VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

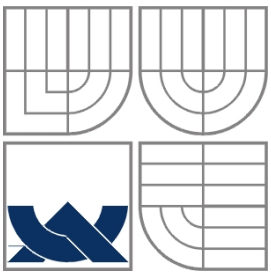
REDUKCE NEDETERMINISTICKÝCH KONEČNÝCH AUTOMATŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

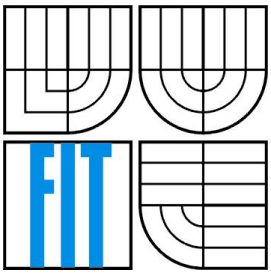
AUTOR PRÁCE
AUTHOR

BC. LUKÁŠ PROCHÁZKA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

REDUKCE NEDETERMINISTICKÝCH KONEČNÝCH AUTOMATŮ

REDUCTION OF THE NONDETERMINISTIC FINITE AUTOMATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. LUKÁŠ PROCHÁZKA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAN KAŠTIL

BRNO 2011

Abstrakt

Nedeterministický konečný automat je důležitým nástrojem, který se používá pro zpracování řetězců v mnoha různých oblastech programování. V rámci zvýšení efektivity programů je důležité snažit se o zmenšování jeho velikosti. Tento problém je však velmi výpočetně náročný, proto je potřeba hledat nové postupy. V této práci jsou uvedeny základy konečných automatů a poté jsou představeny různé metody zabývající se jejich redukcí. Použitelné redukční algoritmy jsou v práci podrobněji popsány, dále implementovány a otestovány. Nakonec jsou výsledky zhodnoceny.

Abstract

Nondeterministic finite automaton is an important tool, which is used to process strings in many different areas of programming. It is important to try to reduce its size for increasing programs' effectiveness. However, this problem is computationally hard, so we need to search for new techniques. Basics of finite automata are described in this work. Some methods for their reduction are then introduced. Usable reduction algorithms are described in greater detail. Then they are implemented and tested. The test results are finally evaluated.

Klíčová slova

nedeterministický konečný automat, NKA, redukce, minimalizace, ekvivalence, předuspořádání, kvaziuspořádání, problém splnitelnosti, prohledávání do šířky

Keywords

nondeterministic finite automaton, NFA, reduction, minimization, minimalization, equivalence, preorder, satisfiability problem, SAT solver, breadth first search

Citace

Lukáš Procházka: Redukce nedeterministických konečných automatů, diplomová práce, Brno, FIT VUT v Brně, 2011

Redukce nedeterministických konečných automatů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Kaštila.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Procházka
24.5.2011

Poděkování

Chtěl bych poděkovat svému vedoucímu práce Ing. Janu Kaštilovi za vedení při zpracovávání diplomové práce.

Obsah

1 Úvod

Regulární výrazy jsou jedním ze základních prostředků popisu řetězců. Regulární výrazy jsou v aplikacích implementovány pomocí nedeterministických konečných automatů (dále jen NKA, anglicky "nondeterministic finite automaton" - NFA). Každý NKA lze také převést na deterministický konečný automat (dále jen DKA, anglicky "deterministic finite automaton" - DFA). NKA nacházejí uplatnění v mnoha různých oblastech, které se obvykle zabývají zpracováním řetězců. Patří mezi ně například zpracovávání přirozeného jazyka, zpracovávání programovacích jazyků, použití v textových editorech, zpracovávání DNA řetězců v biologii, vyhledávání vzorů v řetězcích nebo konstrukce DKA z regulárního výrazu. (Příklady použití převzaty z [20], [21], [28].) Tyto aplikace většinou musejí zpracovat velká množství vstupních dat [21]. V rámci ušetření zabraného místa v paměti nebo zvýšení propustnosti je tedy důležité hledat algoritmy pro zmenšení velikosti NKA.

Nejjednodušší variantou je převést NKA na DKA a ten pak minimalizovat. Minimalizace DKA je totiž jednoduchá a vzniklý minimální automat je navíc jedinečný. Problémem je však expanze počtu stavů při převodu NKA na DKA, kdy DKA může mít až exponenciální velikost v závislosti na velikosti NKA [5]. Je tedy nutné hledat cesty, jak redukovat přímo velikost NKA bez nutnosti převodu na DKA. Tento problém je však na rozdíl od minimalizce DKA velmi výpočetně náročný. Složitost minimalizace počtu stavů NKA je PSPACE-complete [29].

Tato diplomová práce se zabývá algoritmy pro redukci NKA. Jejím cílem bylo seznámit se s problematikou redukce NKA, vyhledat různé prakticky použitelné metody pro zmenšení velikosti NKA a vybrat některé z nich pro implementaci. Implementované algoritmy poté byly vyzkoušeny na regulárních výrazech používaných v systémech pro ochranu moderních sítí. Závěrem práce je pak diskuze dosažených výsledků a navržení zlepšení implementovaných algoritmů s ohledem na použití v systémech pro ochranu moderních sítí.

V kapitole 2 jsou uvedeny základní pojmy a principy. V kapitole 3 jsou představeny různé metody pro redukci NKA a různé další poznatky k tomuto tématu. V kapitole 4 jsou detailněji popsány redukční algoritmy vybrané pro implementaci. Kapitola 5 se zabývá vlastní implementací, popisuje součásti projektu a postup pro jeho spuštění. Kapitola 6 představuje výsledky experimentů s implementovanými algoritmy.

2 Základní pojmy

V této kapitole jsou vysvětleny různé pojmy potřebné pro pochopení dalších částí práce. V první podkapitole jsou nejdříve probrány nejzákladnější pojmy, přes které se postupně dostaneme ke konečným automatům a dalším pojmům, které se jich týkají. V druhé podkapitole jsou probrány různé další pojmy, které se konečných automatů přímo netýkají. Definice základních pojmů z oblasti konečných automatů byly převzaty z [30] a [5].

2.1 Konečné automaty

Regulární výrazy a konečné automaty patří k nejzákladnějším teoretickým konstrukcím informatiky. Regulární výraz slouží k popisu řetězců nějakého jazyka, konečný automat pak tyto řetězce přijímá. Konečný automat je jednoduchý stroj, který má určitý počet stavů a mezi nimi přechází na základě symbolů načtených z řetězce na vstupu. Pro každý řetězec pak rozhodne, zda do daného jazyka patří nebo nepatří. Konečný automat dokáže přijímat pouze regulární jazyky.

Definice 1. *Abeceda* je konečná neprázdná množina symbolů.

