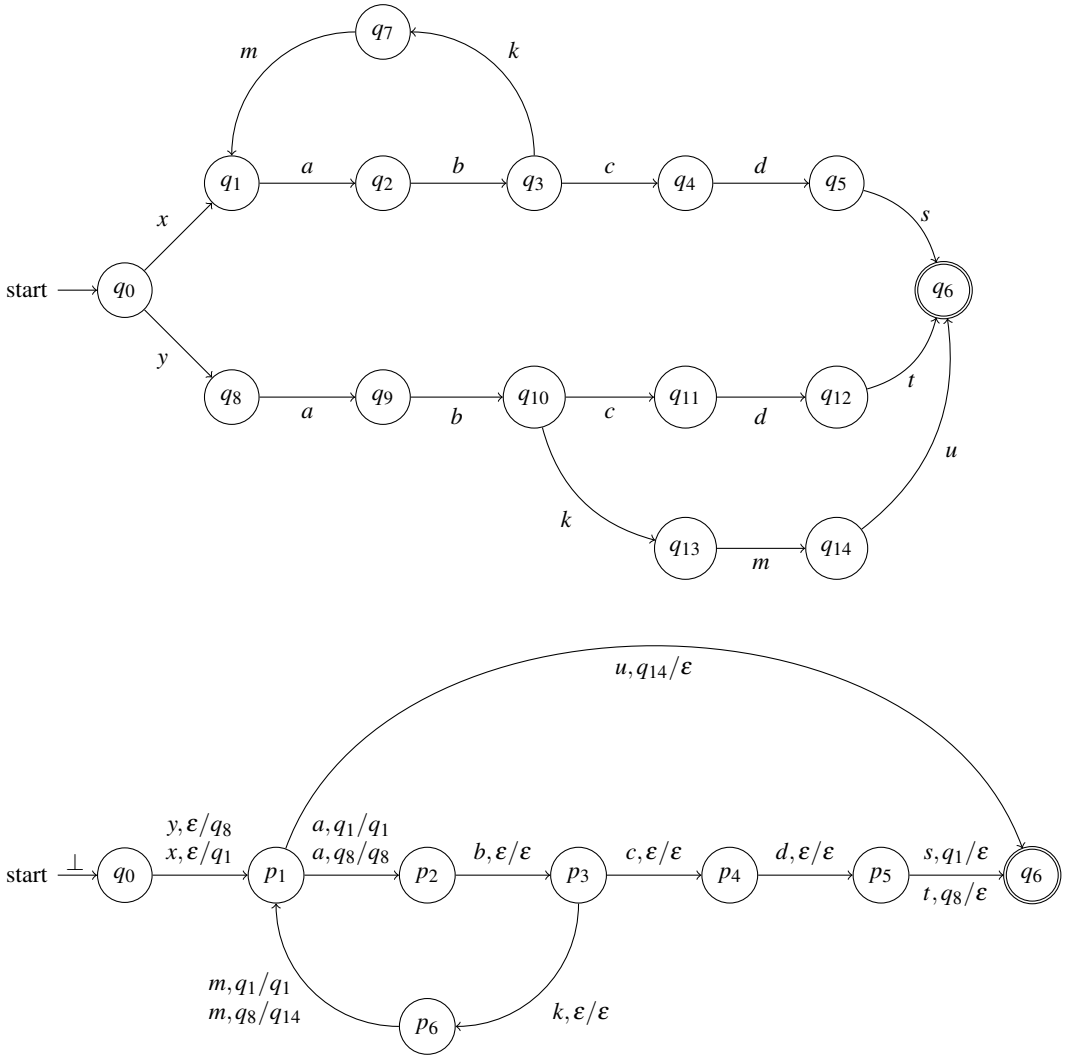
Obrázek 2. Příklad validní α-redukce na části NZA. Vrchní automat je před a spodní po α-redukci. Byla provedena substituce zásobníkového symbolu 2 za 1 (pro odlišení jsou symboly celá čísla).



