BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

Simulačný algoritmus pre symbolické automaty

Juraj Síč*

Abstrakt
V tomto texte sa zaoberáme výpočtom simulačnej relácie pre symbolické automaty, ktoré narozdiel od klasických automatov majú na prechodoch predikáty, zadefinované v oddelenej logickej teórii. Náš algoritmus pre výpočet tejto relácie je založený na známom algoritme pre konečné automaty a je upravený tak, aby využil možnosti symbolických automatov.

Kľúčové slová: symbolický automat — konečný automat — relácia simulácie

Priložené materiály: *N/A*

*xsicju00@stud.fit.vutbr.cz, *Fakulta informačných technológií, Vysoké učení technické v Brne*

1. Úvod

Klasická teória automatov predpokladá, že abeceda je konečná. V niektorých prípadoch môže táto abeceda byť tak veľká, že začne spôsobovať problémy s veľkosťou automatov. Napríklad z každého stavu deterministického konečného automatu prijímajúceho jazyk nad UTF16 abecedou vychádza 2^{16} prechodov. Niekedy je dokonca potrebné používať nekonečnú abecedu, pri ktorých je použitie klasických automatov vylúčené.

Tieto problémy je možné vyriešiť zavedením symbolických konečných automatov (SKA) [9]. Tieto automaty sú parametrizované logickou teóriou nad doménou symbolov. Na prechodoch sú formule z tejto teórie s jednou voľnou premennou, ktoré reprezentujú potencionálne nekonečné množiny symbolov.

Symbolická reprezentácia je oproti klasickej kompaktnejšia a dovoľuje vytvárať efektívnejšie algoritmy. Problémom je, že táto reprezentácia je pomerne nová a veľa takýchto efektívnych algoritmov, ktoré využívajú jej kompaktnejšiu formu, pre ňu neexistuje. Uskutočňuje sa pre ne výskum algoritmov, kde sa zovšeobecňuje teória a algoritmy pre klasické automaty, ale zatiaľ existujú len základné algoritmy pre deterministické SKA, ako napríklad ich minimalizácia [3]. Pre nedeterministické SKA, ktoré sú menšie a pre niektoré aplikácie lepšie využiteľné, je ešte potrebné veľa algoritmov doplniť. Pre výpočet jednej z dôležitých relácií nedeterministických SKA, reláciu simulácie, neexistuje efektívny algoritmus. Táto relácia sa využíva na redukciu počtu stavov nedeterministických automatov, ako ukázali Ilie, Navarro a Yu [6] a na dokazovanie jazykovej inklúzie [1, 2].

V tomto texte ukážeme, ako sa algoritmus pre výpočet simulácie pre klasický nedeterministický konečný automat [6] dá aplikovať na symbolický konečný automat. V kapitole 2 sú uvedené definície automatov, kapitola 3 sa zaoberá simuláciou a algoritmom pre výpočet simulácie pre nedeterministický konečný automat a nakoniec v kapitole 4 je rozobratý algoritmus pre výpočet simulácie pre nedeterministický SKA.

2. Konečné automaty

V tejto kapitole je vysvetlené, čo sú to nedeterministické konečné automaty a symbolické konečné automaty (prevzaté z [9]).

Definícia 2.1 *Nedeterministický konečný automat (NKA) M je pätica $M = (Q, \Sigma, \delta, q_0, F)$, kde Q je konečná množina stavov, Σ je abeceda, $\delta \subseteq Q \times \Sigma \times Q$ je konečná množina prechodov, $q_0 \in Q$ je počiatočný stav a $F \subseteq Q$ je množina koncových stavov.*

Elementy Σ sa nazývajú symboly a konečné postupnosti symbolov (elementy Σ^*) sa nazývajú reťazce; ε značí prázdny reťazec. Prechod $r = (p, a, q) \in \delta$, kde $p, q \in Q$ a $a \in \Sigma$, označujeme $p \xrightarrow{a}_M q$ (alebo $p \xrightarrow{a} q$ ak je M zrejmý).