Definice 2. *Řetězec* je konečná posloupnost symbolů abecedy. Pokud je prázdný, značí se epsilon - ε.

Definice 3. *Množinu všech řetězců* nad abecedou A značíme A^*.

Definice 4. *Jazyk* $L \subseteq A^*$ je určitá podmnožina množiny všech řetězců nad abecedou A.

Definice 5. *Konkatenace* dvou řetězců $w_1 \cdot w_2$ je řetězec vzniklý spojením těchto řetězců za sebe. Platí analogicky pro dva jazyky, kde se spojují řetězce z prvního jazyka s řetězci z druhého jazyka.

Definice 6. *Iterace* jazyka L^* je jazyk vzniklý 0-n opakováními jazyka L, konkatenovanými do výsledného jazyka.

Definice 7. *Regulární výraz* nad abecedou A:

- $\emptyset$ je regulární výraz označující prázdný jazyk
- ε je regulární výraz označující jazyk, který obsahuje pouze prázdný řetězec
- a je regulární výraz označující jazyk, který obsahuje pouze řetězec a (pro všechna $a \in A$)
- Jsou-li p a q regulární výrazy označující jazyky P a Q, pak:
  - $(p+q)$ je regulární výraz označující jazyk $P \cup Q$
  - (pq) je regulární výraz označující jazyk $P \cdot Q$
  - (p^*) je regulární výraz označující jazyk P^*
- Žádné jiné regulární výrazy nad A neexistují.

Definice 8. *DKA* je pětice $M = (Q, A, \delta, q_0, F)$, kde

- Q je konečná množina stavů,
- A je konečná vstupní abeceda,
- δ: $Q \times A \rightarrow Q$ je přechodová funkce,
- $q_0 \in Q$ je počáteční stav a
- $F \subseteq Q$ je množina koncových stavů.

Definice 9. *NKA* je pětice $M = (Q, A, \delta, I, F)$, kde

- Q je konečná množina stavů,
- A je konečná vstupní abeceda,
- $\delta: Q \times A \rightarrow 2^Q$ je přechodová funkce,
- $I \subseteq Q$ je množina počátečních stavů a
- $F \subseteq Q$ je množina koncových stavů.

Konečný automat se tedy skládá ze stavů, mezi nimiž vedou přechody označené symboly vstupní abecedy. Automat načítá vstupní řetězec po jednotlivých znacích a vždy se z aktuálního stavu vydá po přechodu označeném aktuálně načteným symbolem. NKA však může takovýchto přechodů obsahovat více a automat se pak nachází ve více stavech najednou, takže automat zkouší paralelně více různých průchodů. Pokud se po načtení celého řetězce automat aspoň v jednom průchodu nachází v koncovém stavu, pak je řetězec přijat. Jinak je řetězec odmítnut.

NKA přijímá určitý jazyk nad abecedou A. NKA má stejnou vyjadřovací schopnost jako DKA, to znamená, že každý NKA lze převést na DKA. Oproti DKA má NKA tyto vlastnosti:

- Z jednoho stavu může vést více přechodů označených stejným symbolem, nebo naopak žádný přechod.
- Může obsahovat epsilon-přechody, tedy takový přechod, při kterém není načten žádný symbol ze vstupního řetězce.
- Může obsahovat více počátečních stavů. (Toto lze ovšem převést na případ, kdy máme pouze jeden počáteční stav a z něj vedou epsilon-přechody do dalších stavů, které tímto můžeme vlastně považovat také za počáteční.)

Konečný automat lze vyjádřit například tabulkou, kde na řádcích jsou jednotlivé stavy automatu, ve sloupcích jsou jednotlivé symboly vstupní abecedy a v buňkách grafu jsou stavy, do kterých automat přejde z příslušného stavu po načtení příslušného symbolu. Příklad DKA viz tabulka 2.1, příklad NKA viz tabulka 2.2.

Dále lze konečný automat znázornit graficky, a to takto: kroužky jsou stavy, šipky se symboly jsou přechody, dvojitý kroužek je koncový stav, počáteční stav je označen šipkou. Příklad DKA viz obrázek 2.1, příklad NKA viz obrázek 2.2.

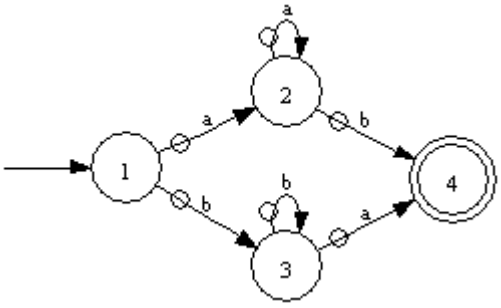
Přechody automatu lze zapisovat $q \in \delta(p, a)$ nebo ekvivalentně (p, a, q).

	a	*b*
1	2	3
2	2	4
3	4	3
4	Ø	Ø

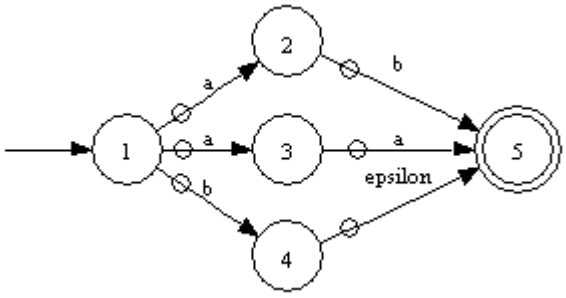
Tabulka 2.1 - Příklad DKA

	a	*b*	*ε*
1	2,3	4	Ø
2	Ø	5	Ø
3	5	Ø	Ø
4	Ø	Ø	5
5	Ø	Ø	Ø

Tabulka 2.2 - Příklad NKA



Obrázek 2.1 - Příklad DKA



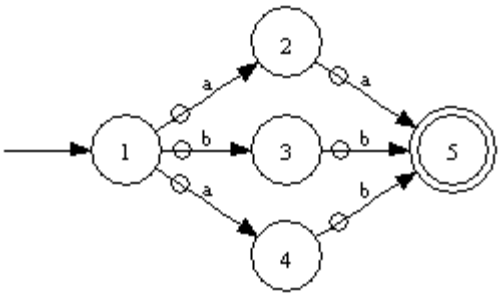
Obrázek 2.2 - Příklad NKA

Dále ještě poznamenejme, že přechodovou funkci δ definovanou nad symboly, lze rozšířit i na řetězce a to takto:
$\delta: Q \times A^* \rightarrow 2^Q$
Pro $p \in Q$, $a \in A$, $w \in A^*$: $\delta(p, \varepsilon) = p$; $\delta(p, aw) = \delta(\delta(p, a), w)$.