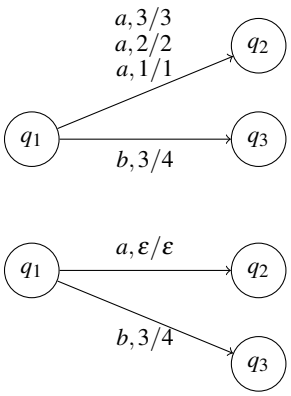
Obrázek 3. Příklad nevalidní α-redukce na části NZA. Vrchní část automatu je před a spodní po α-redukci. Byla provedena substituce zásobníkového symbolu 1 za 6 (pro odlišení jsou symboly celá čísla). Tato substituce změnila jazyk. Nyní je ve spodní části automatu možné provést sekvenci přechodů $(q_2, yx, 2) \vdash (q_1, x, 6) \vdash (q_2, \varepsilon, 6)$, která v původní verzi nebyla možná.

4.7 ε-redukce

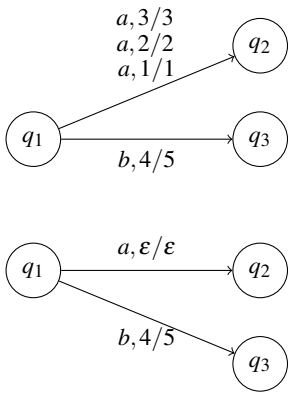
Po provedení všech přemapování a α-redukce je velikost přechodové funkce δ rovna velikosti přechodové funkci původního NKA. Hlavním cílem redukčního algoritmu je právě snížení velikosti přechodové funkce. Snížení počtu přechodů mezi stavy q a r využívajících symbol $a \in \Sigma$ lze docílit náhradou přechodů $\{(q, a, \beta) \to (r, \beta) \mid (r, \beta) \in \delta(q, a, \beta),\ \text{kde } \beta \in \Gamma\}$ za $\{(q, a, \varepsilon) \to (r, \varepsilon) \mid (r, \beta) \in \delta(q, a, \beta),\ \text{kde } \beta \in \Gamma\}$. Pro validní ε-redukci, musí platit rovnost množin symbolů $\{\beta \in \Gamma \mid (r, \beta) \in \delta(q, a, \beta)\} = \{\beta \in \Gamma \mid (r, \gamma) \in \delta(q, \Sigma, \beta)\}$, jinak nelze ε-redukce pro vybrané stavy uskutečnit, protože by došlo ke změně jazyka.



Obrázek 4. Příklad převodu NKA M (nahoře) na NZA Z (dole). Převod byl uskutečněn pomocí uvedených algoritmů (mapování stavů, α-redukce a ε-redukce). Tento převod využívající vyhledávání procedur ušetřil 7 stavů a 4 přechody při použití pouhých 3 zásobníkových symbolů ($\perp$ se nepočítá).



Obrázek 5. Příklad validní ε-redukce na části NZA. Vrchní část automatu je před a spodní po ε-redukci.



Obrázek 6. Příklad nevalidní ε-redukce na části NZA. Vrchní část automatu je před a spodní po ε-redukci. Chybná ε-redukce umožňuje ve spodní části automatu provést přechod $(q_1, a, 4) \vdash (q_2, \varepsilon, 4)$, který není v původní variantě možný.

5. Experimentální výsledky

Efektivnost redukčního algoritmu byla testována na automatech reprezentujících jazyk $\{pwp \mid \forall p \in P \, \forall w \in W : W \subseteq \Sigma^* \wedge P \subseteq \Sigma\}$. Cílem redukce bylo převést NKA na NZA tak, aby byl každý infix w reprezentován pouze jednou, a tím minimalizovat velikost přechodové funkce.

