BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

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

Inkluze jazyků nedeterministických stromových automatů

Petr Žufan*

Abstrakt
Tento článek pojednává o testování ekvivalence stromových automatů (SA). Přináší nový algoritmus vycházející z algoritmu Bonchiho a Pouse pro slovní automaty. Tento nový algoritmus spojuje bisimulaci s determinizací za běhu. Pomocí optimalizace založené na kongruenčním uzávěru se snaží vyhýbat extrémnímu zvětšování stavového prostoru. Tento algoritmus by mohl být efektivnější než jiné metody pro tento problém.

Klíčová slova: automat — stromový automat — inkluze jazyků

Přiložené materiály:

*xzufan00@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Stromové automaty jsou rozšířením slovních automatů, které přijímají stromy namísto slov. Dá se říci, že slovo je strom, jehož každý uzel má pouze jednoho potomka. Jsou užívané v mnoha odvětvích počítačových věd: databáze a XML [1], zpracování přirozených jazyků, model checking [2], formální verifikace systémů [3]. Dá se říci, že stromové automaty jsou využitelné všude tam, kde se zpracovávají regulární množiny stromů.

Zjištění ekvivalence dvou slovních automatů je PSPACE-úplný problém [4]. Pro tento problém existuje několik algoritmů. Nejzákladnější z nich vychází z komplementace, pro niž je potřeba determinizace, a průniku. Právě kvůli determinizaci se projevuje velká teoretická složitost problému. V poslední době byly vyvinuty algoritmy které se snaží vyhnout velké teoretické složitosti nejhorších případů. První z nich je založen na protiřetězcích [5]. Později byl vyvinut algoritmus Bonchiho a Pouse, na kterém budeme stavět v tomto článku. Tento algoritmus vychází z algoritmu Hopcrofta a Karpa pro výpočet bisimulace deterministických automatů, který intuitivně souběžně simuluje průchod automaty a testuje, zda se shodují. Navíc počítá optimalizaci pomocí kongruenčního uzávěru. Algoritmus Bonchiho a Pouse, stejně jako protiřetězce provádí determinizaci automatu za běhu a v některých případech jsou tedy výhodnější. O tom si povíme v první kapitole.

Druhá kapitola je zaměřena na stromové automaty, které jsou v principu velmi podobné těm slovním. Definujeme je a ukážeme algoritmus jejich determinizace.

Ekvivalence stromových automatů je EXPTIME-úplný problém [6]. To je ještě o něco horší než u těch slovních. Algoritmy však staví na stejných myšlenkách. Existují metody založené na komplementech i na protiřetězcích. Na algoritmus vycházející z bisimulace a z HK se podíváme ve třetí kapitole.

2. Slovní automaty

Slovní automat (definice 2.1) je matematický model zpracovávající řetězce symbolů (slova) vstupní abecedy. Slova jsou automatem buď přijímána nebo ne. Množině přijímaných slov se říká jazyk. Každý automat tedy definuje nějaký jazyk. Základní typ automatu je nedeterministický konečný automat (dále NKA).

Definice 2.1. [7] **Nedeterministický konečný automat** je pětice $M = (Q, \Sigma, \delta, q_0, F)$, kde Q je konečná množina stavů, Σ je konečná množina vstupních symbolů, $\delta : Q \times \Sigma \to \mathscr{P}(Q)$ je konečná množina pravidel,

$q_0 \in Q$ je počáteční stav a $F \subseteq Q$ je množina koncových stavů.

Jazyk automatu $L(A)$ (definice 2.2) je potom množina všech slov, po jejichž zpracování skončí automat ve svém koncovém stavu.

Definice 2.2. [8] **Jazyk** definovaný automatem A je označen $L(A)$ a definován:
L je množina slov $a_1, \ldots, a_n$, takových, že existují stavy $q_1, \ldots, q_n$, takové, že pro $i : 1 \leq i \leq n$ platí $q_i \in \delta(q_{i-1}, a_i)$ a $q_n \in F$.

Práce s NKA je složitá, avšak jsou přehlednější a někdy lze díky nim ušetřit prostor i čas. Jako Alternativa k NKA existují deterministické konečné automaty (definice 2.3) (dále DKA). Bylo dokázáno, že každý DKA lze převést na NKA a obráceně. Tento proces se nazývá determinizace.