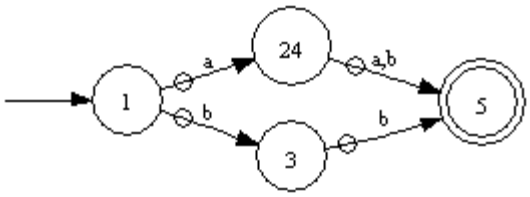
Nyní víme, co je konečný automat, a můžeme se podívat na další pojmy, které se konečných automatů týkají a které budeme v dalším textu potřebovat.

Definice 10. *Determinizace* je postup který má na vstupu NKA a na výstupu DKA, který přijímá stejný jazyk jako vstupní NKA.

Ukázka determinizace automatu je na obrázcích 2.3 a 2.4.



Obrázek 2.3 - NKA pro determinizaci



Obrázek 2.4 - Determinizovaný automat

Definice 11. *Obrácený automat* k automatu M je automat $M^r = (Q, A, \delta^r, F, I)$, kde $q \in \delta^r(p, a)$ pokud $p \in \delta(q, a)$. Obrácený automat se tedy od původního automatu liší tím, že všechny přechody vedou opačným směrem a koncové stavy jsou prohozeny s počátečními stavy. [2] [3]

Definice 12. *Obrácení (reverzace)* automatu je proces vytvoření obráceného automatu z původního automatu.

Definice 13. *Levý jazyk* stavu q v automatu M je $L_L(M, q) = \{w \in A^* \mid q \in \delta(I, w)\}$, tedy množina takových řetězců, pomocí kterých se lze dostat z počátečního stavu do stavu q.

Definice 14. *Pravý jazyk* stavu q v automatu M je $L_R(M, q) = \{w \in A^* \mid \delta(q, w) \cap F \neq \varnothing\}$, tedy množina takových řetězců, pomocí kterých se lze dostat ze stavu q do koncového stavu.

Definice 15. *Rozlišitelné stavy*: Dva stavy p a q jsou rozlišitelné, pokud existuje takový řetězec w, pomocí kterého se lze ze stavu p dostat do koncového stavu a ze stavu q do nekoncového stavu.

Definice 16. *Nerozlišitelné stavy* jsou takové dva stavy, které nejsou rozlišitelné.

Definice 17. *Epsilon-uzávěr* stavu p je množina všech stavů, do kterých se lze dostat ze stavu p pouze po epsilon-přechodech.

Definice 18. *Redukce stavů NKA* je postup, který má na vstupu NKA a na výstupu jiný NKA s co nejmenším počtem stavů, který přijímá stejný jazyk jako vstupní NKA.

Definice 19. *Redukce přechodů NKA* je postup, který má na vstupu NKA a na výstupu jiný NKA s co nejmenším počtem přechodů, který přijímá stejný jazyk jako vstupní NKA.

Vidíme tedy, že se lze zabývat buď redukcí stavů, nebo redukcí přechodů, případně kombinací obou redukcí. Tato práce se zabývá především algorimty pro redukci počtu stavů. Redukcí pouze počtu přechodů ("edge-minimization") se nezabývá, protože podle [15] existuje pouze jediný publikovaný postup pro redukci hran, a to v článku [14]. Všechny ostatní články o redukci NKA se zabývají redukcí počtu stavů. Ve zbytku této práce bude "redukce" znamenat redukci počtu stavů, pokud nebude konkrétně zmíněno, že jde o "redukci přechodů".

Redukce NKA je PSPACE-complete [29], tedy velmi výpočetně náročná. Navíc zde neexistuje žádný unikátní minimální NKA [9] (na rozdíl od minimalizace DKA, která je jednoduchá a vzniklý minimální DKA je unikátní).

2.2 Další pojmy

V této podkapitole jsou popsány různé další pojmy, které budou v dalších kapitolách použity. Jedná se o operace s maticemi a dále o problém splnitelnosti.

Definice 20. *Součet matic:* Mějme matice A a B, obě o rozměrech $m \times n$. Součet matic $A + B$ je matice C o rozměrech $m \times n$, pro jejíž prvky platí: $C_{ij} = A_{ij} + B_{ij}$. [38]

Definice 21. *Součin matic:* Mějme matici A o rozměrech $m \times p$ a matici B o rozměrech $p \times n$. Součin matic $A * B$, je matice C o rozměrech $m \times n$, pro jejíž prvky platí: $C_{ij} = \sum_{k=1}^{p} A_{ik} \cdot B_{kj}$. Důležité tedy je, že pokud chceme násobit dvě matice, tak první musí mít tolik sloupců, kolik má druhá řádků. [39]

Definice 22. *Transpozice matice:* Mějme matici A o rozměrech $m \times n$. Transpozicí matice A je matice B o rozměrech $n \times m$, pro kterou platí $B_{ij} = A_{ji}$. [40]

Definice 23. *Problém splnitelnosti* ("boolean satisfiability problem" nebo také "SAT problem"): Mějme konečnou množinu Booleovských proměnných $X = \{x_1, x_2, x_3, ..., x_n\}$ a množinu klauzulí, kde každá klauzule je disjunkcí některých proměnných a jejich negací. Příklad klauzule: $x_1 \vee x_2 \vee \neg x_3$. Každá proměnná se může vyskytovat ve více různých klauzulích. Cílem SAT problému pak je rozhodnout, zde je tato množina klauzulí splnitelná, tedy zda lze proměnným přiřadit hodnoty *True* nebo *False* tak, aby všechny klauzule měly hodnotu *True*. Tento problém je NP-úplný. [30] [36]

Definice 24. *Konjunktivní normální forma* (CNF): Booleovský výraz je v konjunktivní normální formě, pokud je tvořen konjunkcí klauzulí, z nichž každá je tvořena disjunkcí proměnných a jejich negací. Příklad výrazu v CNF:

$$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$$

[37]

3 Metody redukce NKA

V této části jsou popsány metody redukce NKA, které jsem prozkoumal v rámci semestrálního projektu. Dále jsou také zmíněny různé vlastnosti NKA související s redukcí. Některé nalezené články nabízejí konkrétní redukční metody, jiné jsou spíše teoretičtějšího rázu, a další nepopisují redukční metody, ale věnují se vlastnostem automatů. V tomto pořadí jsou také orgranizovány následující podkapitoly. V prvních třech podkapitolách jsou stručněji popsány čtyři metody, které jsem v rámci diplomové práce implementoval. Tyto metody jsou poté podrobně rozvedeny v následující kapitole.