5.1 Generování testových automatů

Algoritmus 5: Generátor NKA

Input: $alphabet\_cnt, depth, in\_cnt, sections \in \mathbb{N}$, $fork\_p, loop\_p, out\_p \in \langle 0,1 \rangle \subset \mathbb{Q}$
Output: NKA $M = (Q, \Sigma, \delta, i, F)$

$\Sigma = \{a \cdot n \mid 1 \leq n \leq alphabet\_cnt\}$
$i = init$, $F = \{fin\}$, $Q = F \cup \{i\}$, $\delta = \emptyset$, $trans = \emptyset$
$states = \{1, \ldots, in\_cnt\}$, $states\_cnt = in\_cnt$, $reused = \emptyset$
$open^0 = states$
$d = 0$
$branches = 1$
while $d < depth$ **do**
    $open^{d+1} = \emptyset$
    forall $src \in open^d$ **do**
        while True do
            if $rand(\langle 0,1 \rangle) < loop\_p$ **then**
                $dst = rand(Q)$
                $reused = reused \cup \{state\}$
            else
                $states\_cnt = states\_cnt + 1$
                $dst = states\_cnt$
                $states = states \cup \{dst\}$
                if $rand(\langle 0,1 \rangle) < out\_p$ **then**
                    $states\_out = states\_out \cup \{dst\}$
            $open^{d+1} = open^{d+1} \cup \{dst\}$
            $trans = trans \cup \{(src, rand(\Sigma), dst)\}$
            if $src \in reused \vee rand(\langle 0,1 \rangle) > fork\_p$ **then**
                break
            if $((depth - d) * (branches + 1) * sections) + states\_cnt > 800/sections$ **then**
                break `// brání explozi stavů`
    $d = d + 1$
$states\_out = states\_out \cup open^d$
$used\_l = \emptyset$
forall $i \in \{0, \ldots, sections\}$ **do**
    $l = rand(\Sigma \setminus used\_l)$
    $used\_l = used\_l \cup \{l\}$
    $\delta = \delta \cup \{(init, l) \to (q + i * states\_cnt) \mid q \in states\_in\}$
    $\delta = \delta \cup \{((q + i * states\_cnt), l) \to fin \mid q \in states\_out\}$
    $\delta = \delta \cup \{((q + i * states\_cnt), a) \to (r + i * states\_cnt) \mid (q, a, r) \in trans\}$
    $Q = Q \cup \{(q + i * states\_cnt) \mid q \in states\}$
return M

Algoritmus 5 generuje NKA s $k \in \mathbb{N}$ identickými sekcemi. Vstupní a výstupní hrany jedné sekce obsahují stejný znak, který je unikátní pro každou sekci. Generování lza řídit pomocí parametrů, kde: velikost abecedy automatu značí $aplhabet\_cnt$, $sections$ určuje kolik sekcí bude vytvořeno, $depth$ udává hloubku jednotlivých sekcí, in_cnt je počet stavů v každé sekci, do kterých vede hrana z počátečního stavu automatu. V každém kroku generování nového stavu je rozhodnuto, zda z aktuálního stavu povede hrana do: koncového stavu (s pravděpodobností out_p), do k odlišných stavů (s pravděpodobností $fork\_p^{k-1}$) nebo do již vytvořeného stavu (s pravděpodobností $loop\_p$).

5.2 Výsledky testů

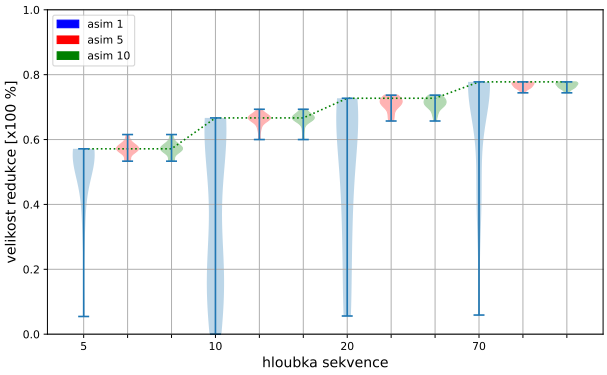
Minimalizační algoritmus využívající vyhledávání procedur a převod na NZA byl testován na 960 automatech (s pěti sekcemi) vygenerovaných pomocí algoritmu 5.