Definícia 2.2 *Nech $M = (Q, \Sigma, \delta, q_0, F)$ je NKA. Reťazec $w = a_1 a_2 \cdots a_k \in \Sigma^*$ je prijatý stavom p NKA*

M, značíme $w \in \mathcal{L}_p(M)$, *ak existuje* $p_{i-1} \xrightarrow{a_i}_M p_i$ *pre* $1 \leq i \leq k$, *kde* $p_0 = p$ *a* $p_k \in F$. *Jazyk príjmaný M je* $\mathcal{L}(M) \stackrel{\text{def}}{=} \mathcal{L}_{q_0}(M)$.

Efektívna Booleova algebra $\mathcal{A}$ má komponenty $(\mathfrak{D}, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$. $\mathfrak{D}$ je r.v. (rekurzívne vyčísliteľná) množina *doménových elementov*. Ψ je r.v. množina *predikátov* uzavretých voči $\vee, \wedge, \neg$ a $\bot, \top \in \Psi$. *Významová funkcia* $[\![\_]\!] : \Psi \to 2^{\mathfrak{D}}$ je r.s., $[\![\bot]\!] = \emptyset, [\![\top]\!] = \mathfrak{D}$ a pre všetky $\varphi, \psi \in \Psi, [\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!], [\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$ a $[\![\neg\varphi]\!] = \mathfrak{D} \setminus [\![\varphi]\!]$. Pre $\varphi \in \Psi$, píšeme $IsSat(\varphi)$ keď $[\![\varphi]\!] \neq \emptyset$ a hovoríme, že φ je *splniteľná*. $\mathcal{A}$ je *rozhodnuteľná*, ak *IsSat* je rozhodnuteľná.

Definícia 2.3 *Symbolický konečný automat (SKA) M je pätica* $(\mathcal{A}, Q, q_0, F, \Delta)$, *kde* $\mathcal{A}$ *je efektívna Booleova algebra, nazývaná abeceda, Q je konečná množina stavov,* $q_0 \in Q$ *je počiatočný stav,* $F \subseteq Q$ *je množina koncových stavov a* $\Delta \subseteq Q \times \Psi_{\mathcal{A}} \times Q$ *je konečná množina prechodov.*

Elementy $\mathfrak{D}_{\mathcal{A}}$ sa nazývajú symboly a konečné postupnosti symbolov (elementy $\mathfrak{D}^*_{\mathcal{A}}$) sa nazývajú reťazce; ε značí prázdny reťazec. Prechod $\rho = (p, \varphi, q) \in \Delta$, taktiež označený $p \xrightarrow{\varphi}_M q$ (alebo $p \xrightarrow{\varphi} q$ ak je M zrejmý), kde p je *zdrojový* stav, označený ako $Src(\rho)$, q je *cieľový* stav, označený ako $Tgt(\rho)$ a φ je *predikát* prechodu, označený ako $Grd(\rho)$. Prechod je *uskutočniteľný*, ak je jeho predikát splniteľný. Ak je daný symbol $a \in \mathfrak{D}_{\mathcal{A}}$, a-prechod M je prechod $p \xrightarrow{\varphi} q$ taký, že $a \in [\![\varphi]\!]$, tiež označený $p \xrightarrow{a}_M q$ (alebo $p \xrightarrow{a} q$ ak je M zrejmý).

Definícia 2.4 *Nech* $M = (\mathcal{A}, Q, q_0, F, \Delta)$ *je SKA. Reťazec* $w = a_1 a_2 \cdots a_k \in \mathfrak{D}^*_{\mathcal{A}}$ *je prijatý stavom p SKA M, značíme* $w \in \mathcal{L}_p(M)$, *ak existuje* $p_{i-1} \xrightarrow{a_i}_M p_i$ *pre* $1 \leq i \leq k$, *kde* $p_0 = p$ *a* $p_k \in F$. *Jazyk príjmaný M je* $\mathcal{L}(M) \stackrel{\text{def}}{=} \mathcal{L}_{q_0}(M)$.