3.1 Equivalences and preorders

Zde jsou popsány dvě příbuzné metody: *equivalences* a *preorders*. Těmito metodami se zabývá celkem šest článků [1] [2] [3] [4] [5] [6] od dvou různých skupin autorů. Nejdříve jsou vysvětleny vlastní metody, poté je popsáno, čím který článek přispěl.

Metoda equivalences pracuje na principu spojování ekvivalentních stavů do jednoho stavu. Definuje ekvivalenční relaci $\equiv_R$ (podrobnosti viz následující kapitolu), která může platit pro dva stavy automatu. Pokud relace platí, pak jsou dané dva stavy ekvivalentní a lze je sloučit do jednoho. Místo ekvivalence $\equiv_R$ lze také použít ekvivalenci $\equiv_L$, která je definována stejně, ale na obráceném automatu.

Metoda ve skutečnosti nezjišťuje, které stavy jsou ekvivalentní, ale zjišťuje, které stavy ekvivalentní nejsou. Na začátku vytvoří množinu dvojic neekvivalentích stavů, kterou během výpočtu postupně upravuje. Nakonec pomocí množiny dvojic neekvivalentích stavů zjistí množinu dvojic ekvivalentních stavů a všechny navzájem ekvivalentní stavy vždy sloučí do jednoho.

Metoda preorders pracuje s relacemi $\subseteq_R$ a $\subseteq_L$, které jsou definovány stejně jako v equivalences. Liší se pouze tím, že nejde o relaci ekvivalence, ale o relaci kvaziuspořádání. Ta na rozdíl od relace ekvivalence není symetrická. Metoda tentokrát používá obě relace $\subseteq_R$ a $\subseteq_L$ zároveň, pomocí nich lze opět některé stavy sloučit do jednoho. Vlastní výpočet probíhá pomocí operací s maticemi a jde v něm opět o zjišťování množiny dvojic stavů, které v příslušné relaci nejsou.

V článku [1] autoři nejdříve popisují tři různé metody používané pro vytváření NKA z regulárních výrazů (position automata, partial derivative automata, follow automata). Poté poprvé představují metodu *equivalences*, její formální definici, algoritmus pro její výpočet, a ukazují, že dokáže vytvořit menší automat, než výše popsané tři metody. Dále ukazují, že použití obou relací $\equiv_R$ i $\equiv_L$ vede k ještě lepším výsledkům, než použití pouze jedné z nich. Udávají, že jejich algoritmus funguje v polynomiálním čase a k pozdějšímu vyřešení nechávají tyto problémy: zrychlení popsaného algoritmu a optimální využití ekvivalencí $\equiv_R$ a $\equiv_L$.

Článek [2] se věnuje zhruba stejným problémům jako předchozí článek, jen je poněkud obsáhlejší.

V článku [3] autoři popisují novou metodu *preorders*, který vychází z předchozí metody *equivalences*. Dokazují různé její vlastnosti a ukazují algoritmus pro její výpočet, který má pro n stavů a m přechodů časovou složitost $O(mn^2)$. Autoři uvádějí, že jejich algoritmus je účinnější, než předchozí algoritmy založené na *equivalences*.

Článek [4] opravuje chybu v předchozím článku a ukazuje výsledky nových testů s opraveným algoritmem.

V článku [5] se autoři zabývají zrychlením obou předchozích algoritmů. Pro *equivalences* navrhují nový algoritmus založený na algoritmu Paige a Tarjana [34]. Jejich nový algoritmus má časovou složitost $O(m \log n)$. Pro *preorders* navrhují rovněž nový algoritmus, který pomocí matic počítá opak relace $\subseteq_R$, tedy stavy, které v relaci $\subseteq_R$ nejsou. Tento nový algoritmus má časovou složitost $O(mn)$. Dále se ještě zabývají speciálním typem automatu - pozičním automatem - a redukcí, která zachová jeho speciální vlastnosti. Nakonec jsou představeny výsledky mnoha testů.

V článku [6] se autoři ještě jednou vracejí k metodám *equivalences* a *preorders*. Tentokrát se zabývají problémem, jak optimálně zkombinovat relace $\equiv_R$ a $\equiv_L$ (a popisují algoritmus, který toho dosáhne), respektive jak optimálně zkombinovat relace $\subseteq_R$ a $\subseteq_L$ (zde naopak dokazují, že tento problém je NP-těžký).

3.2 Použití SAT solveru

Tento nový článek [8] z roku 2010 nabízí zcela nový přístup v podobě použití SAT solveru. Představuje algoritmus redukce NKA, zabývá se zakódováním automatu pro SAT solver a výsledky experimentů.

Funkce algoritmu je následující: Nejdříve je vstupní NKA determinizován a minimalizován. Počet stavů výsledného DKA poslouží pro odhad minimálního a maximalního počtu stavů, které může hledaný NKA mít. Poté je opakovaně hledán NKA s určitým počtem stavů. Vždy se předpokládá, že DKA přijímá stejný jazyk jako hledaný NKA a tedy vznikl jeho determinizací. Na základě tohoto předpokladu jsou kódovány vztahy DKA a hledaného NKA pomocí množiny klauzulí SAT problému. Výsledný SAT problém je poté předán SAT solveru. Pokud SAT solver nalezne řešení, algoritmus končí. Jinak algoritmus pokračuje dalším cyklem, ve kterém je počet stavů hledaného NKA o jedničku větší. Pokud se nepodaří nalézt řešení, vrátí nakonec algoritmus DKA, nebo vstupní NKA, podle toho, který má menší počet stavů.

Výše popsaný přístup tedy převádí jeden výpočetně náročný problém na jiný, který je ale intenzivněji studovaný. SAT solvery jsou hodně používané a dobře optimalizované. Dokonce probíhají i soutěže o nejlepší SAT solver [35]. Chování algoritmu bude zřejmě záviset na volbě konkrétního SAT solveru, autoři ale udávají, že je velmi úspěšný a většinou dokáže najít skoro minimální NKA.

3.3 Prohledávání do šířky

Autoři článku [7] se nezabývají poze redukcí NKA, ale širším tématem zpracování regulárních výrazů. Kromě redukce NKA řeší také například vytváření co nejmenších NKA z regulárních výrazů, redukci velikosti vstupní abecedy symbolů, nebo "multi-stride" NKA, což je takový automat, který v jednom přechodu zpracuje více než jeden symbol ze vstupního řetězce. Tyto postupy mohou vést k ještě větší redukci NKA.