2.1 Determinizace

Všechny algoritmy testu ekvivalence dvou jazyků nějakým způsobem determinizaci provádí. Některé determinizují automat před samotným testováním, jiné zapojí determinizaci přímo do algoritmu a snaží se zmenšit stavový prostor. Pro pochopení algoritmu determinizace NKA, musíme nejdříve popsat DKA, který z tohoto procesu vzniká.

Definice 2.3. [7] **Deterministický konečný automat** je konečný automat $M = (Q, \Sigma, \delta, q_0, F)$ takový, že pro všechna $q \in Q$ a pro všechna $a \in \Sigma$ platí $|\delta(q,a)| \leq 1$.

Vidíme, že u DKA na rozdíl od NKA, existuje právě jeden stav, do kterého se dostaneme z daného stavu a s daným symbolem na vstupu. A právě proto jsou DKA jednodušší pro práci s nimi. Dostáváme se tedy k problému, jak determinizovat NKA. Popíšeme algoritmus 2.4 nazývaný podmnožinová konstrukce.

Algoritmus 2.4. [7] Mějme NKA $M = (Q, \Sigma, \delta, q_0, F)$ přijímající nějaký jazyk A, sestrojíme DKA $M' = (Q', \Sigma', \delta', q_0', F')$ přijímající stejný jazyk A.

1. $\Sigma' = \Sigma$, $Q' = \mathscr{P}(Q)$. Q' je tedy množina všech podmnožin množiny Q.
2. $\delta'(R,a) = \{q \in Q \mid q \in E(\delta(r,a)), \forall r \in R\}$ kde $R \in Q'$, $a \in \Sigma'$.
3. $q_0' = \{q_0\}$.
4. $F' = \{R \in Q' \mid R$ obsahuje $q \in F\}$.

Dodatečně musíme nadefinovat množinu $E(R)$, kterou jsme použili v algoritmu 2.4. Je to množina všech stavů dosažitelných ze stavu R použitím pouze ε-pravidel. Formálně $E(R) = \{q \mid q$ je dosažitelné z R použitím 0 nebo více ε-pravidel$\}$. Do této množiny patří tedy i samotný stav R.

Nově vzniklý DKA obsahuje 2^n stavů, oproti n stavům původního NKA. Mohli bychom DKA ještě minimalizovat odstraněním stavů nedostupných z počátečního stavu, odstraněním stavů nadbytečných a odstranění stavů ekvivalentních.

2.2 Porovnání jazyků KA

Základní algoritmus pro porovnání dvou jazyků je založen na implementaci jazykových operací komplementace a průniku nad automaty. Pokud tento průnik bude prázdná množina, jazyky ekvivalentní jsou. Jinak popsáno, $L(A) = L(B) \iff L(A) \cap \overline{L(B)} = \emptyset$.

Při aplikaci tohoto algoritmu na NKA musíme tento automat prvně determinizovat. Poté jej můžeme komplementovat výměnou koncových stavů za nekoncové a obráceně. Z automatu $M = (Q, \Sigma, \delta, q_0, F)$ vytvoříme automat $M' = (Q, \Sigma, \delta, q_0, F')$ tak, že $F' = Q - F$.

Stejně tak i konstrukce paralelního synchronního produktu dvou jazyků automatů je pro DKA realizovaná jednoduchým algoritmem (2.5).

Algoritmus 2.5. [7] Mějme automaty $M_A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ a $M_B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$ vytvoříme automat $N = (Q, \Sigma, \delta, q_0, F)$, takový že $L(M_A) \cap L(M_B) = L(N)$ následovně:

1. $Q = Q_A \times Q_B, q_0 = (q_{A0}, q_{B0})$,
2. $\delta((r,s),a) = \{(r',s') \in Q \mid \delta_A(r,a) = r', \delta_B(s,a) = s'\}$, kde $(r,s) \in Q$, $a \in \Sigma$,
3. $F = \{(r,s) \in Q \mid r \in F_A \wedge s \in F_B\}$.

