VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KONGRUENCE PRO STROMOVÉ AUTOMATY

CONGRUATION ON TREE AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR ŽUFAN**
AUTHOR

VEDOUCÍ PRÁCE **Mgr. LUKÁŠ HOLÍK, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19744/2016/xzufan00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Žufan Petr**
Obor: Informační technologie
Téma: **Kongrunece pro stromové automaty**
Congruences for Tree Automata
Kategorie: Teoretická informatika

Pokyny:

1. Nastudujte moderní algoritmy pro test inkluze jazyků konečných automatů, zejména algoritmus založený na výpočtu bisimulace optimalizovaný technikou kongruenčního uzávěru. Nastudujte základní teorii stromových automatů
2. Navrhněte zobecnění kongruenčnícho algoritmu pro stromové automaty.
3. Dokažte korektnost nového algoritmu.
4. Implementujte algoritmus a porovnejte jeho výkonnost s existujícími algoritmy, zejména s algoritmy založenými na protiřetězcích. Použijte vhodná testovací data z literatury, z verifikačních úloh, případně náhodně generované automaty.

Literatura:

- H. Comon, M. Dauchet, R. Gilleron, C. Loding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree Automata Techniques and Applications, Available on: http://www.grappa.univ-lille3.fr/tata (2007)
- Filippo Bonchi and Damien Pous. 2015. Hacking nondeterminism with induction and coinduction. Commun. ACM 58, 2 (January 2015), pp. 87-95.
- P.A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr and T. Vojnar. When Simulation Meets Antichains. In Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 6015, pp. 158-174. Springer Berlin Heidelberg (2010)

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2, část 2 nebo 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Holík Lukáš, Mgr., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tento článek pojednává o testování ekvivalence stromových automatů (TA). 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. Z tohoto hlediska je lepší než jiné metody pro tento problém.

Abstract

This thesis discusses testing of tree automata (TA) equivalence. We propose a new algorithm based on Bonchi Pouse's algorithm of word automata. The new algortihm combines bisimulation and determinization on the fly. Using an optimalization based on congruence closure, we try to avoid extreme expansion of state space. From this point of view, the new algorithm is better than others.

Klíčová slova

automat, stromový automat, inkluze jazyků, ekvivalence jazyků, kongruence

Keywords

state machine, tree automata, language inclusion, language equivalence, congruence

Citace

ŽUFAN, Petr. *Kongruence pro stromové automaty.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Mgr. Lukáš Holík, Ph.D.

Kongruence pro stromové automaty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doktora Lukáše Holíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Žufan
17. května 2017

Poděkování

Děkuji doktoru Holíkovi, že mi byl vedoucím, průvodcem a oporou v tvorbě baklářské práce, že mi pomáhal a vysvětloval vše, co jsem potřeboval, že mi připomínal blížící se deadline, když mé soustředění směřovalo jinými směry a že to všechno proběhlo v příjemné a přátelské atmosféře.

Obsah

Kapitola 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 [12], zpracování přirozených jazyků, model checking [7], formální verifikace systémů [10]. Dá se říci, že stromové automaty jsou využitelné všude tam, kde se zpracovávají regulární množiny stromů. Tato práce se věnuje problému zjištění ekvivalence dvou jazyků definovaných stromovými automaty. Protože jsou stromové automaty zobecněním slovních, budeme vycházet z algoritmů pro slovní automaty.

Zjištění ekvivalence dvou slovních automatů je PSPACE-úplný problém [16]. Pro tento problém existuje několik algoritmů. Nejzákladnější z nich vychází z komplementace. Pro tu je potřeba konstrukce podmnožinové determinizace, průniku a testu prázdnosti. Zejména kvůli determinizaci se projevuje velká teoretická složitost problému. V poslední době byly vyvinuty algoritmy, které se snaží vyhnout velké teoretické složitosti nejhorších případů. Tyto techniky determinizují automaty za běhu zároveň s testem prázdnosti. Při tom se vyhýbají vytváření stavů, které nejsou pro výsledek testu ekvivalence relevantní. První z nich je založena na protiřetězcích [3]. Ty využívají subsumpce stavů produktového automatu. Stavy, které mají větší jazyk než jiné dostupné stavy, není potřeba zkoumat [9]. Později byly vyvinuty algoritmy založené na kongruenci, kterým se v této práci budeme věnovat. Ty intuitivně souběžně simulují průchod automaty a sledují, zda se shodují. Při tom mohou vynechat ty páry stavů, které se zcela jistě musí shodovat, jestliže se shodovali všechny dříve navštívené páry. Toho se dosahuje výpočtem kongruenčního uzávěru. Příkladem této metody je algoritmus Hopcrofta a Karpa (dále jen HK) [5] pro deterministické automaty a algoritmus Bonchiho a Pouse (dále jen BP) [5], který tento algoritmus spojil s determinizací za běhu a rozšířil jeho použití na nedeterministické slovní automaty. Ukázalo se, že pro některé automaty fungují kongruenční algoritmy lépe než protiřetězcové. Jde především o automaty, které sdílejí část stavového prostoru [6].

Stromové automaty jsou principiálně velmi podobné slovním automatům. Rozdílem je, že na vstup dostávají stromy namísto slov. Protože ve stromové struktuře mají jednotlivé uzly několik potomků, přechody stromového automatu definují přechod mezi jedním stavem a uspořádanou n-ticí stavů, reprezentující právě rodičovský uzel a jeho potomky. Slovo pak může být prezentováno jako unární strom. Proto jsou stromové automaty zobecněním slovních automatů. Algoritmy, které používají, jsou většinou poměrně přímočarým rozšířením algoritmů pro slovní automaty, ale jejich složitost je často větší. Ekvivalence stromových automatů je EXPTIME-úplný problém. Naivní algoritmus založený na komplementu a algoritmus protiřetězců již existuje [3]. Cílem této práce je zobecnit kongruenční algoritmus na stromové automaty. Pro slovní automaty v některých případech fungoval lépe než pro-

tiřetězcový algoritmus [5]. Na stromových automatech by tedy také mohl přinášet lepší výsledky.

Naše práce přináší nový algoritmus testu ekvivalence nedeterministických stromových automatů, definuje relaci bisimulace na stromových automatech, ze které tento algoritmus vychází. Algoritmus provádí determinizaci za běhu spolu s testem prázdnosti průniku jazyků dvou stromových automatů. Navíc počítá optimalizaci kongruenčním uzávěrem. Tyto techniky značně pomáhají redukovat stavový prostor vznikající při naivním přístupu. Nakonec je tento algoritmus porovnán se zbylými jmenovanými přístupy a ukazuje se, že i pro stromové automaty je v některých případech výhodnější než algoritmus založený na protiřetězcích [6].

V druhé kapitole se podrobněji podíváme na slovní automaty. Definujeme je, ukážeme rozdíl mezi deterministickým a nedeterministickým automatem a předvedeme algoritmus determinizace nazývaný podmnožinová konstrukce. Dále se podíváme na algoritmus založený na komplementaci a na algoritmy Hopcrofta a Karpa a Bonchiho a Pouse. V třetí kapitole se budeme věnovat stromovým automatům, definujeme je a opět ukážeme determinizaci podmnožinovou konstrukcí. Čtvrtá kapitola je pak o samotném algoritmu testu ekvivalence jazyků stromových automatů založeném na kongruenci. Zde je definována bisimulace na stromových automatech a popsán nový algoritmus pro deterministické stromové automaty. Poté je tento algoritmus spojen s determinizací za běhu pro nedeterministické stromové automaty. V páté kapitole jsou pak popsány konkrétní způsoby implementace a možnosti optimalizace této implementace. A nakonec, šestá kapitola shrnuje výsledky práce, porovnává nový algortimus s ostatními přístupy a přináší rozhodnutí, zda a kdy je lepší dát přednost tomuto novému algoritmu před ostatními.

Kapitola 2

Slovní automaty

Abychom správně porozuměli novému algoritmu, začneme nejprve od základů. Řekneme, co je to abeceda a slovo, následně definujeme konečný automat a jazyk přijímaný tímto automatem. Vysvětlíme rozdíly mezi deterministickým a nedeterministickým konečným automatem a ukážeme, jak je mezi sebou převádět. Podíváme se na různé způsoby porovnání jazyků definovaných automaty, mezi které patří zejména algoritmy HK a BP, ze kterých budeme v dalších kapitolách vycházet.

2.1 Abeceda, slovo, jazyk

Definice 2.1.1. [15] **Abeceda** je jakákoliv neprázdná konečná množina. Prvkům této množiny se říká symboly.

Definice 2.1.2. [15] Nechť Σ je abeceda. **Slovo** nad abecedou Σ je konečná posloupnost symbolů z této abecedy. Prázdná posloupnost se nazývá prázdné slovo a označuje se ε.

Definice 2.1.3. [17] Nechť Σ^* je množina všech slov nad abecedou Σ. **Formální jazyk** L je množina slov, pro kterou platí $L \subseteq \Sigma^*$.

Jako příklad můžeme uvést abecedu $\Sigma = \{a, b\}$. Slova nad touto abecedou jsou například "aab" nebo "baba". Mezi symboly ve slově se z praktických důvodů nepíše čárka. Jazyk pak můžeme definovat výčtem všech jeho slov ($L = \{a, b, aba, bbaa, abbaaaab\}$) nebo slovně (L je jazyk obsahující všechna slova nad abecedou Σ, která mají stejný počet symbolu a a b).

I přesto, že se jazyk skládá pouze z konečných slov, sám o sobě konečný být nemusí. Definovat jej výčtem všech jeho slov tedy není vždy možné a slovní popis je mnohdy velmi složitý a nejednoznačný. Pro definici takového jazyka se proto používá gramatika [17] nebo automat. A právě automaty se budeme zabývat dále.