Vlastní algoritmus funguje na principu prohledávání do šířky na automatu, kdy pro každý stav a každý symbol abecedy se zjišťuje sjednocení epsilon-uzávěrů dosažitelných z daného stavu pomocí daného symbolu. Toto sjednocení je uloženo jako množina a podoba výsledného automatu se zjišťuje pomocí množinových operací nad těmito množinami (sjednocení, průnik, rozdíl).

Velká část článku se dále zabývá použitím a implementací NKA v FPGA a analýzou dat z reálných aplikací.

3.4 Další redukční metody

Článek [9] z roku 1970 obsahuje jeden z nejstarších popsaných algoritmů pro redukci NKA. Je to také jeden z nejodkazovanějších článků. Většina článku se zabývá zaváděním pojmů, různých matic popisujících automaty, dokazováním vlastností a postupným vysvětlováním celého procesu redukce.

Redukční postup používá různé automaty vytvořené ze vstupního NKA (obrácený NKA, determinizovaný NKA, determinizovaný obrácený NKA, atd.) a popisuje je pomocí celkem šesti různých matic:

- State composition matrix (SCM)
- States map (SM)
- Elementary automaton matrix (EAM)
- Reduced states map (RSM)

- Reduced automaton matrix (RAM)
- Cover map (CM)

Hlavním principem algoritmu je vytvoření matice RAM, hledání minimálního pokrytí na ní a další práce s touto maticí. Celá práce je silně teoreticky zaměřená, nezabývá se praktickou aplikací a neobsahuje žádné testy. Autoři udávají, že jejich algoritmus je účinnější než jediný tehdy známý způsob - řešení hrubou silou. Připouštějí však, že úspěšnost tohoto algoritmu není zaručena. Ostatní autoři pak udávají, že tento algoritmus je v praxi nepoužitelný ([3], [5]).

Další článek [10] se zabývá hledáním co nejmenšího NKA jako podmnožiny jiného automatu. Definuje pojmy kanonický automat a fundamentální automat, což jsou podobné automaty, pro které platí, že tento automat přijímá daný jazyk a každý NKA přijímající daný jazyk je podmnožinou tohoto automatu. Článek se dále zabývá metodou hledání co nejmenšího NKA ve fundamentálním automatu pomocí heuristiky. Autoři řeší také praktickou implementací algoritmu, který byl implementován v rámci programu AMoRE [27]. Autoři uvádějí, že jejich algoritmus má časovou složitost $O(2^{(n^2)})$, kde *n* je počet stavů minimálního DKA pro daný jazyk. Dále však udávají, že jejich implementace v mnoha praktických případech funguje v přijatelném čase.

Na rozdíl od předchozích se článek [11] nezabývá přímo redukcí NKA, ale příbuzným tématem, a sice konstrukcí malého NKA z regulárního výrazu. Popisuje metody používané pro konstrukci NKA z regulárního výrazu (position automata, follow automata, partial derivative automata) a zabývá se jejich vlastnostmi. Redukce je zmíněna jen krátce a je na ni použita dříve zmíněná metoda equivalences. Dále se článek zabývá postupy pro generování NKA bez epsilon-přechodů, implementací algoritmů v rámci projektu FAdo a výsledky experimentů provedených na náhodně generovaných regulárních výrazech.

Článek [12] představuje kostru algoritmu, který pomocí heuristiky řeší různé výpočetně náročné problémy. Konkrétně jsou zmíněny tři problémy - redukce počtu stavů NKA, minimalizace disjunktní normální formy a problém obchodního cestujícího. Metoda pracuje na základě prohledávání stavového prostoru všech možných řešení za použití metody "branch-and-bound", která funguje ve dvou krocích:

1. Rozdělení problému na dva nebo více podproblémů.
2. Odhad mezí velikosti stavového prostoru řešení pro každý podproblém. Tyto meze dovolí vyřadit některé podproblémy z hledání.

Článek se dále zabývá různými heuristikami pro tyto dva kroky. Tato práce je spíše obecná, nabízí jen jakousi kostru algoritmu, nikoliv nějaké konkrétní řešení.

Podobně jako předchozí článek také práce [15] teoreticky rozebírá různé aspekty použití heuristik na různé výpočetně náročné problémy. Jako příklad jsou zmíněny následující čtyři problémy - problém obchodního cestujícího, redukce NKA, minimalizace disjunktní normální formy a generování binárních fázově modulovaných signálů (BPM) s minimálními autokorelačními vlastnostmi. Autor se zabývá použitím různých heuristik, metody branch-and-bound, genetických algoritmů a dalších postupů. Článek neobsahuje žádné praktické testy, ani žádné formální definice a důkazy, jde pouze o slovní rozpravu o daných problémech. Není nabídnut žádný konkrétní algoritmus pro redukci NKA.

Autoři článku [13] se zabývají porovnáním tří různých algoritmů pro minimalizaci DKA. Testy jsou provedeny na náhodně generovaných automatech. Tato publikace se sice zabývá algoritmy pro minimalizaci DKA, je zde však uvedena proto, že kromě popisu algoritmů pro minimalizaci DKA obsahuje i jeden algoritmus, který dokáže zpracovat DKA i NKA - Brzozowského algoritmus. Ten vezme na vstupu konečný automat (DKA nebo NKA), provede jeho obrácení, poté determinizaci výsledku, poté opět obrácení a poté opět determinizaci. Vzhledem k tomu, že součástí algoritmu jsou dvě determinizace, je nejhorší časová složitost exponenciální. Autoři však udávají, že v praxi bývá při minimalizaci NKA velmi rychlý a dokládají to výsledky testů na NKA.

Na rozdíl od všech ostatních článků se článek [14] nezabývá redukcí počtu stavů v automatu, ale redukcí počtu přechodů v automatu. Popisuje dva nové algoritmy, formálně dokazuje jejich korektnost, první algoritmus vysvětluje na příkladech. Tato práce je teoretická, praktickým použitím se nezabývá. Autoři pouze v závěru nechávají k pozdějšímu vyřešení problém zrychlení popsaných

algoritmů. Navrhují například metodu branch-and-bound nebo genetické algoritmy. Dva zde popsané redukční algoritmy jsou údajně jediné existující algoritmy pro redukci počtu přechodů v NKA (podle [15]).