Pre $q \in Q$, používame notáciu:

$$\overrightarrow{\Delta}(q) \stackrel{\text{def}}{=} \{\rho \in \Delta : Src(\rho) = q\},$$

$$\overleftarrow{\Delta}(q) \stackrel{\text{def}}{=} \{\rho \in \Delta : Tgt(\rho) = q\}.$$

Nasledujúca terminológia popisuje rôzne kľúčové vlastnosti M. Stav p automatu M nazývame *čiastočný* ak existuje symbol a, pre ktorý neexistuje a-prechod z p. M je:

- *deterministický*, ak pre každé $p \xrightarrow{\varphi} q, p \xrightarrow{\varphi'} q' \in \Delta$: ak $IsSat(\varphi \wedge \varphi')$, potom $q = q'$. Determinizácia SKA je vždy možná [10].
- *úplný*, ak neobsahuje žiadne čiastočné stavy. Úplný SKA sa dá vytvoriť pridaním nového stavu $q_\emptyset$, prechodu $q_\emptyset \xrightarrow{\top} q_\emptyset$ a pre každý stav q pridaním prechodu $(q, \bigwedge_{\rho \in \overrightarrow{\Delta}(q)} \neg Grd(\rho), q_\emptyset)$.
- *čistý*, ak pre každé $p \xrightarrow{\varphi} q \in \Delta$: p je dostupný z q_0 a $IsSat(\varphi)$.
- *normalizovaný*, ak pre každé $p, q \in Q$, existuje najviac jeden prechod z p do q. Ak existujú stavy p a q a dva rôzne prechody $p \xrightarrow{\varphi} q$ a $p \xrightarrow{\psi} q$, tak ak tieto nahradíme jedným prechodom $p \xrightarrow{\varphi \vee \psi} q$, vytvoríme normalizovaný SKA.

3. Simulácia

Nech $M = (Q, \Sigma, \delta, q_0, F)$ je NKA. *Simulácia* na M je kváziusporiadanie $\preceq \in Q \times Q$ také, že:

(i) $\preceq \cap (F \times (Q - F)) = \emptyset$,
(ii) pre $p, q \in Q$, $a \in \Sigma$, ak $p \preceq r$, potom pre každý prechod $p \xrightarrow{a} p'$ existuje $r \xrightarrow{a} r'$ taký, že $p' \preceq r'$.

Pre SKA je táto definícia skoro rovnaká, len v (ii) bude $a \in \mathfrak{D}_{\mathcal{A}}$.

Jednou z aplikácií simulácie je redukcia veľkosti automatu spájaním stavov a to tak, že stavy $p, r \in Q$ zlúčime, keď sú ekvivalentné podľa relácie simulácie ($p \preceq r$ a $r \preceq p$)[1], keďže $p \preceq r$ implikuje $\mathcal{L}_p(M) \subseteq \mathcal{L}_r(M)$.

Existuje viacero algoritmov na výpočet simulácie na Kripkeho štruktúrach, ktoré zhrnul Ranzato v [7]. Základným algoritmom pre výpočet simulácie v triede moderných algoritmov s dobrou zložitosťou je algoritmus od Henzingera, Henzingerovej a Kopkeho [5]. Od neho je odvodených viacero ďalších, ešte efektívnejších algoritmov (napríklad algoritmus od Ranzata a Taparra [8]). Nezávisle od týchto vytvorili Ilie, Navarro a Yu algoritmus pre výpočet simulácie aplikovaný na NKA [6], ktorý je veľmi podobný základnému algoritmu. Tento využíva jednoduchšie dátové štruktúry, preto algoritmus 2 vychádza z tohto algoritmu.

Ich algoritmus počíta doplnok k $\preceq$:

$$\not\preceq = Q \times Q \setminus \preceq.$$

Podľa definície $\preceq$ je jej doplnok $\not\preceq$ najmenšia relácia nad Q taká, že:

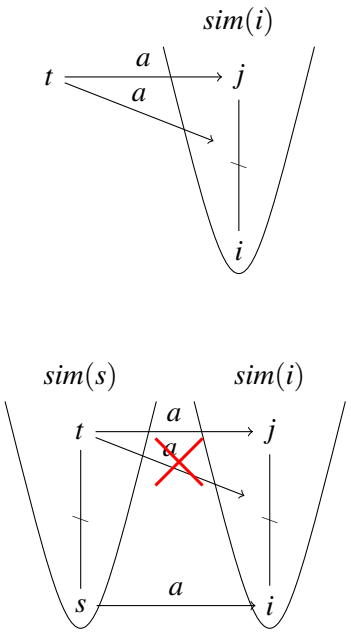
(i)' $(F \times (Q - F)) \subseteq \not\preceq$,
(ii)' pre $i, j \in Q$ je $i \not\preceq j$ práve vtedy, keď existujú $a \in \Sigma$ a prechod $i \xrightarrow{a} i'$ také, že pre každé $j' \in Q$, pre ktoré existuje prechod $j \xrightarrow{a} j'$, platí $i' \not\preceq j'$.

[1] V [6] spájajú stavy, aj keď sú v relácii simulácie obráteného automatu (pravidlo $p \xrightarrow{a} q$ nahradí $q \xrightarrow{a} p$), ale potom je potrebné upravovať $\preceq$.

Na začiatku (kroky 3-6) si podľa (i)' inicializujú reláciu ω, ktorá sa na konci stane $\not\preceq$. Túto reláciu postupne zväčšujú a to tak, že si novo pridané dvojice do ω ukladajú do frontu (kroky 6 a 16) a potom ich z neho neskôr vyberajú (krok 8), aby použitím (ii)', pridali nové dvojice do ω. Teda ak spracovávajú dvojicu (i,j), potrebujú otestovať, či existuje $t \in Q$ a $a \in \Sigma$, také, že $t \xrightarrow{a} j$ a zároveň je j posledný stav, ktorý by mohol simulovať i. Aby sa toto urýchlilo, vypočítavajú si matice počítadiel $N(a)$, pre $a \in \Sigma$, kde $N(a)$ je $|Q| \times |Q|$ matica taká, že

$$N(a)_{it} = |\{t \xrightarrow{a} j, i \not\preceq l\}|$$

na konci výpočtu algoritmu pre všetky $i,t \in Q$. Tieto počítadlá sú inicializované na 0 a pri každom páre (i,j) sa pre všetky $t \in Q$ a $a \in \Sigma$, kde $t \xrightarrow{a} j$, zvýšia o jedna (krok 11) a ak dané počítadlo nadobudne maximálnu hodnotu $|\{p : t \xrightarrow{a} p\}|$ (krok 12), všetky páry (s,t) také, že $s \xrightarrow{a} i$, sú pridané do ω, ak tam ešte nie sú.



Obrázok 1. Spracovanie (i,j) keď existuje z t nejaký prechod do $sim(i)$ a keď takýto prechod už neexistuje.

Obrázok 1 ilustruje, čo to vlastné znamená. $sim(p)$ označuje množinu $sim(p) = \{q \in Q : p \preceq q\}$, pre nejaké $p \in Q$. Spracovávame (i,j), takže sa najprv nájde t, ktoré ide do j cez a. Počítadlo $N(a)_{it}$ (počet stavov, do ktorých ide t a nie sú v $sim(i)$) sa zvýši o jedna. Ak existuje nejaký prechod $t \xrightarrow{a} j'$, kde $j' \in sim(i)$, tak počítadlo $N(a)_{it}$ nenadobudlo svoju najväčšiu hodnotu a nič nové do $\not\preceq$ nepribudne. Ale ak taký prechod už neexistuje, znamená to, že pre každé s, ktoré ide do i cez a, je pár (s,t) pridaný do ω.

Input: NFA M
Output: $\not\preceq$

```
1  nastav všetky N(a) na 0
2  ω ← ∅, C ← NEWQUEUE()
3  for i ∈ F do
4      for j ∈ Q − F do
5          ω ← ω ∪ {(i, j)}
6          ENQUEUE(C, (i, j))
7  while C ≠ ∅ do
8      (i, j) ← DEQUEUE(C)
9      for a ∈ Σ do
10         for t where ∃t -a-> j do
11             N(a)_it ← N(a)_it + 1
12             if N(a)_it = |{p : ∃(t, a, p)}| then
13                 for s where ∃s -a-> i do
14                     if (s, t) ∉ ω then
15                         ω ← ω ∪ {(s, t)}
16                         ENQUEUE(C, (s, t))
17 return ω
```

Algoritmus 1: Algoritmus pre výpočet doplnku $\not\preceq$ NKA.