Tato metoda testu ekvivalence však vytváří mnohem více stavů než měli původní automaty, a to ať pouze v konstrukci průniku (kvadratický nárůst), nebo i v determinizaci (exponenciální nárůst). Proto se pokračovalo v hledání efektivnějších způsobů. Algoritmy využívající protiřetězce nebo kongruenci se tento problém snaží vyřešit determinizací během testu ekvivalence a tím vytváří jen stavy nezbytné pro test.

2.3 Algoritmus Hopcrofta a Karpa

Jeden ze způsobů staví na bisimulaci dvou stavů (definice 2.6).

Definice 2.6. [9] Je dán konečný automat $M = (Q, \Sigma, \delta, q_0, F)$. **Bisimulace** je binární relace R na Q taková, že pro každou dvojici prvků $p, q \in Q$ platí: jestliže $(p,q) \in R$ pak

i $\forall a \in \Sigma$ a $\forall p' \in Q$ platí: jestliže $p' \in \delta(p,a)$ pak $\exists q' \in Q$ takové, že $q' \in \delta(q,a)$ a $(p',q') \in R$.

ii $\forall a \in \Sigma$ a $\forall q' \in Q$ platí: jestliže $q' \in \delta(q,a)$ pak $\exists p' \in Q$ takové, že $p' \in \delta(p,a)$ a $(p',q') \in R$.

iii $q \in F \iff p \in F$

Jinými slovy, jsou-li dva stavy bisimilární, pak dokáží dorovnat svoje tahy a jsou oba koncové nebo oba nekoncové. Maximální bisimulaci nazýváme bisimilarita a je to relace ekvivalence. Tyto znalosti můžeme použít v testu inkluze dvou jazyků tak, že provedeme test bisimilarity počátečních stavů jejich automatů (Teorem 2.7).

Teorém 2.7. [10] Mějme dva DKA $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ a $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$. Jestliže platí q_{A0} je bisimilární s q_{B0}, pak $L(A) = L(B)$.

Nyní se dostáváme k algoritmu, který počítá bisimulaci (a tedy i ekvivalenci) dvou DKA (algoritmus 2.1). Na vstup dostane dva stavy $x, y \in Q$, jejichž bisimulaci počítá [10].

Následující algoritmus 2.1 tedy postupně prochází dvojice stavů, do kterých se bisimulací z počátečních stavů dostane a když tyto stavy ještě nejsou v relaci přidá je tam. Pokud algoritmus narazí na dvojici stavů, která by měla být přidána do relace, ale jeden ze stavů je koncový a druhý není, může skončit a na otázku zda jsou stavy bisimilární, vrátit odpověď "false".

Algoritmus je optimalizován použitím reflexivního, symetrického a tranzitivního uzávěru $e(R)$ relace R. Ten můžeme popsat následujícími pravidly:
pro $x, y, z \in Q$:

identita: $xRy \implies xe(R)y$,
reflexivita: $xe(R)x$,
symetrie: $xe(R)y \implies ye(R)x$,
tranzitivita: $xe(R)y \wedge ye(R)z \implies xe(R)z$.

Identita říká, že stavy, které sou v bisimulaci patří i do relace $e(R)$. Tedy pokud $(x,y) \in R$, nemusíme ji již znova testovat. Další bodem je reflexivita. Znamená, že každý stav je ekvivalentní sám se sebou. Nemusíme tedy simulovat dvakrát průchod z jednoho stavu. Vždy bisimuluje sám se sebou. Symetrie říká, že nezáleží, jestli počítáme bisimulaci dvou stavů x a y nebo y a x. Jestliže platí jedno, druhé platí také. Poslední tranzitivita popisuje přenositelnost bisimulace. Pokud je x a y v bisimulaci a zároveň jsou v bisimulaci i y a z, potom spolu x a z také bisimulují.