Článek [16] představuje nový algoritmus pro redukci počtu stavů NKA. Práce je silně toreticky zaměřená, zabývá se formální definicí algoritmu, vysvětlením na příkladech a formálním důkazem korektnosti algoritmu. Autor udává, že jeho algoritmus má exponenciální časovou složitost. Praktickou aplikací se ovšem nezabývá. Popsaný algoritmus je například podle [5] v praxi nepoužitelný. Algoritmus obsahuje chybu, kterou autor opravuje v článku [17].

Publikace [18] je diplomová práce na téma redukce NKA. Autor vychází z nejstaršího algoritmu [9] a navrhuje zlepšení. Jeho algoritmus funguje na principu rozdělení stavů automatu na více stavů s navzájem různým chováním a poté opětovném spojování stavů. Je to silně matematicky zaměřená práce, zabývá se formální definicí různých pojmů, popisem algoritmu, formálním důkazem korektnosti algoritmu. Nezabývá se jeho časovou složitostí ani praktickou aplikací, to pouze zmiňuje v závěru s tím, že to bude předmětem dalšího výzkumu. Popsaný algoritmus je velmi nepraktický (podle [3]), podobně jako algoritmus z [9], ze kterého vychází, i když autor udává, že jeho algoritmus je efektivnější.

Brněnský autor Libor Polák se v článku [19] zabývá redukcí počtu stavů NKA pomocí takzvaného univerzálního automatu, který přijímá daný jazyk. Hledaný NKA se získá výběrem určité podmnožiny stavů univerzálního automatu. Tato práce je silně matematicky zaměřená, zabývá se především definicí mnoha různých pojmů a vlastností a jejich formálními důkazy. Za zmínku stojí, že autor na rozdíl od zahraničních autorů použil v názvu slovo "minimalization" namísto obvyklého "minimization".

3.5 Vlastnosti automatů

Článek [20] se zabývá důkazy časových složitostí různých problémů týkajících se regulárních výrazů, NKA a DKA. Mimo jiné dokazuje, že je nemožné účinně minimalizovat regulární výrazy ani NKA, pokud neplatí P = PSPACE. Dále pro minimalizaci NKA dokazuje, že nelze zjistit ani velikost minimálního NKA s časovou složitostí $o(n)$, kde n je počet stavů vstupního NKA.

Výpočetní náročností minimalizace NKA se zabývá také článek [21]. Řeší však pouze speciální případ - minimalizaci NKA pro konečné a unární regulární jazyky, pokud je vstup specifikován pomocí DKA. Jde tu tedy o práci zabývající se pouze jistými speciálními případy, takže se jí nebudeme dále zabývat.

V článku [23] se autoři zabývají nedávnými pokroky v oblasti NKA. Uvádějí mnoho různých článků zabývajících se touto oblastí. Konkrétně diskutují například tyto problémy: simulace konečných automatů, redukce konečných automatů, odhady velikosti minimálního NKA a prostorová složitostí jazykových operací.

Stejní autoři se zabývají podobnými tématy také v článku [24]. Znovu zde řeší NKA, DKA a další konečné automaty. Opět se odkazují na mnoho různých článků zabývajících se těmito tématy. Konkrétně diskutují například tyto problémy: simulace různých typů konečných automatů, redukce konečných automatů nebo problémy konečných automatů (např. prázdnost, univerzalita, ekvivalence)

V článku [25] autoři udávají, že (na rozdíl od DKA) může existovat NKA, který není minimální, ale přesto už neobsahuje žádné slučitelné stavy. Dále se zabývají důkazem tvrzení, že pro daný regulární jazyk existuje velikost, nad kterou musí každý NKA obsahovat slučitelné stavy.

Podobné téma jako v předchozím článku řeší také autoři článku [26]. Studují však jiný typ slučování stavů a dokazují, že pro tento typ může existovat libovolně velký NKA bez slučitelných stavů.

Na dříve nezodpovězenou otázku, a to jaká je mez maximálního počtu stavů NKA, které nemohou být sloučeny, hledají odpověď autoři článku [22]. Mimo jiné dokazují, že maximální počet neslučitelných stavů v NKA přijímajícím daný jazyk není větší než 2^n-1, kde n je počet stavů minimálního DKA přijímajícího daný jazyk.

4 Popis vybraných algoritmů

Během studia různých článků jsem nabyl dojem, že opravdu v praxi použitelných algoritmů pro redukci NKA nebude mnoho. Zdá se totiž, že tato oblast není až tolik zkoumaná, jak by si zřejmě zasloužila. Zmiňují se o tom i některé články, například:

"Je nečekané, že takovému velmi důležitému problému, jakým je získávání malých NKA z regulárních výrazů, nebylo věnováno více pozornosti. A navíc, obecnější problém zmenšování velikosti libovolného NKA byl zkoumán ještě méně." [1]

"Zatím neexistují žádné účinné algoritmy (vzhledem k výpočetní složitosti) pro redukci počtu stavů a přechodů NKA." [25]

Množství článků, které v předchozí kapitole zmiňuji, může na první pohled svědčit o opaku. Článků na téma minimalizace NKA je hodně. Při bližším studiu se však ukáže, že autoři se velmi často zaměřují spíše na teorii a matematické důkazy, zatímco jejich algoritmy bývají v praxi nepoužitelné. Takže těch opravdu použitelných mnoho není. Pokusil jsem se tedy vybrat alespoň těch několik algoritmů, které bude možné implementovat a prakticky použít v další části diplomového projektu. Tyto algoritmy jsou podrobně popsány v následujících podkapitolách.

4.1 Slučování stavů automatu

Nejdříve uvedu, jak probíhá slučování stavů automatu, které budou některé redukční metody využívat. Mějme dva stavy s_1 a s_2, které chceme sloučit. Určíme jeden stav, který bude odstraněn, například stav s_2. V automatu pak provedeme následující úpravy:

- Úprava stavů: Odstraníme stav s_2.
- Úprava počátečního stavu: Pokud byl stav s_2 počáteční, označíme stav s_1 jako počáteční.
- Úprava koncových stavů: Pokud byl stav s_2 koncový, označíme stav s_1 jako koncový.
- Úprava přechodů - pro všechny symboly a:
  - Odstraníme každý přechod (s_2, a, s_2) a přidáme přechod (s_1, a, s_1).
  - Odstraníme každý přechod (s_2, a, p) a přidáme přechod (s_1, a, p).
  - Odstraníme každý přechod (p, a, s_2) a přidáme přechod (p, a, s_1).
- (Pro úplnost: Abeceda automatu zůstane nezměněna.)