4. Vlastný algoritmus

Algoritmus 2 vychádza z algoritmu 1, takže počíta doplnok k $\preceq$, $\not\preceq$, ale pre SKA. Pre $\not\preceq$ SKA, platí (i)', ale pre (ii)' berieme $a \in \mathfrak{D}_{\mathcal{A}}$. V originálnom algoritme si vypočítajú matice počítadiel $N(a)$, pre $a \in A$. To pre symbolické automaty nie je možné, pretože prechody sú značené predikátmi a abeceda je príliš veľká (alebo nekonečná), čo by mohlo spôsobiť, že by sa algoritmus nikdy neukončil. Takže namiesto počítadiel nad symbolmi budeme používať počítadla nad predikátmi.

Pre každý stav $q \in Q$ si zavedieme *rozklad na lokálne mintermy*, čo je najhrubší rozklad $\mathfrak{D}_{\mathcal{A}}$, ktorý je kompatibilný s predikátmi nad prechodmi vychádzajúcimi z q. Potom vytvoríme počítadlá nad týmito mintermami (kroky 1-19) a ich počiatočná hodnota bude počet predikátov vychádzajúcich z q, ktorých konjunkcia s mintermom je splniteľná. Toto nastavenie znamená, že $\not\preceq = \emptyset$ alebo „všetko simuluje všetko".

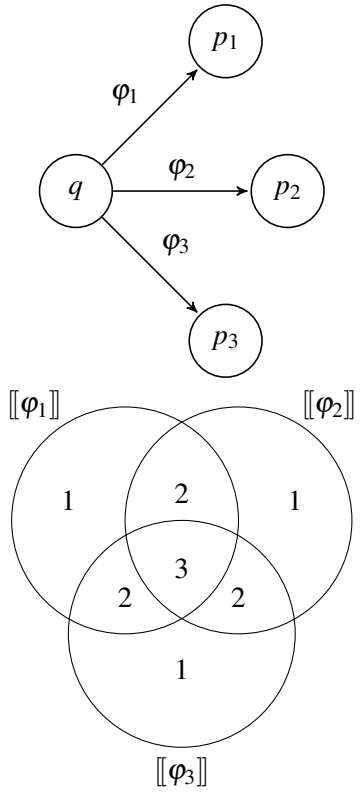
Napríklad majme stav q, z ktorého vychádzajú 3 prechody (obrázok 2) s predikátmi φ_1, φ_2, φ_3. Mintermov bude 7: $\varphi_1 \wedge \varphi_2 \wedge \varphi_3, \varphi_1 \wedge \varphi_2 \wedge \neg\varphi_3, \varphi_1 \wedge \neg\varphi_2 \wedge \varphi_3 \ldots$ Ak je nejaký minterm nesplniteľný, tak počítadlo preň nevytvoríme, inak budú hodnoty počítadiel nastavené napríklad pre minterm $\varphi_1 \wedge \varphi_2 \wedge \neg\varphi_3$ na 2, ale pre $\neg\varphi_1 \wedge \varphi_2 \wedge \neg\varphi_3$ na 1...