2.4 Algoritmus Bonchiho a Pouse

F. Bonchi a D. Pous tento algoritmus ještě upravili pro NKA (algoritmus 2.2). Jejich algoritmus ve skutečnosti testuje ekvivalenci determinizovaných NKA a jejich determinizaci počítá za běhu. Díky tomu generuje jenom relevantní stavy a nedochází k přílišné

Algoritmus 2.1: HK(x,y) [10]

```
1  R = ∅;
2  todo = {(x,y)};
3  while todo ≠ ∅ do
4      todo = todo − {(x′,y′)};
5      if (x′,y′) ∈ e(R) then
6          continue;
7      else
8          if x′ ∈ F ⟺ y′ ∈ F then
9              return false;
10         else
11             foreach a ∈ Σ do
               todo = todo ∪ {(r,s) | r ∈
               δ(x′,a), s ∈ δ(y′,a)} ;
12             R = R ∪ {(x′,y′)};
13         end
14     end
15 end
16 return true;
```

expanzi stavového prostoru. Ve svém návrhu označili velkými písmeny X a Y stavy determinizovaného NKA. F' je potom množina koncových stavů determinizovaného automatu a δ' je funkce determinizovaných přechodů (jako v algoritmu 2.4 determinizace). Dále použili funkci $c(R)$ [10]. Kongruenčnímu uzávěru $c(R)$ se budeme věnovat níže.

Algoritmus 2.2: BP(X,Y) [10]

```
1  R = ∅;
2  todo = {(X,Y)};
3  while todo ≠ ∅ do
4      todo = todo − {(X′,Y′)};
5      if (X′,Y′) ∈ c(R) then
6          continue;
7      else
8          if X′ ∈ F′ ⟺ Y′ ∈ F′ then
9              return false;
10         else
11             foreach a ∈ Σ do todo =
               todo ∪ {(S,T) | S ∈ δ′(X′,a),
               T ∈ δ′(Y′,a)} ;
12             R = R ∪ {(X′,Y′)};
13         end
14     end
15 end
16 return true;
```

Kongruenční uzávěr c(R) relace R je nejmenší relace ekvivalence, která je kongruencí s ohledem na operaci sjednocení. [10] Je to tedy sjednocení identity,

reflexivního uzávěru, symetrického uzávěru, tranzitivního uzávěru a nového pravidla:

$$X_1 c(R) Y_1 \wedge X_2 c(R) Y_2 \implies (X_1 \cup X_2) c(R) (Y_1 \cup Y_2).$$

Jak jsem již naznačil, tento algoritmus lze aplikovat na problém ekvivalence dvou automatů provedením bisimulace jejich počátečních stavů. Na vstup algoritmu bychom dali počáteční stavy (q_0, r_0) dvou NKA. Proběhla by jejich zjednodušená bisimulace pomocí kongruenčního uzávěru a dostali bychom výsledek. Jestliže tyto automaty nebudou ekvivalentní, algoritmus skončí při prvním nesouladu a nebude muset běžet dál. V opačném případě však také nebude muset procházet všechny stavy díky tomuto kongruenčnímu uzávěru a i v tomto případě tedy ušetří.

3. Stromové automaty

Stromové automaty (definice 3.1) jsou struktury podobné slovním automatům, které na vstup dostávají stromy místo slov. Stromový automat je tedy rozšířením slovního automatu, protože slovo je speciálním případem stromu ve kterém má každý uzel pouze jednoho potomka.

Definice 3.1. [11, 12] **Nedeterministický konečný stromový automat** (dále NKSA) je čtveřice $M = (Q, \Sigma, \Delta, F)$, kde Q je konečná množina stavů, $F \subseteq Q$ je množina koncových stavů, Σ je vstupní abeceda ohodnocena funkcí $arita : \Sigma \to \mathbb{N}$ a $\Delta : Q^* \times \Sigma \to \mathscr{P}(Q)$ je množina přechodových pravidel. Jestliže $q \in \Delta(q_1, \ldots, q_n, f)$, potom vždy platí, že $arita(f) = n$

Ohodnocená abeceda [12] je množina symbolů doplněna o funkci arity, která každému symbolu přiřadí přirozené číslo n. Přechodová pravidla s aritou nula se nazývají listová pravidla a zapisují se $\Delta(a) = q$. Námi definovaný automat je tzv. bottom-up (česky zdola-nahoru), protože přechodová pravidla definují přechod od potomků k rodičovskému uzlu. Existují i automaty top-down (česky shora-dolů), jejichž pravidla popisují přechody od rodičovských uzlů k potomkům. Tyto dva způsoby popisu jsou ekvivalentní.