2.2 Konečný automat

Konečný automat (definice 2.2.1) je matematický model jednoduchého počítače zpracovávajícího řetězce symbolů vstupní abecedy (slova). Slova jsou automatem buď přijímána nebo ne. Množina přijímaných slov se nazývá jazyk přijímaný automatem. Základní typ automatu je nedeterministický konečný automat (dále jen NFA).

Definice 2.2.1. [17] **Nedeterministický konečný automat** je pětice $M = (Q, \Sigma, \delta, q_0, F)$, kde

- Q je konečná množina stavů,
- Σ je konečná množina vstupních symbolů,
- $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ je přechodová funkce,
- $q_0 \in Q$ je počáteční stav a
- $F \subseteq Q$ je množina koncových stavů.

V definici použité $\mathcal{P}(Q)$ značí potenční množinu množiny Q. Tedy množinu všech jejích podmnožin. Σ_ε označuje množinu symbolů doplněnou o prázdný řetězec. Formálně $\Sigma \cup \{\varepsilon\}$. Použití ε v přechodu na místě vstupního symbolu znamená, že automat může přejít do následujícího stavu bez přečtení vstupního symbolu. Tyto přechody budeme nazývat ε-přechody.

Pro rozhodnutí, zda jsou slova automatem přijímána, je spuštěn takzvaný běh automatu nad daným slovem (definice 2.2.2). Ten probíhá následovně: Na začátku je automat v počátečním stavu q_0. Následně čte postupně symboly vstupního slova. Při přečtení prvního symbolu a_1 přejde do stavu q_1 podle pravidla $q_1 \in \delta(q_0, a_1)$. Tak postupuje dále, dokud nepřečte všechny symboly vstupního slova. Nyní se automat nachází ve stavu q_n Jestliže je stav q_n koncový, tedy $q_n \in F$, pak je slovo automatem přijato. V opačném případě, kdy $q_n \notin F$, slovo přijímáno není. Množina všech slov přijímaných automatem se nazývá jazyk (definice 2.2.3) definovaný tímto automatem.

Definice 2.2.2. [17] **Běh** automatu M nad slovem $w = a_1 a_2 \dots a_n \in \Sigma^*$ je posloupnost stavů q_0, q_1, $\dots q_n$, kde q_0 je počáteční stav automatu M a $\forall q_i \in Q$, kde $0 < i \leq n$, platí, že $q_i \in \delta(q_{i-1}, a_i)$.

Definice 2.2.3. [15] **Jazyk** definovaný automatem M je označen $L(M)$ a definován: L je množina slov $a_1 a_2 \dots a_n$, takových, že existují stavy $q_1, \dots, q_n$, takové, že pro $1 \leq i \leq n$ platí $q_i \in \delta(q_{i-1}, a_i)$ a $q_n \in F$.

Dalším typem automatu je deterministický konečný automat (dále jen DFA, definice 2.2.4). DFA je vlastně pouze speciální případ NFA. Práce s NFA je složitá, avšak jsou přehlednější a někdy lze díky nim ušetřit prostor i čas. Každý NFA lze převést na ekvivalentní DFA a obráceně. To lze dokázat například algoritmem nazvaným determinizace.

Definice 2.2.4. [17] **Deterministický konečný automat** je konečný automat $M = (Q, \Sigma, \delta, q_0, F)$, takový že:

$$\forall q \in Q \text{ a } \forall a \in \Sigma \text{ platí, že } |\delta(q, a)| \leq 1.$$

Příklad 2.2.1. Na ukázku uvedeme DFA a NFA definující stejný jazyk $L = \{aa, aaa, aaaaa\}$. Samozřejmě existuje více automatů, který tento jazyk definují. Uvedeme příklad nedeterministického automatu M_N a deterministického M_D.

$M_N = (Q, \Sigma, \delta, q_0, F)$, kde
$Q = \{q_0, q_1, q_2\}$,
$\Sigma = \{a\}$,
$\delta = \{$
$\quad (q_0, a) \mapsto \{q_1\}$,
$\quad (q_1, a) \mapsto \{q_0, q2\}\}$,
q_0,
$F = \{q_2\})$

$M_D = (Q, \Sigma, \delta, q_0, F)$, kde
$Q = \{q_0, q_1, q_2\}$,
$\Sigma = \{a\}$,
$\delta = \{$
$\quad (q_0, a) \mapsto q_1$,
$\quad (q_1, a) \mapsto q_2$,
$\quad (q_2, a) \mapsto q_0\}$,
q_0,
$F = \{q_2\})$

2.3 Determinizace

Některé operace nad automaty nejsou pro NFA možné, je potřeba je determinizovat. K těmto operacím patří i test ekvivalence. Všechny algoritmy testu ekvivalence jazyků dvou automatů nějakým způsobem determinizaci provádí. Některé determinizují automat před samotným testováním, jiné zapojí determinizaci přímo do algoritmu a snaží se tak zmenšit stavový prostor tímto procesem generovaný. Z definice DFA (2.2.4) vyčteme, že u DFA na rozdíl od NFA, existuje nejvýše jeden stav, do kterého se dostaneme z daného stavu a při daném symbolu na vstupu. A právě proto jsou DFA jednodušší na práci s nimi.

Při běhu NFA nad slovem přechází automat z jednoho stavu do množiny několika stavů. Při zpracování dalšího symbolu ze slova pak použije přechody ze všech těchto aktuálních stavů. Následující množinu stavů získá sjednocením všech množin stavů, do kterých se pomocí těchto použitých přechodů dostane. Aby bylo slovo nedeterministickým automatem přijímáno, stačí aby alespoň jeden stav z množiny stavů, ve kterých automat skončí, byl koncový.

Po pochopení rozdílu mezi NFA a DFA se můžeme pustit do problému, jak determinizovat NFA. Popíšeme Algoritmus 2.3.1 nazývaný podmnožinová konstrukce.

Algoritmus 2.3.1. [17] Mějme nedeterministický konečný automat $M = (Q, \Sigma, \delta, q_0, F)$ přijímající nějaký jazyk L, sestrojíme deterministický konečný automat $M' = (Q', \Sigma', \delta', q_0', F')$ přijímající stejný jazyk L.

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

Dodatečně musíme nadefinovat ε-uzávěr $E(R)$, který jsme použili v algoritmu 2.3.1. Je to množina všech stavů dosažitelných ze stavu R bez přečtení vstupního symbolu. Toto je možné pomocí ε-přechodů popsaných výše. Formálně $E(R) = \{q \mid q$ je dosažitelné z R použitím 0 nebo více ε-přechodů$\}$. Do této množiny patří tedy i samotný stav R.

Poznámka. Všimněme si, že stavy původního nedeterministického automatu jsou značeny malými písmeny (q, r, ...), zatímco stavy determinizovaného automatu písmeny velkými (R, S, ...). Je to proto, že determinizované stavy jsou množiny původních stavů ($R = \{r_1, r2, \dots\}$).

Nově vzniklý DFA obsahuje 2^n stavů, oproti n stavům původního NFA. Mohli bychom DFA ještě minimalizovat odstraněním stavů nedostupných z počátečního stavu, odstraněním stavů nadbytečných, tedy těch, ze kterých nelze přejít do žádného z koncových stavů použitím libovolného počtu přechodů a odstraněním stavů ekvivalentních.

Příklad 2.3.1. Jako příklad determinizujeme automat M_N z příkladu 2.2.1. Vytvoříme automat M_d podle Algoritmu 2.3.1. Jak jsme naznačili, vzniklý automat lze minimalizovat. Potom bychom získali automat shodný s automatem M_D z příkladu 2.2.1. Minimalizace je ovšem další náročný algoritmus, a i potom může být výsledný automat o mnoho větší, než automat původní.

$M_d = (Q, \Sigma, \delta, q_0, F)$, kde
$Q = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$,
$\Sigma = \{a\}$,
$\delta = \{$
$(\emptyset, a) \mapsto \emptyset$,
$(\{q_0\}, a) \mapsto \{q_1\}$,
$(\{q_1\}, a) \mapsto \{q_0, q_2\}$,
$(\{q_2\}, a) \mapsto \emptyset$,
$(\{q_0, q_1\}, a) \mapsto \{q_0, q_1, q_2\}$,
$(\{q_0, q_2\}, a) \mapsto \{q_1\}$,
$(\{q_1, q_2\}, a) \mapsto \{q_0, q_2\}$,
$(\{q_0, q_1, q_2\}, a) \mapsto \{q_0, q_1, q_2\}\}$,
$\{q_0\}$,
$F = \{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\})$

2.4 Porovnání jazyků konečných automatů

Základní algoritmus pro porovnání dvou jazyků se počítá jako inkluze jazyků oběma směry: $L(A) = L(B) \iff L(A) \subseteq L(B) \wedge L(B) \subseteq L(A)$. Test inkluze je založen na implementaci jazykových operací komplementace a průniku nad automaty. Pokud tento průnik bude prázdná množina, jazyky inkludují. Jinak popsáno $L(A) \subseteq L(B) \iff L(A) \cap \overline{L(B)} = \emptyset$.

Při aplikaci tohoto algoritmu na NFA 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 $F' = Q \setminus F$. Stejně tak i konstrukce paralelního synchronního produktu dvou jazyků automatů je pro DFA realizovaná jednoduchým algoritmem (2.4.1).

Algoritmus 2.4.1. [17] Mějme dva konečné 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. $\delta((r, s), a) = \{(r', s') \in Q \mid r' \in \delta_A(r, a),\ s' \in \delta_B(s, a)\}$, 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. Mějme automaty M s m stavy a automat N s n stavy. V konstrukci průniku

automatu M s automatem N vzniká automat s $m \cdot n$ stavy, jde tedy o kvadratický nárůst. Při determinizaci automatu N vznikne automat s 2^n stavy. V tomto případě jde dokonce o exponenciální nárůst stavového prostoru. Pokud použijeme naivní implementaci této metody bez optimalizací, celkem vznikne $2^m \cdot 2^n$ stavů z původních $m+n$ stavů. Asymptotická složitost tohoto algoritmu je tedy $O(2^m \cdot 2^n)$. Proto se pokračovalo v hledání efektivnějších způsobů testu ekvivalence. Algoritmy využívající protiřetězce nebo kongruenci se tento problém snaží vyřešit mimo jiné determinizací během testu ekvivalence a tím vytváří pouze stavy nezbytné pro tento test.