Input: SFA M
Output: $\not\preceq$

```
   /* nastavenie počítadiel                    */
1  for ρ ∈ Δ do
2      φ ← Grd(ρ)
3      q ← Src(ρ)
4      for ψ where ∃N(ψ)_{-q} and IsSat(ψ ∧ φ)
       do
5          if IsSat(ψ ∧ ¬φ) then
6              create N(ψ ∧ ¬φ)_{-q} ← N(ψ)_{-q}
7          create N(ψ ∧ φ)_{-q} ← N(ψ)_{-q} + 1
8          remove N(ψ)_{-q}
9          φ ← φ ∧ ¬ψ
10     if IsSat(φ) then
11         create N(φ)_{-q} ← 1
12 forall the N(ψ)_{-q} do
13     for r ∈ Q do
14         create N(ψ)_{rq} ← N(ψ)_{-q}
15 ω ← ∅, C ← NEWQUEUE()
   /* inicializácia podľa (i)'                 */
16 for i ∈ F do
17     for j ∈ Q − F do
18         ω ← ω ∪ {(i, j)}
19         ENQUEUE(C, (i, j))
   /* aktualizácia podľa (ii)'                 */
20 while C ≠ ∅ do
21     (i, j) ← DEQUEUE(C)
22     S ← pre(i), T ← pre(j)
23     for t ∈ T where t -φ_tj-> j do
24         for ψ where ∃N(ψ)_it and
           IsSat(ψ ∧ φ_tj) do
25             N(ψ)_it ← N(ψ)_it − 1
26             if N(ψ)_it = 0 then
27                 for s ∈ S where s -φ_si-> i and
                   IsSat(ψ ∧ φ_si) do
28                     if (s,t) ∉ ω then
29                         ω ← ω ∪ {(s,t)}
30                         ENQUEUE(C, (s,t))
31 return ω
```

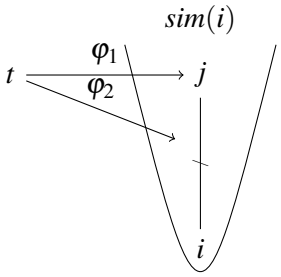
Algoritmus 2: Simulácia SKA. Očakáva, že M je čistý, normalizovaný a úplný.

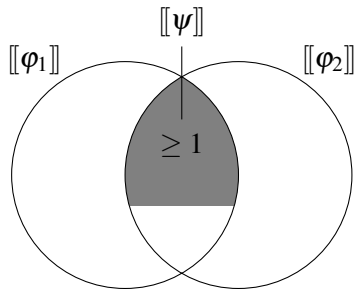
Aktualizácia počítadiel nastáva podobne ako v originálnom algoritme. Každý nový pár v $\not\preceq$ sa zaradí do frontu (kroky 19 a 30). V krokoch 16-19 sa do $\not\preceq$ zaradia všetky $(F \times (Q - F))$. Pri tom, ako ich spracovávame (krok 21), sa všetky potrebné počítadlá znížia o jedna (krok 25). Ak niektoré z týchto počítadiel $N(\psi)_{ti}$ nadobudne hodnotu 0 (krok 26), všetky (s,t), také, že $s \xrightarrow{\varphi_{si}} i$ a $\psi \wedge \varphi_{si}$ je splniteľné, sú vložené do $\not\preceq$, ak tam ešte nie sú.



Obrázok 2. Nastavenie počítadiel.

Na obrázkoch 3 a 4 je zobrazený tento postup. Spracovávame pár (i, j), takže pre každé $t \in Q$, ktoré ide do j, musíme aktualizovať počítadlá. Všetky počítadlá $N(\psi)_{ti}$, pre ktoré $\varphi_1 \wedge \psi$ je splniteľné (ψ je mintermom φ_1), sa znížia o jedna. Ak existuje iný

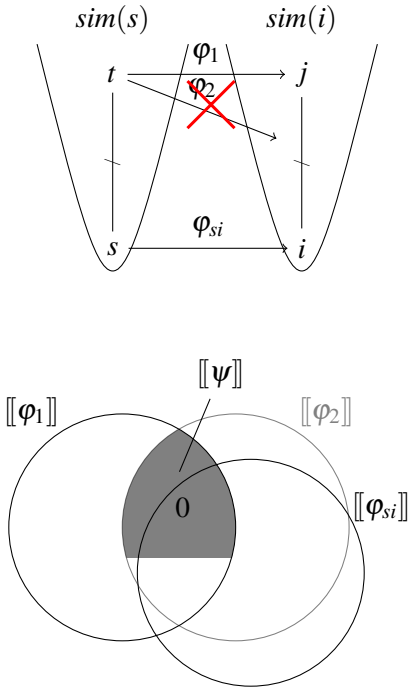




Obrázok 3. Existuje prechod φ_2, kde ψ je jeho mintermom.