Definujeme si tedy strom, co je to běh automatu na stromu a kdy je strom automatem přijímán.

Definice 3.2. [13] Definujeme *uzel* jako sekvenci prvků $\mathbb{N}$, kde ε je prázdná sekvence. Pro uzel $v \in \mathbb{N}$, definujeme i-tého potomka uzlu v jako vi, kde $i \in \mathbb{N}$.

Je dána ohodnocená abeceda Σ, **strom** nad abecedou Σ je definován jako parciální zobrazení $t : \mathbb{N}^* \to \Sigma$ takové, že pro každé $v \in \mathbb{N}^*$ a $i \in \mathbb{N}$ platí, jestliže $vi \in dom(t)$ pak $v \in dom(t)$ a $\text{arita}(t(v)) \leq i$.

Definice 3.3. [13] **Běh** automatu A na stromu t je parciální zobrazení $\pi : \mathbb{N}^* \to Q$ takové, že $v \in dom(\pi)$ právě tehdy když buď $v \in dom(t)$ nebo $v = v'i$, kde $v' \in dom(t)$ a $i \leq \text{arita}(t(v'))$.

Jazyk stromového automatu $L(A)$ (definice 3.4) je množina všech stromů T_Σ nad abecedou Σ, které tento automat přijímá.

Definice 3.4. Jazyk přijímaný stromovým automatem A je značen $L(A)$ a definován:
$L(A) = \{t \in T_\Sigma \mid q \in \Delta(t) \text{ a } q \in F\}$, kde $\Delta(t) = \{q \in Q \mid \exists \text{ běh } \pi \text{ na } t \text{ takový, že } \pi(\varepsilon) = q\}$

3.1 Determinizace

Stejně jako slovní automaty, i stromové automaty můžeme determinizovat a vytvořit tak deterministický stromový automat (definice 3.5).

Definice 3.5. [11] **Deterministický konečný stromový automat** (dále DKSA) je stromový automat $A = (Q, \Sigma, \Delta, F)$, takový, že pro všechna $a \in \Sigma$ a pro všechna $q_1, \ldots, q_n \in Q$, kde $\text{arita}(a) = n$ platí $|\Delta(q_1, \ldots, q_n, a)| \leq 1$.

Algoritmus determinizace 3.1 nazývaný podmnožinová konstrukce vytváří jednotlivé stavy DKSA jako podmnožiny množiny stavů NKSA.

Algoritmus 3.1: determinizace stromového automatu [11]
input : NKSA $M = (Q, \Sigma, \Delta, F)$
output : DKSA $M = (Q', \Sigma, \Delta', F')$
$Q' = \emptyset$;
$\Delta' = \emptyset$;
repeat
  $Q' = Q' \cup \{s\}$;
  $\Delta' = \Delta' \cup \{(s_1, \ldots, s_n, a) = s\}$;
  Kde:
  $a \in \Sigma$;
  $s_1, \ldots, s_n \in Q'$;
  $s = \{q \in Q | \exists q_1 \in s_1, \ldots, q_n \in s_n,$
  $q \in \Delta(q_1, \ldots, q_n, a)\}$;
until *není možné přidat žádné pravidlo do* Δ';
$F' = \{s \in Q' | s \cap F \neq \emptyset\}$;

4. Ekvivalence stromových automatů

Dostáváme se k cíli naší práce. Navrhneme postup, který by vycházel z algoritmu Bonchiho Pouse a zjišťoval by ekvivalenci stromových automatů. Nejdříve definujeme bisimulaci pro stromové automaty 4.1, která je ekvivalentní jazykové ekvivalenci. Potom vytvoříme algoritmus pro výpočet této relace pro DKSA. Nakonec ho spojíme s determinizací za běhu a dostaneme algoritmus pro testování ekvivalence jazyků NKSA.