Lze slučovat i více stavů najednou. V tom případě vybereme jeden stav, který v automatu zůstane, a potom postupujeme analogicky.

4.2 Algoritmus 1: Equivalences

V této podkapitole je vysvětlen princip základní verze metody *equivalences* tak, jak je popsán v článku [2].

Metoda equivalences pracuje na principu spojování ekvivalentních stavů do jednoho stavu. Dva stavy mohou být sloučeny, pokud jsou v realci $\equiv_R$. Relace $\equiv_R$ je definována jako nejhrubší relace ekvivalence (tedy taková relace, která je reflexivní, symetrická a tranzitivní [31]), pro kterou platí obě následující podmínky:

- $\equiv_R \cap (F \times (Q - F)) = \varnothing$, tedy dva stavy jsou ekvivalentní, pokud jsou oba koncové, nebo oba nekoncové.
- Pro každé dva stavy p a q a symbol a platí:
  $p \equiv_R q \Rightarrow \forall q' \in \delta(q, a),\ \exists p' \in \delta(p, a),\ q' \equiv_R p'$, tedy že když jsou stavy p a q ekvivalentní, tak pro každý stav q', do kterého se lze dostat ze stavu q pomocí symbolu a, existuje alespoň jeden stav p', do kterého se lze dostat ze stavu p pomocí symbolu a, takový, že q' je ekvivalentní s p'.

Místo ekvivalence $\equiv_R$ lze pro slučování stavů použít také ekvivalenci $\equiv_L$, která je definována stejně, ale na obráceném automatu. Případně lze použít také kombinaci obou, tím se však dále zabývat nebudu. Autoři původního článku [2] ukazují, že kombinací obou relací lze dosáhnout větší

redukce, ovšem problém, jak optimálně tyto relace zkombinovat, pouze nechávají k pozdějšímu vyřešení v závěru článku.

Za zmínku ještě stojí, že ekvivalentní stavy jsou nerozlišitelné, opak ale nutně platit nemusí. Je to způsobeno tím, že relace $\equiv_R$ je pouze jakýmsi přiblížením, které nemusí zachytit všechny nerozlišitelné stavy.

Samotný algoritmus výpočtu ve skutečnosti přímo nezjišťuje, které stavy jsou ekvivalentní. Místo toho pracuje s množinou dvojic stavů, které ekvivalentní nejsou. Tuto množinu během výpočtu postupně upravuje a až v závěru pomocí ní získá množinu dvojic ekvivalentních stavů.

Princip algoritmu je následující: Algoritmus potřebuje, aby byly definovány všechny přechody. To znamená, aby z každého stavu po každém symbolu vedl alespoň jeden přechod. Nejdříve je tedy přidán "sink" stav, který je nekoncový a jsou do něj svedeny všechny chybějící přechody. Poté se začíná vytvářet množina dvojic stavů, pro které ekvivalenční relace $\equiv_R$ neplatí. Nejdříve jsou jako neekvivalentí označeny každé dva stavy p a q, z nichž jeden je koncový a druhý nekoncový. Dále jsou jako neekvivalentí označeny každé dva stavy p a q, z nichž alespoň jeden je "sink" stav. V hlavním cyklu algoritmu je pak množina neekvivalentních stavů postupně doplňována o další dvojice stavů, které splňují následující podmínku: Pro stavy p a q a symbol a platí, že mezi stavy, do kterých se lze dostat za stavu p pomocí symbolu a, je stav, který není ekvivalentní s žádným stavem, do kterého se lze dostat ze stavu q pomocí symbolu a. (Poznámka: Do množiny neekvivalentních stavů se vždy přidávají zároveň dvojice stavů (p, q) a (q, p), protože relace je symetrická.) Pokud se po dalším průchodu cyklem množina již nezmění, potom cyklus končí. Nakonec je vypočítána množina ekvivalentních stavů, což jsou všechny dvojice stavů, které nejsou v předchozím výpočtem zjištěné množině neekvivalentních stavů. Ekvivalentní stavy jsou v automatu nakonec sloučeny a algoritmus končí.

Algoritmus 1. Redukce NKA pomocí metody *equivalences*:

- Do automatu přidej sink stav a všechny chybějící přechody.
- Do množiny neekvivalentních stavů přidej všechny dvojice stavů (koncový, nekoncový) a (nekoncový, koncový).
- Do množiny neekvivalentních stavů přidej všechny dvojice stavů (jakýkoliv stav, sink stav) a (sink stav, jakýkoliv stav).
- Dokud se mění množina neekvivalentních stavů:
  - Najdi stavy p a q a symbol a, pro které platí: Mezi stavy, do kterých se lze dostat za stavu p pomocí symbolu a, je stav, který není ekvivalentní s žádným stavem, do kterého se lze dostat ze stavu q pomocí symbolu a.
  - Do množiny neekvivalentních stavů přidej dvojice (p, q) a (q, p).
- Pomocí množiny neekvivalentních stavů vypočítej množinu ekvivalentních stavů.
- Všechny navzájem ekvivalentní stavy vždy sluč do jednoho stavu.

Místo relace $\equiv_R$ lze použít relaci $\equiv_L$. Algoritmus pak pracuje úplně stejně, pouze na vstup dostane obrácený automat, který je potřeba předtím vytvořit ze vstupního NKA. Výpočet pak probíhá nad obráceným NKA, ale stavy jsou nakonec samozřejmě sloučeny v původním NKA.

4.3 Algoritmus 2: Preorders

V této podkapitole je vysvětlen princip základní verze metody *preorders* tak, jak je popsán v článku [3].

Metoda preorders pracuje s relacemi $\subseteq_R$ a $\subseteq_L$. Relace $\subseteq_R$ je relace kvaziuspořádání (nebo též předuspořádání, tedy taková relace, která je reflexivní a tranzitivní [32] [33]), pro kterou platí stejné dvě podmínky jako pro relaci $\equiv_R$, tedy:

- $\subseteq_R \cap (F \times (Q - F)) = \emptyset$
- Pro každé dva stavy p a q a symbol a platí:
  $q \subseteq_R p \Rightarrow \forall q' \in \delta(q, a),\ \exists p' \in \delta(p, a),\ q' \subseteq_R p'$

Relace $\subseteq_L$ je opět definována stejně, ale na obráceném automatu. Relace *preorders* se tedy liší od relací *equivalences* pouze v tom, že nejsou symetrické. Dále lze také říci, že platnost relace $p \equiv_R q$ implikuje platnost relací $p \subseteq_R q$ a $q \subseteq_R p$, naopak to ale platit nemusí.