prechod z t do $sim(i)$ označený φ_2, pre ktorý je ψ

jeho mintermom, $N(\psi)_{it}$ bude stále väčší alebo rovný jednej, takže ω sa nezmení. Ak takýto prechod ne-



Obrázok 4. Neexistuje prechod φ_2, kde ψ je jeho mintermom.

existuje, potom je prechod $t \xrightarrow{\varphi_1} j$ posledný, pre ktorý je ψ mintermom a teda počítadlo $N(\psi)_{it}$ nadobudne hodnotu nula a do ω sú vložené všetky (s,t), pre ktoré $s \xrightarrow{\varphi_{si}} i$ a $\varphi_{si} \wedge \psi$ je splniteľné.

5. Záver a ďalšia práca

V tejto práci sme prezentovali algoritmus, ktorý vypočíta reláciu simulácie symbolických konečných automatov, ktorý vychádza zo základného algoritmu pre výpočet simulácie konečných automatov [6]. Tento algoritmus je založený na lokálnom rozkladu prechodovej relácie na mintermy.

Plánujeme tento algoritmus implementovať a otestovať na príkladoch symbolických automatov generovaných nástrojom MONA pri rozhodovaní logiky WS1S [4] a na testoch použitých pri minimalizácii deterministických SKA [3]. Veríme, že sa potvrdí hypotéza, že nedeterministická redukcia počtu stavov pomocou simulácie dáva oveľa menšie automaty ako ich deterministická minimalizácia. Taktiež uskutočníme analýzu zložitosti a dôkaz korektnosti.

Literatúra

[1] Abdulla, P. A.; Chen, Y.; Holík, L.; aj.: When Simulation Meets Antichains. In *TACAS*, *Lecture Notes in Computer Science*, ročník 6015, Springer, 2010, s. 158–174.

[2] Bustan, D.; Grumberg, O.: Simulation Based Minimization. In *Automated Deduction - CADE-17, 17th International Conference on Automated Deduction, Pittsburgh, PA, USA, June 17-20, 2000, Proceedings*, *Lecture Notes in Computer Science*, ročník 1831, editace D. A. McAllester, Springer, 2000, ISBN 3-540-67664-3, s. 255–270, doi:10.1007/10721959˙20.

[3] D'Antoni, L.; Veanes, M.: Minimization of symbolic automata. In *POPL*, ACM, 2014, s. 541–554.

[4] Elgaard, J.; Klarlund, N.; Møller, A.: MONA 1.x: New Techniques for WS1S and WS2S. In *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, *Lecture Notes in Computer Science*, ročník 1427, editace A. J. Hu; M. Y. Vardi, Springer, 1998, ISBN 3-540-64608-6, s. 516–520, doi: 10.1007/BFb0028773.

[5] Henzinger, M. R.; Henzinger, T. A.; Kopke, P. W.: Computing Simulations on Finite and Infinite Graphs. In *FOCS*, IEEE Computer Society, 1995, s. 453–462.

[6] Ilie, L.; Navarro, G.; Yu, S.: On NFA Reductions. In *Theory Is Forever*, *Lecture Notes in Computer Science*, ročník 3113, Springer, 2004, s. 112–124.

[7] Ranzato, F.: A More Efficient Simulation Algorithm on Kripke Structures. In *MFCS*, *Lecture Notes in Computer Science*, ročník 8087, Springer, 2013, s. 753–764.

[8] Ranzato, F.; Tapparo, F.: A New Efficient Simulation Equivalence Algorithm. In *LICS*, IEEE Computer Society, 2007, s. 171–180.

[9] Veanes, M.: Applications of Symbolic Finite Automata. In *CIAA*, *Lecture Notes in Computer Science*, ročník 7982, Springer, 2013, s. 16–23.

[10] Veanes, M.; Bjørner, N.; de Moura, L. M.: Symbolic Automata Constraint Solving. In *LPAR (Yogyakarta)*, *Lecture Notes in Computer Science*, ročník 6397, Springer, 2010, s. 640–654.