Definice 4.1. Mějme stromové automaty $M = (Q_M, \Sigma, \Delta_M, F_M)$ a $N = (Q_N, \Sigma, \Delta_N, F_N)$. R je binární relace na $Q_M \times Q_N$ pro kterou platí:

i jestliže $\exists q_1, \ldots, q_n, q' \in Q_M \wedge q' \in \Delta_M(q_1, \ldots, q_n, a) \wedge \forall i \in \langle 1,n \rangle$, potom $\exists r_1, \ldots, r_n, r' \in Q_N \wedge r' \in \Delta_N(r_1, \ldots, r_n, a)$, kde $(q', r') \in R$.

ii pro $\forall q \in Q_M$, takové že $q \in \Delta_M(a)$ a pro $\forall r \in Q_N$, takové že $r \in \Delta_N(a)$, kde $a \in \Sigma$ platí $(q,r) \in R$.

iii $q \in F_M \iff r \in F_N$

Pro tuto relaci tedy předpokládáme, že všechny iniciální stavy (stavy do kterých vede přechod s aritou 0) jsou bisimilární. Dáváme do relace postupně všechny dvojice stavů do kterých se automaty dostanou. Jsou-li oba stavy koncové nebo oba nekoncové, dané dva automaty jsou ekvivalentní.

Teorém 4.2. Mějme dva DKSA $A = (Q_A, \Sigma, \Delta_A, F_A)$ a $B = (Q_B, \Sigma, \Delta_B, F_B)$. $L(A) = L(B)$ právě tehdy když

$$\forall a \in \Sigma : q \in \Delta_A(a) \iff (r \in \Delta_B(a) \Rightarrow qRr).$$

Náš nový algoritmus 4.1 pracuje s množinami *todo* kde uchovává dvojice, které už jsou v relaci ale ještě je nezpracoval a množinu *done* kam bude přesouvat zpracované dvojice z *todo*. Na začátku vloží do *todo* dvojice listových stavů. V dalším kroku jednu dvojici zpracuje. Vloží ji do *done* a podívá se jestli může použít nějaká přechodová pravidla, která obsahují prvek z dvojice a zároveň všechny ostatní stavy v tomto pravidle už jsou v relaci (tedy sjednocení *todo* a *done*). Když najde takovou dvojici do které lze přejít, zkontroluje jestli jsou oba stavy koncové (resp. nekoncové). Při záporném výsledku skončí a oznámí, že jazyky těchto automatů nejsou ekvivalentní. V opačném případě přidá novou dvojici na konec seznamu *todo*, tedy pouze v případě, že ještě dvojice v seznamech není. Jakmile zpracuje všechny dvojice v *todo*, jazyky jsou ekvivalentní. Pro optimalizaci jsme použili kongruenční uzávěr. Místo testování páru stavů v seznamech se testuje zda jsou v kongruenčním uzávěru těchto seznamů.

Algoritmus 4.1: Ekvivalence stromových automatů

input : DKSA $M = (Q_M, \Sigma, \Delta_M, F_M)$, DKSA $N = (Q_N, \Sigma, \Delta_N, F_N)$

```
1  done = ∅;
2  todo = {(x,y) | x ∈ Q_M, Δ_M(a) = x, y ∈ Q_N, Δ_N(a) = y, pro všechna a ∈ Σ};
3  while todo ≠ ∅ do
4      todo = todo − {(x',y')};
5      done = done ∪ {(x',y')};
6      foreach a ∈ Σ do
7          if ∃q' ∈ Δ_M(q_1,...,x',...,q_n,a),
           r' ∈ Δ_N(r_1,...,y',...,r_n,a) takové, že
           (q_i,r_i) ∈ e(done ∪ todo) pro každé
           i ∈ ⟨1,n⟩ then
8              if q' ∈ F_M ⇔ r' ∈ F_N then
9                  return false;
10             end
11             if (q',r') ∉ e(done ∪ todo) then
12                 todo = todo ∪ {(q',r')};
13             end
14         end
15     end
16 end
17 return true;
```

Nový algoritmus 4.1 počítá pouze s DKSA. Lze ho upravit obdobně jako BP spojením stromové bisimulace s determinizací za běhu. Nový algoritmus 4.2 bude testovat ekvivalenci i pro nedeterministické stromové automaty. Velká písmena X a Y značí stavy determinizované podmnožinovou konstrukcí.