Míra redukce velikosti přechodové funkce automatu je zobrazena s použitím houslových grafů. Každý houslový graf je výsledkem minimalizace 20 automatů. Velikost abecedy poloviny automatů je 8 a druhé 16 znaků. Algoritmus byl testován na automatech s hloubkami sekvencí 5, 10, 20 a 70 pro čtyři různé kombinace parametrů $fork\_p$, $loop\_p$, in_cnt a out_p.

Každá testovací sada automatů s danou hloubkou sekvence byla minimalizovaná za použití $asim_1$ (modrá), $asim_5$ (červená) a $asim_{10}$ (zelená). Jak již bylo dříve zmíněno, tak pro minimalizaci s $asim_5$ jsou nejprve použity všechny relace z $asim_5$, poté z $asim_4$ až $asim_1$ (obdobně pro $asim_{10}$).

Pokud by byly všechny sekce nahrazeny pouze jednou společnou procedurou, pak by minimalizace dosáhla optimální velikosti. Nechť t je celkový počet přechodů, f je počet přechodů vedoucích do koncového stavu a s je počet přechodů vedoucích z počátečního stavu, poté je optimální redukce (v grafu znázorněna zelenou tečkovanou čárou) dána vzorcem:

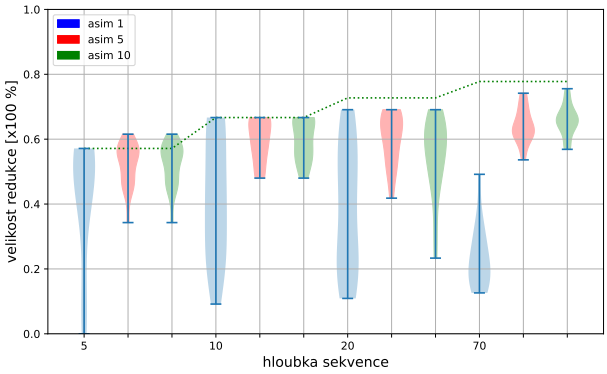
$$\frac{\frac{t-f-s}{w} + f + s}{t}$$



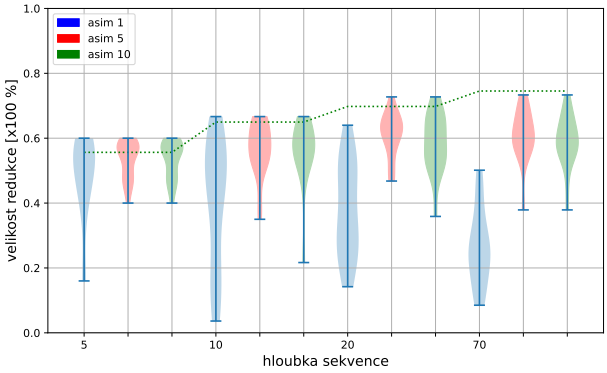
Obrázek 7. Graf výsledků minimalizace přechodové funkce automatů s následující konfigurací: $in\_cnt = 1$, $fork\_p = 0.1$, $loop\_p = 0$, $out\_p = 0$.

Z obrázku 7, a také z ostatních výsledků, je patrné, že využívání $asim_1$ přináší neoptimální výsledky. Tato neoptimálnost redukce je způsobena vytvářením procedur uvnitř jednotlivých sekcí, namísto mezi sekcemi.

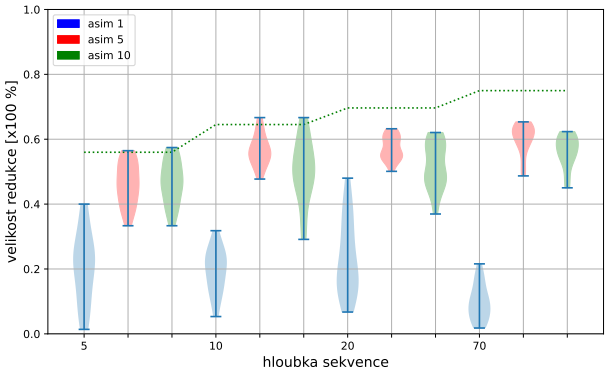
Relace $asim_1$ využívá pouze přechody do vzdálenosti 1 od zkoumaných stavů což způsobuje nález většího množství stavů v relaci $\precsim$ v rámci jedné procedury. Řešením je zvolení větší vzdálenosti, pak již bude většina stavů v relaci $\precsim$ patřit různým sekcím.