2.5 Algoritmus Hopcrofta a Karpa

Jedním ze způsobů je algoritmus Hopcrofta a Karpa [5]. Ten je založen na bisimulaci dvou stavů (definice 2.5.1).

Definice 2.5.1. [6] 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, že 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 jsou oba koncové nebo oba nekoncové a dokáží dorovnat svoje tahy. To znamená, že i všechny dvojice stavů, do kterých lze z této dvojice přejít jsou bisimilární. Maximální bisimulaci nazýváme *bisimilarita* a značíme $\sim$. Je to relace ekvivalence. Tyto znalosti můžeme použít v testu inkluze dvou jazyků tak, že provedeme test bisimilarity počátečních stavů jejich automatů (Teorém 2.5.2).

Teorém 2.5.2. [5] Mějme dva DFA $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ a $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$. Jestliže platí $q_{A0} \sim q_{B0}$, pak $L(A) = L(B)$.

Nyní se dostáváme k algoritmu 2.5.1, který počítá bisimulaci (a tedy i ekvivalenci) dvou DFA. Na vstup dostane dva stavy $x, y \in Q$, jejichž bisimularitu testuje [5]. Aby platilo, že je tato dvojice bisimilární, musí být bisimilární i dvojice stavů, do kterých mohou automaty z této dvojice přejít. Pro nově vygenerované stavy pak platí totéž. Tímto způsobem se prochází automatem tak dlouho, dokud se nenajde protipříklad, nebo nejsou zkontrolovány všechny dvojice stavů.

Protože je bisimulace relací ekvivalence, můžeme algoritmus optimalizovat použitím ekvivalenčního uzávěru $e(R)$ relace R. Pro ten platí, že je zároveň reflexivním, symetrickým a tranzitivním uzávěrem R. Můžeme ho popsat následujícími pravidly, kde $x, y, z \in Q$:

identita: $(x, y) \in R \implies (x, y) \in e(R)$,

reflexivita: $(x, x) \in e(R)$,

symetrie: $(x, y) \in e(R) \implies (y, x) \in e(R)$,

tranzitivita: $(x, y) \in e(R) \wedge (y, z) \in e(R) \implies (x, z) \in e(R)$.

Identita říká, že stavy, které jsou bisimilární 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. Symetrie říká, že nezáleží, jestli počítáme bisimulaci dvou stavů x a y nebo y a x. Jestliže platí jedno, druhé platí také. Poslední tranzitivita znamená, že pokud je bisimilární x s y a zároveň y se z, potom je x se z také bisimilární.

Algoritmus 2.5.1: HK(x,y) [5]

```
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;
```

2.6 Algoritmus Bonchiho a Pouse

F. Bonchi a D. Pous tento algoritmus ještě upravili pro NFA (algoritmus 2.6.1). Jejich algoritmus ve skutečnosti testuje ekvivalenci determinizovaných NFA, jejichž determinizaci počítá za běhu. To probíhá tak, že vytváří vždy pouze ty stavy, do kterých algoritmus během bisimulace přechází. Tímto způsobem redukuje množství vygenerovaných stavů. Ve svém návrhu označili velkými písmeny X a Y stavy determinizovaného NFA. F' je potom množina koncových stavů determinizovaného automatu a δ' je funkce determinizovaných přechodů (jako v algoritmu 2.3.1 determinizace). Dále použili kongruenční uzávěr $c(R)$ relace R [5], kterým dále redukují počet generovaných dvojic stavů.

2.6.1 Kongruenční uzávěr

Bonchi a Pous ve svém článku [5] využívají fakt, že relace bisimulace je kongruencí (definice 2.6.1). Protože je bisimulace kongruencí můžeme spočítat její kongruenční uzávěr. Potom víme, že všechny dvojice, které jsou v tomto uzávěru jsou také bisimilární. Kdyby nebyly, muselo by pro některé dvojice z původní relace také platit, že bisimilární nejsou. V algoritmu testu ekvivalence tedy můžeme tento kongruenční uzávěr počítat z množiny R již zkontrolovaných dvojic stavů. Když přidáváme novou dvojici, podíváme se, zda neleží v tomto uzávěru a až poté ji případně přidat do *todo*.

Definice 2.6.1. [6] Nechť $\sim$ je binární relace na množině X. Relaci $\sim$ nazveme **kongruence**, pokud je relací ekvivalence a pokud pro každou n-ární operaci O definovanou nad X

Algoritmus 2.6.1: BP(X,Y) [5]

```
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;
```

a pro všechna $a_1, \ldots, a_n$ a $b_1, \ldots, b_n \in X$ platí:

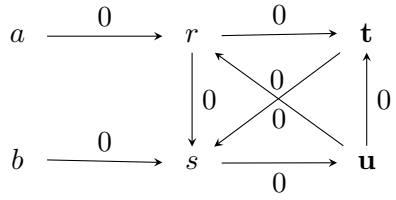
$$a_1 \sim b_1 \wedge \cdots \wedge a_n \sim b_n \implies O(a_1, \ldots, a_n) \sim O(b_1, \ldots, b_n)$$

Kongruenční uzávěr $c(R)$ relace bisimulace R je nejmenší relace ekvivalence, která obsahuje R a která je kongruencí vzhledem k operaci sjednocení [5]. Kongruenčí uzávěr nám tedy k optimalizaci ekvivalenčním uzávěrem přidává nové pravidlo:

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

Jak jsme již naznačili, tento algoritmus lze aplikovat na problém ekvivalence dvou automatů provedením testu bisimilarity jejich počátečních stavů. Na vstup algoritmu bychom dali počáteční stavy (q_0, r_0) dvou DFA, případně determinizovaných NFA. 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.

Příklad 2.6.1. Na ukázku provedeme test ekvivalence na dvou nedeterministických automatech A a B s překrývající se množinou stavů. Prvně ukážeme naivní přístup skládající se z podmnožinové konstrukce, průniku a testu prázdnosti, potom algoritmus Hopcrofta a Karpa, pro který prvně automaty také determinizujeme podmnožinovou konstrukcí a nakonec algoritmus Bonchiho a Pouse, který determinizuje za běhu. Automat A má stavy $Q_A = \{a, r, s, t, u\}$ s počátečním stavem a, automat B potom $Q_B = \{b, r, s, t, u\}$ a počáteční stav b. Přechody jsou znázorněny na následujícím schématu. Koncové stavy jsou zvýrazněny tučně.



Při komplementačním algoritmu vznikne prvně determinizací $2^5 = 32$ stavů jednoho automatu a stejný počet stavů druhého. Jeden z automatů se komplementuje a vytvoří se automat průniku automatů. Tak vznikne $32 \cdot 32 = 1024$ stavů. Nakonec provede test prázdnosti.

Algoritmus Hopcrofta a Karpa také prvně vytvoří 32 stavové deterministické automaty. Potom testuje bisimilaritu počátečních stavů. Tím testuje prázdnost průniku automatů za běhu a zároveň vytváří jen nezbytné dvojice stavů, ukládaných do relace R. Navíc počítá ekvivalenční uzávěr této relace, což také může způsobit generování méně dvojic stavů. Z uvedeného zadání se postupně vygeneruje tato relace $R = \{(\{a\}, \{b\}), (\{r\}, \{s\}), (\{st\}, \{u\}), (\{su\}, \{rt\}), (\{rtu\}, \{st\}), (\{rst\}, \{su\}), (\{stu\}, \{rtu\}), (\{rstu\}, \{rst\}), (\{rstu\}, \{stu\}), (\{rstu\}, \{rstu\})\}$.

Poslední je algoritmus Bonchiho a Pouse, který i determinizaci provádí za běhu. Tedy místo 32 stavů generuje jen stavy potřebné. Z těch pak opět vytváří dvojice do R. Postupně vytvoří tyto dvojice $R = \{(\{a\}, \{b\}), (\{r\}, \{s\}), (\{st\}, \{u\}), (\{su\}, \{rt\})\}$. Další dvojice by byla $(\{rtu\}, \{st\})$, ta už ale leží v kongruenčním uzávěru. Nemusíme ji tedy testovat a algoritmus může skončit. Počty vytvořených stavů jsou shrnuty v tabulce 2.1.

Tabulka 2.1: Srovnání algoritmů pro slovní automaty

Algoritmus	**Počet stavů**		
	původní automaty	**determinizace**	**průnik/bisimulace**
Komplementace	5 + 5	32 + 32	1024
Hopcroft a Karp	5 + 5	32 + 32	10
Bonchi a Pouse	5 + 5	8	4

Kapitola 3

Stromové automaty

V předchozí kapitole jsme definovali konečný automat (definice 2.2.1) zpracovávající slova. Budeme je tedy nazývat slovní automaty. V této kapitole si představíme stromové automaty (definice 3.0.5). Jsou to struktury podobné slovním automatům, které na vstup dostávají stromy (definice 3.0.4) místo slov. Stromový automat je tedy rozšířením slovního automatu, protože slovo je speciálním případem stromu, ve kterém má každý uzel pouze jednoho potomka.

Abychom plně porozuměli, jak stromový automat funguje, definujeme nejprve, co je to strom, poté stromový automat a běh automatu na stromu a nakonec jazyk stromového automatu.

Definice 3.0.2. [4] **Ohodnocená abeceda** Σ je množina symbolů doplněná o funkci $arita$: $\Sigma \to \mathbb{N}_0$.

Definice 3.0.3. [4] 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 $i \in \mathbb{N}$.