5. Závěr

V našem článku jsme přišli s algoritmem pro testování inkluze jazyků stromových automatů, který zobecňuje algoritmus Bonchiho a Pouse. Tento nový algoritmus, kombinuje test bisimulace s determinizací za běhu. Bisimulace stromových automatů a algoritmus, který ji testuje, jsou také přínosy tohoto článku. V budoucí práci se budeme věnovat formálnímu důkazu správnosti algoritmu a experimentálnímu porovnání našeho algoritmu s již existujícími metodami, zejména s algoritmem založeným na protiřetězcích [12].

Poděkování

Děkuji p. Holíkovi, za to, že mi byl průvodcem v semestrálním studiu, že mi pomáhal a vysvětloval vše co bylo potřeba a že to všechno proběhlo v příjemné a přátelské atmosféře.

Algoritmus 4.2: Ekvivalence nedeterministických stromových automatů

input : NKSA $M = (Q_M, \Sigma, \Delta_M, F_M)$, NKSA $N = (Q_N, \Sigma, \Delta_N, F_N)$

```
1  done = ∅;
2  todo = {(X,Y) | X ∈ ⋃_{x∈Q_M}, x ∈ Δ_M(a),
   Y ∈ ⋃_{y∈Q_N}, y ∈ Δ_N(a), pro všechna a ∈ Σ};
3  while todo ≠ ∅ do
4      todo = todo − {(X',Y')};
5      done = done ∪ {(X',Y')};
6      foreach a ∈ Σ do
7          foreach q' ∈ Δ_M(q_1,...,x',...,q_n,a),
           r' ∈ Δ_N(r_1,...,y',...,r_n,a) takové, že
           x' ∈ X', y' ∈ Y' a
           (q_i,r_i) ∈ e(done ∪ todo) pro každé
           i ∈ ⟨1,n⟩ do
8              if q' ∈ F_M ⇔ r' ∈ F_N then
9                  return false;
10             end
11             Q' = ⋃_{q'};
12             R' = ⋃_{r'};
13         end
14         if (Q',R') ∉ e(done ∪ todo) then
15             todo = todo ∪ {(q',r')};
16         end
17     end
18 end
19 return true;
```

Literatura

[1] Haruo Hosoya, Jerome Vouillon, and Benjamin C. Pierce. Regular expression types for xml. In *ACM TOPLAS*, volume 27. January 2005.

[2] A. Bouajjani, P. Habermehl, A. Rogalewicz, , and T. Vojnar. Abstract regular tree model checking of complex dynamic data structures. In *SAS'06*, LNCS. Springer, 2006.

[3] P. Habermehl, L. Holík, A. Rogalewicz, J. Šimáček, and T. Vojnar. Forest automata for verification of heap manipulation. In *CAV 2011*. Snowbird, UT, USA, 2011.

[4] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *FOCS'72*. IEEE, 1972.

[5] P. A. Abdulla, Y. F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *TACAS'10*, volume 6015 of *LNCS*. Springer, 2010.

[6] L. Holík. Simulations and antichains for efficient handling of finite automata. disertační práce, Brno, FIT VUT v Brně, 2010.

[7] Michael Sipser. *Introduction to the Theory of Computation*. Thomson, second edition, 2006. Massachusetts Institute of Technology. ISBN 0-534-95097-3.

[8] A. Meduna and R. Lukáš. Formální jazyky a překladače. Opora IFJ. Verze: 1.2006. Revize 2009-2015. FIT VUT v Brně, 2006.

[9] Bisimulace[online], wikipedie, otevřená encyklopedie, 10. 3. 2013.

[10] F. Bonchi and D. Pous. Hopcroft and karp's algorithm for non-deterministic finite automata. Available on: `<https://hal.archives-ouvertes.fr/hal-00639716v2>`, November 2011. Technical Report hal-00639716v2.

[11] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `<http://www.grappa.univ-lille3.fr/tata>`, 2007. release October, 12th 2007.

[12] P.A. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar. Composed bisimulation for tree automata, 2008. Technical Report FIT-TR-2008-004, FIT BUT, Brno, Czech Republic.

[13] R. Almeida, L. Holík, and R. Mayr. Reduction of nondeterministic tree automata. In *TACAS'16*, LNCS. Springer, 2016.