Obrázek 8. Graf výsledků minimalizace přechodové funkce automatů s následující konfigurací: $in\_cnt = 1$, $fork\_p = 0.1$, $loop\_p = 0.1$, $out\_p = 0$.

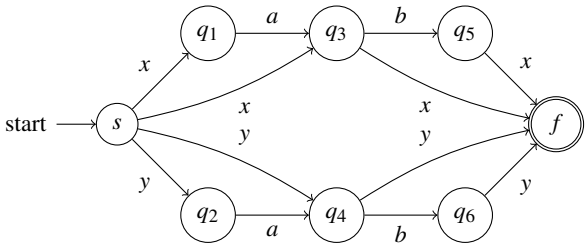


Obrázek 9. Graf výsledků minimalizace přechodové funkce automatů s následující konfigurací: $in\_cnt = 1$, $fork\_p = 0.1$, $loop\_p = 0.1$, $out\_p = 0.05$.



Obrázek 10. Graf výsledků minimalizace přechodové funkce automatů s následující konfigurací: $in\_cnt = 4$, $fork\_p = 0.1$, $loop\_p = 0.1$, $out\_p = 0.05$.

Lze vidět, že se výsledky na obrázcích 8, 9 a 10 postupně odchylují od optimálního řešení. Důvodem je složitější struktura automatu, které způsobuje, že sekce reprezentují slova (infixy) různých délek, což znemožňuje plnohodnotně využít relaci $\precsim$. Tuto situaci lze vidět na obrázku 11. Pro vyřešení tohoto problému je zapotřebí definovat relaci průniku (podobnosti) jazyků.



Obrázek 11. Příklad automatu přijímajícího slova se stejným prefixem a sufixem, kde jeho složitá struktura znemožňuje použití relace $\precsim$ pro minimalizaci. Řešením je zavedení relace jazykového průniku.

6. Závěr

Minimalizace velikosti nedeterministických konečných automatů je základní technikou snižující nároky na výpočetní zdroje při operacích s NKA. Dosavadní používané redukční metody, jakými jsou slučování stavů, prořezávání hran přechodů nebo saturace zanechávají v automatech duplicitní sekvence přechodů. Dokonce existují typy automatů, které tyto algoritmy neumí minimalizovat vůbec.

V rámci této práce byla zkoumaná nová technika redukce velikosti NKA, využívající vyhledávání podobných sekvencí přechodů v automatech a jejich nahrazení jednou "procedurou", která využívá zásobník pro uchování informace o tom, v jaké sekvenci přechodů původního automatu se výpočet nachází. Tato redukční metoda by se dala připodobnit převodu sekvenčního programu na soubor komunikujících procedur.

Zkoumaný minimalizační algoritmus byl testován na 960 (za tímto účelem vygenerovaných) automatech. Při použití relace *aproxsim* do vzdálenosti 10 přechodů od zkoumaných stavů se míra redukce blížila maximální možné teoretické hranici. Pro zvýšení míry redukce by bylo nutné použít relaci jazykového průniku (podobnosti jazyků).

7. Budoucí práce

Jak již bylo zmíněno, pro vylepšení minimalizačního algoritmu založeného na vyhledávání procedur je potřeba definovat relaci jazykové podobnosti. Tato relace

musí nejenom udávat průnik jazyků zkoumaných stavů, ale také musí určovat teoretický možný počet ušetřených přechodů, kterého by bylo dosaženo při nahrazení těchto stavů a jejich následníků procedurou.