Definice 3.0.4. [4] Je dána ohodnocená abeceda Σ, **strom** nad abecedou Σ je definován jako parciální zobrazení $t : \mathbb{N}^* \to \Sigma$ takové, že pro každý uzel $v \in \mathbb{N}^*$ a $i \in \mathbb{N}$ platí, jestliže uzel $vi \in dom(t)$ pak $v \in dom(t)$ a $arita(t(v)) \leq i$.

Arita nám tedy říká, kolik potomků má uzel označený daným symbolem. Této znalosti využívají stromové automaty ve své přechodové funkci.

Definice 3.0.5. [8, 1] **Nedeterministický konečný stromový automat** (dále NTA) je čtveřice $M = (Q, \Sigma, \Delta, F)$, kde

- Q je konečná množina stavů,
- $F \subseteq Q$ je množina koncových stavů,
- Σ je ohodnocená vstupní abeceda a
- $\Delta : Q^* \times \Sigma \to \mathcal{P}(Q)$ je přechodová funkce. Jestliže $q \in \Delta(q_1, \ldots, q_n, f)$, potom vždy platí, že $arita(f) = n$.

Vidíme, že stromové automaty přechází z n-tice stavů do jednoho (v případě nedeterministického stromového automatu i do více). Přechodová pravidla s aritou symbolu rovnou nule 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.

Definice 3.0.6. [3] Nechť $A = (Q, \Sigma, \Delta, F)$ je stromový automat. **Běh** automatu A na stromu $t \in T_\Sigma$ je totální zobrazení $\pi : dom\,(t) \to Q$ takové, že pro všechny uzly $v \in dom(t)$, kde $arita(t(v)) = n$ a kde $q = \pi(v)$ platí: Když $q_i = \pi(vi)$ pro $1 \leq i \leq n$, potom $q \in \Delta(q_1, \ldots, q_n)(t(v))$.

Na počátku běhu není automat v žádném stavu a začíná číst listy vstupního stromu. Na každý list je aplikováno listové pravidlo z přechodové funkce a automat tak přejde do nějakého stavu. Po přečtení dalších listů je automat v několika stavech zároveň. Nyní může přečíst rodičovský uzel listových uzlů, které již přečetl, aplikovat příslušné přechodové pravidlo a ze stavů, do kterých se těmito listovými stavy dostal, přejde do jednoho následujícího stavu. Tímto způsobem pokračuje dál. Z n-listových stavů se nakonec po přečtení celého vstupního stromu dostane do jednoho stavu. Jestliže je tento stav koncový, pak je strom automatem přijímán, jestliže koncový není, strom automatem přijímán není.

Jazyk stromového automatu $L(M)$ (definice 3.0.7) je množina všech stromů T_Σ nad abecedou Σ, které tento automat přijímá. Automat definovaný shora-dolů by zpracovával vstupní strom obráceně - od kořene k listům.

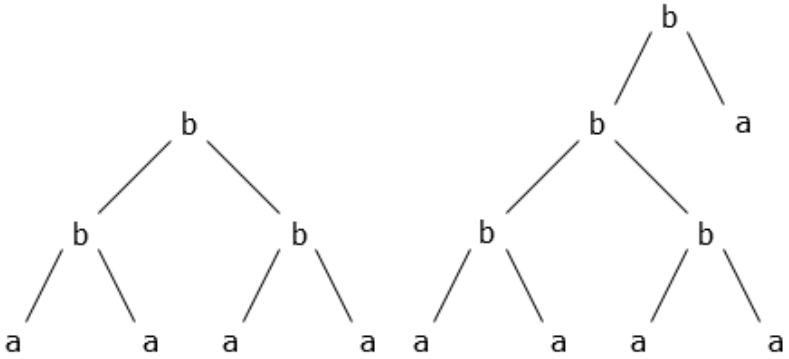
Definice 3.0.7. [11] **Jazyk** přijímaný stromovým automatem M je značen $L(M)$ a definován:

$L(M) = \{t \in T_\Sigma \mid q \in \Delta(t) \text{ a } q \in F\}$, kde $\Delta(t) = \{q \in Q \mid \exists \text{ běh } \pi \text{ na } t \text{ takový, že } \pi(\varepsilon) = q\}$

Příklad 3.0.2. Jako příklad uvedeme dva stromové automaty, které tentokrát nedefinují stejný jazyk, ale rozdíl deterministicity je na nich vidět také. Automat M_N je nedeterministický a automat M_D deterministický. Oba automaty mají symboly a a b s funkcí arity $arita(a) = 0$ a $arita(b) = 2$. Přechodové pravidlo $(a) \mapsto q_0$ je pak to takzvané listové pravidlo a q_0 listový stav. Oba automaty přijímají například strom 1 z obrázku 3.1 ale strom 2 je přijímán pouze automatem M_N. Při pozornějším prohlédnutí automatů si všimneme, že jazyk automatu M_D je podmnožina jazyku M_N.

$M_N = (Q, \Sigma, \Delta, F)$, kde
$Q = \{q_0, q_1, q_2\}$,
$\Sigma = \{a, b\}$,
$\Delta = \{$
$\quad (a) \mapsto \{q_0\}$,
$\quad (q_0, q_0, b) \mapsto \{q_1\}$,
$\quad (q_1, q_1, b) \mapsto \{q_1, q_2\}$,
$\quad (q_2, q_2, b) \mapsto \{q_2\}$,
$\quad (q_1, q_0, b) \mapsto \{q_2\}\}$,
$F = \{q_2\}$

$M_D = (Q, \Sigma, \Delta, F)$, kde
$Q = \{q_0, q_1, q_2\}$,
$\Sigma = \{a, b\}$,
$\Delta = \{$
$\quad (a) \mapsto q_0$,
$\quad (q_0, q_0, b) \mapsto q_1$,
$\quad (q_1, q_1, b) \mapsto q_2$,
$\quad (q_2, q_2, b) \mapsto q_2$,
$\quad (q_1, q_0, b) \mapsto q_2\}$,
$F = \{q_2\}$



Obrázek 3.1: Vlevo Strom 1, vpravo Strom 2

3.1 Determinizace

Stejně jako slovní automaty, i stromové automaty můžeme dělit na nedeterministické a deterministické. Deterministický konečný stromový automat (dále DTA, definice 3.1.1) je, analogicky se slovními automaty, speciální případ NTA, kde z dané n-tice stavů pod daným symbolem lze přejít nejvýše do jednoho stavu. Výhody a nevýhody NTA oproti DTA jsou principiálně stejné jako výhody a nevýhody NFA oproti DFA. Stejně tak lze každý NFA převést na DFA a obráceně. Jejich jazyky jsou pak ekvivalentní.

Definice 3.1.1. [8] **Deterministický konečný stromový automat** (dále DTA) je stromový automat $M = (Q, \Sigma, \Delta, F)$, takový, že

$\forall a \in \Sigma$, kde $arita(a) = n$ a pro všechny n-tice $(q_1, \ldots, q_n)$, kde $q_1 \ldots q_n \in Q$, platí:

$$|\Delta(q_1, \ldots, q_n, a)| \leq 1.$$

Algoritmus determinizace NFA 3.1.1 nazývaný podmnožinová konstrukce vytváří jednotlivé stavy DTA jako množiny stavů NTA. Vzniká 2^n stavů DTA oproti původním n stavům NTA. Opět jsou malými písmeny označeny stavy nedeterministického automatu, zatímco velká písmena značí stavy deterministického automatu. Pro použití v algoritmu nejprve definujeme funkci přechodovou funkci pro determinizované stavy:

Definice 3.1.2. Je dán DTA $M = (Q, \Sigma, \Delta, F)$. Pro $S_1, \ldots, S_n \in \mathcal{P}(Q)$ a pro $a \in \Sigma$, kde $arita(a) = n$ definujeme přechodovou funkci

$$\Delta(S_1, \ldots, S_n, a) = \bigcup_{(s_1,\ldots,s_n) \in S_1 \times \cdots \times S_n} \Delta(s_1, \ldots, s_n, a).$$

Algoritmus 3.1.1: determinizace stromového automatu [8]

input : NTA $M = (Q, \Sigma, \Delta, F)$
output: DTA $M = (Q', \Sigma, \Delta', F')$

1 $Q' = \emptyset$;
2 $\Delta' = \emptyset$;
3 **repeat**
4 $\quad \Delta' = \Delta' \cup \{(S_1, \ldots, S_n, a) \mapsto S\}$;
5 $\quad Q' = Q' \cup \{S\}$;
6 $\quad$ Kde:
7 $\quad a \in \Sigma$ a $arita\,(a) = n$, $S_1, \ldots, S_n \in Q'$, $S = \Delta\,(S_1, \ldots, S_n, a)\}$;
8 **until** *není možné přidat žádné pravidlo do* Δ';
9 $F' = \{S \in Q' | S \cap F \neq \emptyset\}$;

Příklad 3.1.1. Ukážeme determinizaci stromového automatu M_N z příkladu 3.0.2. Množina stavů nového automatu M_d je množina podmnožin původního automatu. Přechody se pak vytváří z původních přechodů tak, že se najdou všechny přechody ze stavů, které nové stavy obsahují a nový stav je pak množina všech stavů, do kterých tyto nalezené přechody vedou. Tedy když hledáme výsledný stav přechodu

$$(\{q_0, q_1\}, \{q_0, q_1\}, b) \mapsto ?$$

zajímají nás přechody původního automatu:

$$\begin{gathered}(q_0, q_0, b) \mapsto q_1 \\ (q_0, q_1, b) \mapsto \emptyset \\ (q_1, q_0, b) \mapsto q_2 \\ (q_1, q_1, b) \mapsto q_1, q_2.\end{gathered}$$

Nový přechod tedy vypadá následovně

$$(\{q_0, q_1\}, \{q_0, q_1\}, b) \mapsto \{q_1, q_2\}$$

Takových přechodů je v tomto novém automatu mnoho, uvedeme tedy jen některé na ukázku.

$M_d = (Q, \Sigma, \Delta, F)$, kde
$Q = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$,
$\Sigma = \{a, b\}$,
$\Delta = \{$

$(a) \mapsto q_0$,
$(\{q_0\}, \{q_0\}, b) \mapsto \{q_1\}$,
$(\{q_1\}, \{q_1\}, b) \mapsto \{q_1, q_2\}$,
$(\{q_2\}, \{q_2\}, b) \mapsto \{q_2\}$,
$(\{q_1\}, \{q_0\}, b) \mapsto \{q_2\}$,

$(\{q_1, q_2\}, \{q_1, q_2\}, b) \mapsto \{q_1, q_2\}$
$(\{q_1, q_2\}, \{q_2\}, b) \mapsto \{q_2\}$
$(\{q_0, q_1, q_2\}, \{q_0, q_1, q_2\}, b) \mapsto \{q_0, q_1, q_2\}$
$\vdots$

$\}$,
$F = \{\{q_2\}\}$

Kapitola 4

Ekvivalence jazyků stromových automatů

Pro zjištění ekvivalence jazyků dvou stromových automatů lze opět použít myšlenku, že $L(M) = L(N) \iff L(M) \subseteq L(N) \wedge L(N) \subseteq L(M)$, a že $L(M) \subseteq L(N) \iff L(M) \cap \overline{L(N)} = \emptyset$. Pro komplementaci jazyka automatu je nutné nejprve automat determinizovat. I u stromových automatů operace komplementace zvětšuje stavový prostor kvadraticky a determinizace exponenciálně. Jsou to tedy drahé operace, které se snažíme redukovat.

O to se snaží algoritmy testu ekvivalence založené na kongruenci, které jsou cílem naší práce a algoritmy využívající protiřetězce, se kterými budeme nový algoritmus porovnávat. Oba algoritmy používají determinizaci za běhu zároveň s testem prázdnosti, čímž zabraňují vytváření stavů, které během testu ekvivalence nevyužijí. Dále využívají různé optimalizace, u kongrunčních algoritmů je to výpočet kongruenčního uzávěru (sekce 5.3), které jim umožní redukovat stavový prostor ještě více.

Dostáváme se tedy k cíli naší práce. Navrhneme postup, který by vycházel z algoritmů Hopcrofta a Karpa a Bonchiho a Pouse a zjišťoval by ekvivalenci stromových automatů. Nejdříve definujeme bisimulaci pro stromové automaty 4.0.3, která je u DTA ekvivalentní jazykové ekvivalenci. Potom vytvoříme algoritmus pro výpočet této relace pro DTA. Nakonec ho spojíme s determinizací za běhu a dostaneme algoritmus pro testování ekvivalence jazyků NTA.

Definice 4.0.3. Mějme stromové automaty $M = (Q_M, \Sigma, \Delta_M, F_M)$ a $N = (Q_N, \Sigma, \Delta_N, F_N)$. **Bisimulace** R je binární relace na $Q_M \times Q_N$ pro kterou platí:

i $\forall r_1, \ldots, r_n, r' \in Q_M, a \in \Sigma, s_1, \ldots, s_n \in Q_N :$
$[(r' \in \Delta_M(r_1, \ldots, r_n, a) \wedge (r_1, s_1) \in R \wedge \cdots \wedge (r_n, s_n) \in R) \Rightarrow$
$\exists s' : s' \in \Delta_N(s_1, \ldots, s_n, a) \wedge (r', s') \in R]$

ii $\forall s_1, \ldots, s_n, s' \in Q_N, a \in \Sigma, r_1, \ldots, r_n \in Q_M :$
$[(s' \in \Delta_N(s_1, \ldots, s_n, a) \wedge (r_1, s_1) \in R \wedge \cdots \wedge (r_n, s_n) \in R) \Rightarrow$
$\exists r' : r' \in \Delta_M(r_1, \ldots, r_n, a) \wedge (r', s') \in R]$

iii $\forall r \in Q_M, s \in Q_N; (r, s) \in R :$
$r \in F_M \iff s \in F_N$

Bisimulace nad stromovými automaty M a N je tedy množina dvojic (r, s), kde $r \in M$ a $s \in N$. Maximální bisimulaci nad těmito automaty budeme nazývat *bisimilarita* a značit $\sim_t$.

Teorém 4.0.4. Mějme dva DTA $A = (Q_A, \Sigma, \Delta_A, F_A)$ a $B = (Q_B, \Sigma, \Delta_B, F_B)$. $L(A) = L(B)$ právě tehdy, když

$$\forall a \in \Sigma : [\exists r \in \Delta_A(a) \Rightarrow (\exists s \in \Delta_B(a) \text{ takové, že } r \sim_t s)]$$

$$\bigwedge$$

$$\forall a \in \Sigma : [\exists s \in \Delta_B(a) \Rightarrow (\exists r \in \Delta_A(a) \text{ takové, že } r \sim_t s)]$$

V teorému 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.

Náš nový algoritmus 4.0.2 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 počítáme, zda jsou páry stavů v reflexivním, symetrickém a tranzitivním uzávěru seznamů *todo* a *done*.

Algoritmus 4.0.2: Ekvivalence stromových automatů

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

```
1  done = ∅;
2  todo = {(Δ_M(a), Δ_N(a))};
3  while todo ≠ ∅ do
4      todo = todo \ {(x', y')};
5      done = done ∪ {(x', y')};
6      foreach a ∈ Σ do
7          foreach (r', s') takové, že ∃r_1, ..., r_n ∈ Q_M, s_1, ..., s_n ∈ Q_N :
           (∀i : 1 ≤ i ≤ n ⇒ (r_i, s_i) ∈ e(done ∪ todo)) ∧
8          r' ∈ Δ_M(r_1, ..., x', ..., r_n, a) ∧
9          s' ∈ Δ_N(s_1, ..., y', ..., s_n, a) do
10             if r' ∈ F_M ⇎ s' ∈ F_N then
11                 return false;
12             end
13             if (r', s') ∉ e(done ∪ todo) then
14                 todo = todo ∪ {(r', s')};
15             end
16         end
17     end
18 end
19 return true;
```

Nový algoritmus 4.0.2 počítá pouze s DTA. Lze ho upravit obdobně jako BP spojením stromové bisimulace s determinizací za běhu. Nový algoritmus 4.0.3 bude testovat ekvivalenci i pro nedeterministické stromové automaty. Navíc počítá kongruenční uzávěr relace, jako tomu bylo u NFA. Velká písmena X a Y značí stavy determinizované podmnožinovou konstrukcí.

Algoritmus 4.0.3: Ekvivalence nedeterministických stromových automatů

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

$done = \emptyset$;
$todo = \{(\Delta_M(a), \Delta_N(a)) \mid a \in \Sigma\}$;
while $todo \neq \emptyset$ **do**
    $todo = todo \setminus \{(X', Y')\}$;
    $done = done \cup \{(X', Y')\}$;
    foreach $a \in \Sigma$ **do**
        foreach (R', S') *takové, že* $\exists R_1, \ldots, R_n \in Q_M, S_1, \ldots, S_n \in Q_N$:
        $(\forall i : 1 \leq i \leq n \Rightarrow (R_i, S_i) \in c(done \cup todo)) \wedge$
        $R' \in \Delta_M(R_1, \ldots, X', \ldots, R_n, a) \wedge$
        $S' \in \Delta_N(S_1, \ldots, Y', \ldots, S_n, a)$ **do**
            if $R' \in F_M \nLeftrightarrow S' \in F_N$ **then**
                return false;
            end
        end
        if $(R', S') \notin c(done \cup todo)$ **then**
            $todo = todo \cup \{(R', S')\}$;
        end
    end
end
return true;

Při zamyšlení nás napadá, proč se počítá reflexivní symetrický a tranzitivní uzávěr, když jsou množiny stavů Q_M a Q_N automatů M a N disjunktní. Ano v tomto případě tato optimalizace nevede k vylepšení, ba naopak, náročnost algoritmu se zhorší počítáním těchto uzávěrů, které zůstanou nevyužity. Kongruenční uzávěr pro nedeterministické automaty ale smysl má i přesto. Tento algoritmus byl cílený na automaty, jejichž množiny stavů se částečně, nebo nejlépe úplně, překrývají. Ukázalo se ale, že i když nemají žádné společné stavy, optimalizace kongruenčním uzávěrem funguje překvapivě dobře.

Příklad 4.0.2. V příkladu si ukážeme průběh algoritmů komplementace, algoritmu kongruence pro deterministické automaty a algortimu kongruence pro nedeterministické automaty. Použijeme dva NTA M a N se společnými stavy t, u a v.

$M = (Q, \Sigma, \Delta, F)$, kde
$Q = \{r, t, u, v\}$,
$\Sigma = \{a, b\}$,
$\Delta = \{$
$(a) \mapsto \{r\}$,
$(r, r, b) \mapsto \{t\}$,
$(t, t, b) \mapsto \{t, u\}$,
$(u, u, b) \mapsto \{t, u\}$,
$(t, u, b) \mapsto \{v\}\}$,
$F = \{\{v\}\}$

$N = (Q, \Sigma, \Delta, F)$, kde
$Q = \{s, t, u, v\}$,
$\Sigma = \{a, b\}$,
$\Delta = \{$
$(a) \mapsto \{s\}$,
$(s, s, b) \mapsto \{u\}$,
$(t, t, b) \mapsto \{t, u\}$,
$(u, u, b) \mapsto \{t, u\}$,
$(t, u, b) \mapsto \{v\}\}$,
$F = \{\{v\}\}$

Komplementační algoritmus nejprve při determinizaci podmnožinovou konstrukcí vygeneruje $2^4 = 16$ stavů pro oba automaty. Potom provede průnik, čímž vygeneruje $16 \cdot 16 = 256$ stavů z determinizovaných automatů.