8. Poděkování

Děkuji svému vedoucímu doc. Mgr. Lukáši Holíkovi, Ph.D za jeho podporu a čas, který mne věnoval při práci na tomto výzkumu.

Literatura

[1] ABDULLA, A. P., HOLÍK, L., CHEN, Y.-F., MAYR, R. a VOJNAR, T. *When Simulation Meets Antichains (On Checking Language Inclusion of Nondeterministic Finite (Tree) Automata)*. 2010. 22 s. Dostupné z: https://www.fit.vut.cz/research/publication/9152.

[2] ABDULLA, P. A., ATIG, M. F., CHEN, Y.-F., HOLÍK, L., REZINE, A. et al. String Constraints for Verification. In: BIERE, A. a BLOEM, R., ed. *Computer Aided Verification*. Cham: Springer International Publishing, 2014, s. 150–166. ISBN 978-3-319-08867-9.

[3] AIN, Q., SAEED, Y., NASEEM, S., AHAMD, F., ALYAS, T. et al. DNA Pattern Analysis using Finite Automata. *International Research Journal of Computer Science (IRJCS)*. Říjen 2014, roč. 1, s. 1–4.

[4] AZIZ, A., SINGHAL, V., BRAYTON, R. a SWAMY, G. Minimizing interacting finite state machines: a compositional approach to language containment. In: *Proceedings 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors*. 1994, s. 255–261.

[5] BOUAJJANI, A., HABERMEHL, P., ROGALEWICZ, A. a VOJNAR, T. Abstract regular (tree) model checking. *International Journal on Software Tools for Technology Transfer*. Apr 2012, roč. 14, č. 2, s. 167–191. Dostupné z: https://doi.org/10.1007/s10009-011-0205-y. ISSN 1433-2787.

[6] BUSTAN, D. a GRUMBERG, O. Simulation Based Minimization. In: MCALLESTER, D., ed. *Automated Deduction - CADE-17*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, s. 255–270. ISBN 978-3-540-45101-3.

[7] CLEMENTE, L. a MAYR, R. Efficient reduction of nondeterministic automata with application to language inclusion testing. *CoRR*. 2017, abs/1711.09946. Dostupné z: http://arxiv.org/abs/1711.09946.

[8] FIEDOR, T., HOLÍK, L., LENGÁL, O. a VOJNAR, T. Nested antichains for WS1S. *Acta Informatica*. 2019, roč. 56, č. 3, s. 205–228. Dostupné z: https://doi.org/10.1007/s00236-018-0331-z.

[9] FU, C., DENG, Y., JANSEN, D. a ZHANG, L. On Equivalence Checking of Nondeterministic Finite Automata. In: LARSEN, K. G., SOKOLSKY, O. a WANG, J., ed. *Dependable Software Engineering. Theories, Tools, and Applications*. Cham: Springer International Publishing, 2017, s. 216–231. ISBN 978-3-319-69483-2.

[10] HENZINGER, M., HENZINGER, T. a KOPKE, P. Computing simulations on finite and infinite graphs. In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. 1995, s. 453–462.

[11] ILIE, L., NAVARRO, G. a YU, S. On NFA Reductions. In: KARHUMÄKI, J., MAURER, H., PĂUN, G. a ROZENBERG, G., ed. *Theory Is Forever: Essays Dedicated to Arto Salomaa on the Occasion of His 70th Birthday*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, s. 112–124. Dostupné z: https://doi.org/10.1007/978-3-540-27812-2_11. ISBN 978-3-540-27812-2.

[12] RABIN, M. a SCOTT, D. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*. Duben 1959, roč. 3, s. 114–125.

[13] SOURDIS, I. a PNEVMATIKATOS, D. Fast, Large-Scale String Match for a 10Gbps FPGA-Based Network Intrusion Detection System. In: Y. K. CHEUNG, P. a CONSTANTINIDES, G. A., ed. *Field Programmable Logic and Application*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, s. 880–889. ISBN 978-3-540-45234-8.