Pro použití algoritmu pro deterministické automaty prvně determinizujeme automaty podmnožinovou konstrukcí. Získáme opět dvakrát 16 stavů. Poté budeme testovat bisimulaci. Z listových stavů získáme a do R uložíme dvojici $(\{r\}, \{s\})$. Při dalších iteracích se postupně do R přidávají další páry stavů. Nakonec je $R = \{(\{r\}, \{s\}), (\{t\}, \{u\}), (\{t, u\}, \{t, u\}), (\{t, u, v\}, \{t, u, v\})\}$. Poté už negeneruje žádné nové stavy a může skončit.

Algoritmus kongruence pro NTA ve skutečnosti také testuje pouze deterministické automaty, ale determinizaci provádí za běhu. Do R se postupně generují stavy, obdobně jako v předchozím algoritmu. $R = \{(\{r\}, \{s\}), (\{t\}, \{u\})\}$. Další vygenerovaný stav $(\{t, u\}, \{t, u\})$, již leží v kongruenčním uzávěru R. Srovnání počtu vytvořených stavů je uvedeno v tabulce 4.1.

Tabulka 4.1: Srovnání algoritmů pro stromové automaty

Algoritmus	**Počet stavů**		
	původní automaty	**determinizace**	**průnik/bisimulace**
Komplementace	4 + 4	16 + 16	256
Hopcroft a Karp	4 + 4	16 + 16	4
Bonchi a Pouse	4 + 4	4	2

Kapitola 5

Implementace

Algoritmus byl implementován jako komponenta knihovny VATA v jazyce C++. Jde o knihovnu pro efektivní manipulaci s nedetrministickými stromovými automaty, která byla vytvořena Výzkumnou skupinou automatizované analýzy a verifikace na Fakultě informačnách technologií Vysokého učení technického v Brně - VeriFIT.

5.1 VATA

Vata - knihovna pro efektivní manipulaci se stromovými automaty je open-source knihovna vytvořená pro účely formální verifikace. Kromě verifikace programů pracující s komplexními dynamickými datovými strukturami nebo programů využívající haldu pro ukládání dat, lze knihovnu využít například na model checking regularních stromů nebo na rozhodovací procesy mnohých logik. [13] Tato knihovna zprostředkovává nejnovější a vysoce optimalizované algoritmy pro práci nejen s nedeterministickými stromovými automaty, ale i deterministickými TA (DTA je přece speciální případ NTA) a slovními automaty. Slovo je v podstatě unární strom [14].

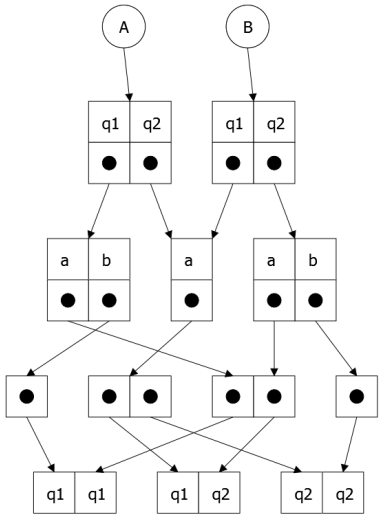
5.1.1 Vnitřní reprezentace automatu

Automaty mohou být v knihovně VATA uloženy několika způsoby. Uživatel si může vybrat, který je pro něj nejvhodnější. Semi-symbolické kódování využívá k uložení přechodové funkce multi-terminální binární rozhodovací diagramy. Druhý způsob zakódování, pro které je implementován i náš algoritmus, je explicitní kódování.

Tato explicitní reprezentace ukládá "top-down"přechody ve tvaru $\Delta(q, a) = \{(q_1, \ldots, q_n), \ldots\}$ do hierarchické datové struktury podobné hashovací tabulce. První hashovací tabulka mapuje stavy na množiny přechodů. Tedy ke stavu je přiřazen ukazatel na vnitřní hashovací tabulku. Tato vnitřní tabulka mapuje symbol na stavy. Přesněji řečeno k symbolu přiřadí ukazatel na množinu. Kdyby byly automaty deterministické, místo této množiny by byla již uspořádaná n-tice stavů. Automaty jsou ale nedeterministické, proto je zde množina ukazatelů na tyto uspořádané n-tice stavů. [13] Schéma uložení je zobrazeno na obrázku 5.1.

Tato reprezentace je vhodnější než jednoduchý lineární seznam přechodů. Pro přechody, se stejným symbolem a n-ticí jsou tyto překrývající se části uloženy pouze jednou. Stejně tak každá n-tice stavů je uložena nejvýše jednou.

Všimněme si, že tento způsob vnitřního uložení uchovává přechody definované shora-dolů. Zatímco náš algoritmus pracuje s přechody zdola-nahoru. To nám nevadí, protože



Obrázek 5.1: Příklad explicitní reprezentace přechodových funkcí stromových automatů A a B. Můžeme vyčíst například přechodové pravidlo $\Delta(q_1, a) = \{(q_1, q_1), (q_1, q_2)\}$ automatu A.

nad automaty v této reprezentaci je ve VATě implementován iterátor, který nám umožní přímý přístup k rodičovskému stavu i k stavům potomků. Na přechod pak akorát pohlížíme opačným směrem.

Tímto přístupem se však vytrácí výhoda tohoto způsobu uložení, kterou je rychlost vyhledání přechodového pravidla. Je zde tedy prostor pro optimalizaci. Může se implementovat podobná struktura pro automaty definované zdola-nahoru.

5.2 Ekvivalence vs. Inkluze

Test ekvivalence a test inkluze jsou téměř stejné problémy. Náš algoritmus je primárně určen pro test ekvivalence jazyků $L(A)$ a $L(B)$. Chceme-li spočítat inkluzi těchto jazyků, převedeme ji na test ekvivalence. A to následujícím způsobem:

$$L(A) \subseteq L(B) \iff (L(A) \cup L(B)) = L(B).$$

Na druhou stranu lze náš algoritmus snadno modifikovat, aby počítal přímo inkluzi dvou automatů. A to tak, že bude sledovat pouze jednostrannou bisimulaci. Tedy pokud se automat A dostane do stavu q, pak musí existovat ekvivalentní stav r v automatu B. Obráceně už to platit nemusí. Test ekvivalence jazyků by se potom převedl na test inkluze takto:

$$L(A) = L(B) \iff (L(A) \subseteq L(B)) \wedge (L(B) \subseteq L(A)).$$

Pro porovnání těchto způsobů byly obě varianty implementovány a můžeme se pokusit zjistit, který pracuje lépe. Je nutné podotknout, že v algoritmu inkluze není možné použít symetrický uzávěr. Nedojde tedy k tak velké redukci počtu kontrolovaných stavů. Proto očekáváme, že inkluze počítaná pomocí algoritmu ekvivalence bude efektivnější i přes přidanou operaci sjednocení.

5.3 Kongruenční uzávěr

Pro výpočet kongruenčního uzávěru relace je použita metoda vycházející z článku Bonchiho a Pouse [5]. Dvojice stavů v počítané relaci jsou uloženy v množině dvojic množin. Vnitřní množina reprezentuje jeden stav determinizovaný podmnožinovou konstrukcí. Dvojice obsahující dva takové stavy znázorňuje, že ty dva stavy jsou spolu v relaci bisimulace.

Použitý algoritmus pak zjišťuje, jestli nová dvojice je v kongruenčním uzávěru této relace. Pracuje tak, že pro každý ze dvou stavů v dvojici vypočítá kanonického reprezentanta třídy ekvivalence, do které tento stav patří. Poté porovná, zda jsou oba prvky ve stejné ekvivalenční třídě. Pokud ano, pak tato dvojice leží v kongruenčním uzávěru této relace. V opačném případě ne. Kanonický reprezentant třídy ekvivalence je největší množina stavů, která se dá vygenerovat z relace R algoritmem 5.3.1. Tato množina je množina stavů původního nedeterministického automatu (stavů značených malými písmeny).

Algoritmus 5.3.1: Výpočet kanonického reprezentanta třídy ekvivalence
input: R - množina reprezentující relaci R,
S - stav, pro který počítám kanonického reprezentanta třídy ekvivalence.

```
1 while Existuje (A, B) ∈ R, takové, že A ⊆ S ∨ B ⊆ S do
2     S = S ∪ A ∪ B;
3     R = R \ {(A, B)};
4 end
5 return S;
```

Algoritmus 5.3.1 postupně hledá ve všech dvojicích relace R stav, který by byl podmnožinou nového stavu S. Dejme příklad, že $R = \{(\{q,r\},\{s,t\}),(\{r,s\},\{u\})$ a nová dvojice $(X,Y) = (\{t,u,v\},\{q,r,v\})$. Potom pro $S_x = X = \{t,u,v\}$ hledáme v R podmnožinu tohoto stavu. Nalezena byla množina $\{u\}$ z dvojice $(A,B) = (\{r,s\},\{u\})$ (*řádek 1*). Nyní provedeme sjednocení druhého stavu z dvojice nalezeného stavu se stavem S_x. $S_x = S_x \cup A = \{t,u,v\} \cup \{r,s\} = \{r,s,t,u,v\}$ (*řádek 2*). Každá dvojice z R může být použita pouze jednou, jinak by mohlo dojít k nekonečnému cyklu opakovaným používáním jednoho páru z R. Odstraníme tedy z R dvojici $(\{r,s\},\{u\})$ (*řádek 3*). Pak znovu hledáme v R podmnožinu nového stavu S_x. Další nalezenou množinou je $\{s,t\}$ z dvojice $(\{q,r\},\{s,t\})$ (*řádek 1*). Po sjednocení vzniká stav $S_x = \{q,r,s,t,u,v\}$ (*řádek 2*). Po *řádek 3* už není v R žádný další stav, který by byl podmnožinou S_x. S_x nyní představuje kanonického reprezentanta třídy ekvivalence do které patří X. Pro druhý prvek z (X,Y) použijeme stejný postup a dostaneme postupně $S_y = \{q,r,v\} \to \{q,r,s,t,v\} \to \{q,r,s,t,u,v\}$. $S_x = S_y$ platí tedy, že X a Y jsou ve stejné třídě ekvivalence a jsou proto i v ekvivalenčním uzávěru relace R.

Pokud budeme uvažovat lineární složitost množinových operací (sjednocení množin, porovnání množin, vyhledání v množině, ...), složitost algoritmu 5.3.1 je, jak je ukázáno v článku od *Bonchiho a Pouse* [5], nr^2, kde $r = |R|$ je velikost relace R a $n = |Q|$ je počet stavů NTA [5].

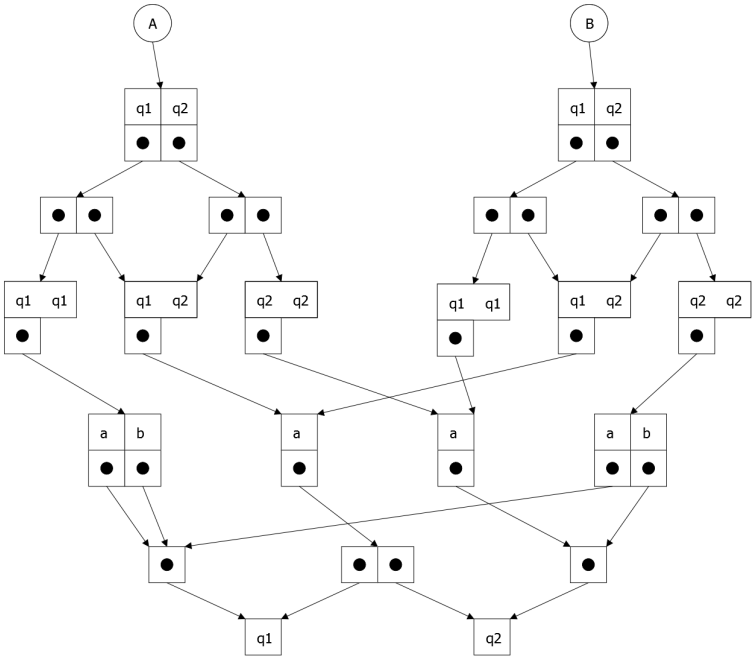
5.4 Optimalizace

Algoritmus byl funkčně implementován. Ale jak se říká, vždycky je co zlepšovat. I tuto implementaci lze optimalizovat, a to hned v několika věcech.

5.4.1 Vnitřní uložení automatu

První z uvedených optimalizací se týká vnitřní reprezentace automatu. V explicitní reprezentaci je automat uložen pouze ve formě shora-dolů (popsáno v sekci 5.1.1). Nad přechody automatu existuje iterátor a přechody se vždy procházejí sekvenčně. Tím se však vytrácí potenciál této reprezentace automatu, kterým je převážně rychlost vyhledání daného přechodu.

Explicitní uložení pro automaty zdola-nahoru bychom navrhli podobně, jak je tomu při opačné podobě. Automat by byl reprezentován ukazatelem na hashovací tabulku. Klíč této tabulky by byl stav a položka množina ukazatelů na n-tice stavů přechodů, ve kterých se tento stav nachází. Tyto n-tice stavů jsou také uloženy jako hashovací tabulka, kde n-tice je klíč a položka je množina symbolů. Množina symbolů je opět reprezentována hashovací tabulkou, jejíž klíč je symbol a prvek je množina stavů, do kterých tento přechod přechází. Tato reprezentace je ukázána na obrázku 5.2. Navržený způsob umožňuje rychlé vyhledání přechodu podle stavů.



Obrázek 5.2: Příklad explicitní reprezentace přechodových funkcí zdola-nahoru stromových automatů A a B. Můžeme vyčíst například přechod $\Delta(q_1, q_2, a) = q_1$ automatu A.

Příklad na obrázku 5.2 ukazuje pro porovnání stejný automat jako na obrázku 5.1. Tyto automaty mají následující přechody:

A:

$$\Delta(q_1, q_1, a) = q_1$$
$$\Delta(q_1, q_1, b) = q_1$$
$$\Delta(q_1, q_2, a) = q_1$$
$$\Delta(q_1, q_2, a) = q_2$$
$$\Delta(q_2, q_2, a) = q_2$$

B:

$$\Delta(q_1, q_1, a) = q_2$$
$$\Delta(q_1, q_2, a) = q_1$$
$$\Delta(q_1, q_2, a) = q_2$$
$$\Delta(q_2, q_2, a) = q_1$$
$$\Delta(q_2, q_2, b) = q_2$$

5.4.2 Kongruenční uzávěr

Další z možných úprav optimalizuje kongruenční uzávěr relace. Relace je implementována jako množina dvojic množin stavů a zda dvojice stavů leží v uzávěru se počítá pomocí výše uvedeného algoritmu (sekce 5.3). Mohlo by být výhodnější rovnou ukládat množinu reprezentující kongruenční uzávěr relace. Ten by se pak nemusel znova počítat pro každý hledaný prvek. Musela by se však zvolit vhodná datová struktura pro uložení této relace a algortimus pro její výpočet. Nechávám to tedy na zamyšlení a jako možnost pokračování v této práci.

Kapitola 6

Výsledky

Testování proběhlo na stroji s dvoujádrovým procesorem s frekvencí 2.6 GHz, s 6 GB operační paměti a s 64-bitovým operačním systémem Ubuntu 16.04.

Provedli jsme srovnání čtyř algoritmů implementovaných v knihovně VATA. Algoritmus komplementace, protiřetězců a dva algoritmy kongruence. Jeden, co testuje přímo ekvivalenci a jeden, co počítá inkluzi tak, jak je to popsáno v sekci 5.2. U těchto algoritmů budeme srovnávat počet generovaných stavů, počet generovaných dvojic a dobu výpočtu. Algoritmus založený na komplementaci jsme neimplementovali. Pro srovnání budeme pouze počítat počty generovaných stavů a dvojic. Algoritmus protiřetězců již ve VATě implementován byl, a tak jsme ho použili. Ten ale testuje inkluzi. Ekvivalence se pak testuje jako inkluze oběma směry. Porovnávaná data jsou tedy z testu inkluze jedním směrem. Počty generovaných stavů a dvojic stavů z obou směrů by byly přibližně dvojnásobné. Stejně tak budeme přistupovat ke kongruenčnímu algoritmu počítajícímu inkluzi.

Použité testovací automaty jsou získané z experimentů s nástrojem pro stromový regulární model-checking použitým v článku [2]. Testovali jsme na vzorku malých automatů s 0 - 100 stavy, středních automatů s 101 - 500 stavy, na velkých automatech s 501 - 1000 stavy. Chtěli jsme testovat i na obrovských automatech s 1000+ stavy, ale kvůli velké časové náročnosti algoritmů to nebylo možné. Dále jsme pro testování vytvořili automaty s překrývající se množinou stavů. Toho je dosaženo tak, že testované automaty jsou sjednoceny s některým dalším automatem, tedy $A \cup M \stackrel{?}{=} A \cup N$, kde A, M, N jsou NTA. Výsledky jsou uvedeny v tabulkách níže. Všechny uvedené hodnoty jsou průměrem vzhledem k počtu testů v dané sadě automatů. Ve sloupci *true* jsou statistiky testů, které vrátili kladný výsledek. Ve sloupci *false* pak analogicky testy se záporným výsledkem. Sloupec *total* shrnuje statistiku všech testů bez ohledu na výsledek.

Tabulka 6.1: Porovnání algoritmů pro malé automaty

Malé automaty	Průměrný počet stavů automatu:								70,4
Algoritmus	**Počet gen. stavů**			**Počet gen. párů**			**Čas výpočtu**		
	true	false	total	true	false	total	true	false	total
Komplementace	x	x	10^{25}	x	x	10^{121}	x	x	x
Protiřetězce	48,5	6	16,6	86,7	17,7	34,9	31 ms	1 ms	9 ms
Kongr. inkl.	39,8	6,1	14,9	20,2	3,1	7,5	658 ms	42 ms	202 ms
Kongr. ekviv.	42,6	3,5	10,5	21,5	1,7	5,3	833 ms	11 ms	159 ms

Tabulka 6.2: Porovnání algoritmů pro středně velké automaty

Střední automaty	Průměrný počet stavů automatu:		311,6						
Algoritmus	**Počet gen. stavů**			**Počet gen. párů**			**Čas výpočtu**		
	true	false	total	true	false	total	true	false	total
Komplementace	x	x	10^{144}	x	x	10^{386}	x	x	x
Protiřetězce	157,8	11,2	58,1	447	26	160,7	639 ms	10 ms	211 ms
Kongr. inkl.	35,5	8,2	17	17,9	4,1	8,5	41,03 s	2,81 s	15,04 s
Kongr. ekviv.	35,5	8	16,3	17,8	4,0	8,1	43 s	2,69 s	14,8 s

Tabulka 6.3: Porovnání algoritmů pro velké automaty

Velké automaty	Průměrný počet stavů automatu:						691,2		
Algoritmus	**Počet gen. stavů**			**Počet gen. párů**			**Čas výpočtu**		
	true	false	total	true	false	total	true	false	total
Komplementace	x	x	10^{144}	x	x	10^{305}	x	x	x
Protiřetězce	363,6	36,2	129,7	1633,1	137,7	564,9	36,7 s	544 ms	10,57 s
Kongr. inkl.	48,9	7,9	22,6	24,5	4	11,3	1844,9 s	72,72 s	705,64 s
Kongr. ekviv.	54,8	6,2	18,9	27,5	3,1	9,5	1674,56 s	58,19 s	481,52 s

Tabulka 6.4: Porovnání algoritmů pro automaty se sdílenými stavy

Sdílené stavy	Průměrný počet stavů automatu:						123		
	Průměrný počet sdílených stavů:						53		
Algoritmus	**Počet gen. stavů**			**Počet gen. párů**			**Čas výpočtu**		
	true	false	total	true	false	total	true	false	total
Komplementace	x	x	10^{41}	x	x	10^{83}	x	x	x
Protiřetězce	46,6	7,3	19,5	80,6	19,5	38,4	50 ms	3 ms	18 ms
Kongr. inkl.	46	6,6	19,2	23,1	3,3	9,7	955 ms	47 ms	337 ms
Kongr. ekviv.	45,4	4,2	12,4	22,8	2,1	6,2	1,1 s	18 ms	231 ms

I přesto, že některá srovnání nejsou tak jednoznačná, z výsledků vyplývá, že z hlediska stavového prostoru jsou kongruenční algoritmy výhodnější i v případě, že automaty nemají žádné společné stavy. Nutno ještě dodat, že jeden pár stavů vygenerovaných v kongruenčním algoritmu odpovídá několika párům v algoritmu protiřetězců. Ten totiž generuje páry (q, Q), kde q je stav nedeterministického automatu a Q deterministického. Dále je z výsledků vidět, že algoritmus protiřetězců je oproti kongruenčnímu algoritmu vysoce optimalizován a je tedy značně rychlejší. Vysoká časová náročnost kongruenčního algoritmu je dána pro náš algoritmus málo efektivním vnitřním uložením automatu, jak je popsáno v sekci 5.4.1. Tato reprezentace neumožňuje efektivní vyhledání přechodu podle stavů, ze kterých vychází. Protože je tato operace používána opravdu velmi často, na velkých automatech se to znatelně projeví. Další příležitostí pro zlepšení je výpočet kongruenčního uzávěru, případně způsobu jeho uložení. Výpočet podle algoritmu ze sekce 5.3 je ještě zatížen množinovými operacemi, kterou je například vyhledání stavu v relaci. Tu můžeme vnímat jako množinu množin. Vnořené uložení způsobuje, že vyhledání stavu má kvadratickou složitost.

Kapitola 7

Závěr

V této práci jsme přišli s algoritmem pro testování inkluze jazyků stromových automatů, který zobecňuje algoritmus Bonchiho a Pouse. Tento nový algoritmus kombinuje test bisimulace optimalizovaný kongruenčním uzávěrem a determinizaci za běhu. Získaný algoritmus značně redukuje počet vytvářených stavů. Předpokládali jsme, že úbytek generovaných stavů bude oproti protiřetězcovému algoritmu výhodnější převážně u automatů, které sdílí část stavového prostoru. Ukázalo se však, že i automaty, které stavy nesdílejí, vytváří méně stavů. V tomto bodu nový algoritmus předčil algoritmus protiřetězců. Na druhou stranu nepříznivým výsledkem je časová náročnost kongruenčního algoritmu, která několikanásobně převyšuje protiřetězce. To je dané nepříliš efektivní implementací, jejíž nedostatky jsou popsány v kapitole 5.4. Problém efektivní implementace kongruenčního uzávěru nechává prostor mnohým úvahám a budeme se mu nadále věnovat. Kromě toho implementujeme vnitřní explicitní reprezentaci automatů definovaných zdola-nahoru a budeme hledat další optimalizace, které by snížily časovou náročnost algoritmu. Dále můžeme provést formální důkaz správnosti nově vytvořeného algoritmu.

Literatura

[1] Abdulla, P.; Bouajjani, A.; Holík, L.; aj.: Composed Bisimulation for Tree Automata. 2008, technical Report FIT-TR-2008-004, FIT BUT, Brno, Czech Republic.

[2] Abdulla, P.; Habermehl, P.; Holík, L.; aj.: Antichain-based Universality and Inclusion Testing over Nondeterministic Finite Tree Automata. 2008, technical Report FIT-TR-2008-001, FIT BUT, Brno, Czech Republic.

[3] Abdulla, P. A.; Chen, Y. F.; Holík, L.; aj.: When Simulation Meets Antichains. In *TACAS'10*, LNCS 6015, Springer, 2010.

[4] Almeida, R.; Holík, L.; Mayr, R.: Reduction of Nondeterministic Tree Automata. In *TACAS'16*, LNCS, Springer, 2016.

[5] Bonchi, F.; Pous, D.: Hopcroft and Karp's algorithm for Non-deterministic Finite Automata. 2011, technical Report hal-00639716v2.

[6] Bonchi, F.; Pous, D.: Hacking nondeterminism with induction and coinduction. Communications- ACM, Association for Computing Machinery, 2015, 58 (2), s. 87–95.

[7] Bouajjani, A.; Habermehl, P.; Rogalewicz, A.; aj.: Abstract Regular Tree Model Checking of Complex Dynamic Data Structures. In *SAS'06*, LNCS, Springer, 2006, s. 52–70.

[8] Comon, H.; Dauchet, M.; Gilleron, R.; aj.: Tree Automata Techniques and Applications. Available on: www.grappa.univ-lille3.fr/tata, 2007, release October, 12th 2007.

[9] Doyen, L.; Raskin, J.: Antichain Algorithms for Finite Automata. In *TACAS, 16th International Conference, 2010. Proceedings*, *Lecture Notes in Computer Science*, ročník 6015, Springer, 2010, s. 2–22.

[10] Habermehl, P.; Holík, L.; Rogalewicz, A.; aj.: Forest automata for verification of heap manipulation. In *CAV 2011*, snowbird, UT, USA, 2011.

[11] Holík, L.: Simulations and Antichains for Efficient Handling of Finite Automata. Disertační práce, Brno, FIT VUT v Brně, 2010.

[12] Hosoya, H.; Vouillon, J.; Pierce, B. C.: Regular Expression Types for XML. In *ACM TOPLAS*, ročník 27, January 2005, s. 46–90.

[13] Lengál, O.; Šimáček, J.; Vojnar, T.: VATA: A Library for Efficient Manipulation of Non-deterministic Tree Automata. In *TACAS'12*, LNCS 7214, Springer, 2012.

[14] Lengál, O.; Šimáček, J.; Vojnar, T.; aj.: libVATA - A C++ library for efficient manipulation with non-deterministic finite (tree) automata [online], Gitgub.com. 21. 3. 2017.
URL github.com/ondrik/libvata

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

[16] Meyer, A. R.; Stockmeyer, L. J.: The equivalence problem for regular expressions with squaring requires exponential space. In *FOCS'72*, iEEE, 1972.

[17] Sipser, M.: *Introduction to the Theory of Computation*. Thomson, druhé vydání, 2006, massachusetts Institute of Technology. ISBN 0-534-95097-3.

Přílohy

Seznam příloh

Příloha A

CD

A.1 obsah

1. libvata
2. outputs
3. tex_src
4. projekt.pdf
5. README

A.2 Libvata

Adresář *libvata/* obsahuje knihovnu VATA [14] doplněnou o naimplementovaný algoritmus. Nově vytvořené soubory se nachází v podsložce *src/explicit_tree_congr_incl_up/*.

Spuštění

Před spuštěním je nutné nejprve knihovnu přeložit příkazem *make release*. Nezapomeňte nainstalovat potřebné knihovny, jak je uvedeno v [14].
Algoritmus založený na kongruenci počítající ekvivalenci je možné spustit příkazem:

build/cli/vata -o dir=up,order=depth,alg=congr,sim=no incl automat1 automat2

Algoritmus založený na kongruenci počítající inkluzi se spustí zadáním příkazu:

build/cli/vata -o dir=up,order=breadth,alg=congr,sim=no incl automat1 automat2

Tyto dva algoritmy jsou rozlišeny parametrem *order*.
Algoritmus založený na protiřetězcích lze spustit příkazem:

build/cli/vata -o dir=up,order=depth,alg=antichains,sim=no incl automat1 automat2

Automaty

V adresáři *automata/* jsou soubory s automaty. Do příkazů místo parametrů *automat1* a *automat2* zadejte cestu ke zvolenému automatu. Soubory s automaty použité při testování jsou dostupné v *automata/artmc_incl/*.

Výstup

Spuštěný algoritmus vypíše na standardní výstup 4 řádky. První řádek udává počet vygenerovaných stavů, druhý řádek počet vygenerovaných párů. Na třetím řádku je čas výpočtu a konečně na čtvrtém řádku se objeví "1"v případě kladného výsledku spuštěného testu a "0"v případě záporného výsledku.

A.3 Outputs

V adresáři *outputs/* jsou data získaná během experimentů. Ty jsou rozděleny do adresářů podle velikosti automatů ze kterých jsou data získána. Ve složce *small/* jsou výstupy algoritmů spuštěných na automatech v adresáři */libvata/automata/artmc_incl/small/*. Jedná se o sadu deseti malých automatů o velikosti do sta stavů. Ekvivalence (respektive inkluze) byla testována mezi každými dvěma automaty v sadě. Obdobně je tomu v adresářích *medium/*, *big/*, *huge/* a *shared/*. Obrovské automaty kvůli dlouhé době výpočtu nebyly vůbec použity.

Výsledky v každém z adresářů jsou rozděleny do tří csv soouborů podle použitého algoritmu: *antichains.csv*, *congr_eq.csv*, *congr_incl.csv*. Tyto soubory obsahují sloupce s hodnotami: počet vygenerovaných stavů, počet vygenerovaných párů, čas, výsledek. Z těchto dat jsou potom vypočítány průměrné hodnoty do tabulek.

A.4 Ostatní

Adresář *tex_src/* obsahuje zdrojové soubory potřebné pro opětovné vysázení textu této práce. Dále CD obsahuje soubor *projekt.pdf* s touto prací a *README* soubor s popisem a instrukcemi k CD.