Máme-li relace $\subseteq_R$ a $\subseteq_L$, pak můžeme sloučit dva stavy, pokud pro ně platí jedna z těchto podmínek:

1. $p \subseteq_R q$ and $q \subseteq_R p$
2. $p \subseteq_L q$ and $q \subseteq_L p$
3. $p \subseteq_R q$ and $p \subseteq_L q$ and $L(p, p) = \{\varepsilon\}$ (Třetí část podmínky říká, že stav p nesmí sám o sobě generovat žádný jazyk, tedy nesmí obsahovat jiný než epsilon-přechod sám na sebe. Tato třetí část původně v podmínce zahrnuta nebyla, toto bylo opraveno až v článku [6].)

Po sloučení dvou stavů musí být relace aktualizovány. Předpokládejme, že jsme sloučili stavy p a q do jednoho stavu q. Pak je potřeba upravit relace následovně (pravidla 1-3 viz výše):

1. Odebrat z $\subseteq_L$ všechny dvojice stavů (q, p'), pro které neplatí $p \subseteq_L p'$.
2. Odebrat z $\subseteq_R$ všechny dvojice stavů (q, p'), pro které neplatí $p \subseteq_R p'$.
3. Tato možnost nepotřebuje žádnou úpravu.

Algoritmus pracuje, podobně jako u *equivalences*, na principu hledání stavů, které v relaci *preorder* nejsou. Množina dvojic stavů, které v relaci nejsou, je tentokrát vyjádřena maticí. Matice X je matice nul a jedniček, která vyjadřuje nějakou relaci R takto: $X_{i,j} = 1 \Leftrightarrow jRi$.

Dále jsou popsány různé matice, se kterými algoritmus pracuje:

- Matice X_0, která vyjadřuje relaci $F \times (Q \backslash F)$, což jsou všechny dvojice stavů (p, q), kde p je koncový a q je nekoncový.
- Množina matic δ, kde každá matice δ_a vyjadřuje relaci $q \in \delta(p, a)$. Jsou to tedy všechny přechody automatu se symbolem a.
- Množina matic ${}^t\delta$, kde každá matice ${}^t\delta_a$ je tranponovanou maticí δ_a.
- Matice V, kde platí: $V_{a,p} = |\delta(p, a)|$. Je to tedy počet přechodů automatu ze stavu p po symbolu a.
- Matice X, která je postupně upravována v hlavním cyklu algoritmu.

Princip algoritmu pro výpočet relace *preorder* je následující: Nejdříve je vytvořena matice X, která vznikne transpozicí matice X_0. V hlavním cyklu se pak postupně pro každý symbol a z abecedy automatu provádí následující kroky:

1. Výpočet matice X_1:
   $X_1 = X * \delta_a$
   Zde se jedná o násobení matic v oboru přirozených čísel (a nuly).
2. Výpočet matice X_2:
   $X_{2i,j} = 1$ pokud $X_{1i,j} = V_{a,j}$
   $X_{2i,j} = 0$ jinak
3. Výpočet matice X_3:
   $X_3 = {}^t\delta_a * X_2$
   Zde se jedná o násobení matic v oboru booleovských hodnot. Tedy 0 = *False*, 1 = *True*, logický součin (and) je místo násobení a logický součet (or) je místo sčítání.
4. Aktualizace matice X:
   $X = X_3 + X$
   Zde se jedná o sčítání matic v oboru booleovských hodnot. Tedy 0 = *False*, 1 = *True* a logický součet (or) je místo sčítání.

Pokud proběhne cyklus algoritmu, během kterého se matice X nezmění pro všechny symboly a, pak hlavní cyklus algoritmu končí. Transpozicí matice X pak získáme výslednou matici, která vyjadřuje opak relace *preorder*. Pomocí ní nakonec získáváme množinu dvojic stavů v relaci *preorder*.

Algoritmus 2. Výpočet relace *preorder* pro NKA:

- Vytvoř matici X_0 vyjadřující relaci $F \times (Q \backslash F)$ v NKA.
- $X = {}^{t}X_0$
- Dokud se matice X mění:
  - Pro všechny symboly a:
    - $X_1 = X * \delta_a$
    - $X_{2i,j} = 1$ pokud $X_{1i,j} = V_{a,j}$
      $X_{2i,j} = 0$ jinak
    - $X_3 = {}^{t}\delta_a * X_2$
    - $X = X_3 + X$
- Matice ${}^{t}X$ vyjadřuje stavy, které v relaci *preorder* nejsou. Pomocí ní získej stavy, které v relaci *preorder* jsou.

Hlavní algoritmus redukce NKA pak nejdříve vytvoří ze vstupního automatu obrácený automat, poté zavolá výše popsaný algoritmus jednou pro vstupní automat a jednou pro obrácený automat. Tím získá relace $\subseteq_R$ a $\subseteq_L$, pomocí nichž pak může slučovat stavy vstupního automatu podle pravidel, která byla popsána výše.

Algoritmus 3. Redukce NKA pomocí metody *preorders*:

- Vytvoř ze vstupního automatu obrácený automat.
- Získej relaci $\subseteq_R$ zavoláním *Algoritmu 2* pro vstupní automat.
- Získej relaci $\subseteq_L$ zavoláním *Algoritmu 2* pro obrácený automat.
- Pomocí relací $\subseteq_R$ a $\subseteq_L$ slučuj stavy a příslušně upravuj relace.

4.4 Algoritmus 3: Použití SAT solveru

V této podkapitole je vysvětlen princip metody založené na použití SAT solveru, která byla představena v článku [8]. Jde o úplně nový přístup, který spočívá v převodu jednoho obtížného problému na jiný, který je ovšem daleko více studovaný a tím pádem ho umíme lépe řešit. Princip algoritmu spočívá v zakódování hledaného NKA do SAT problému. SAT problém je pak předán k vyřešení SAT solveru.

4.4.1 Popis algoritmu

Tento redukční algoritmus pracuje se třemi automaty: vstupním NKA, DKA, který vznikne determinizací a minimalizací vstupního NKA, a hypotetickým hledaným NKA. Nejdříve je tedy ze vstupního NKA vytvořen DKA. Počet stavů tohoto DKA určuje rozmezí pro počet stavů hledaného NKA. Označme $|S_{NKA}|$ počet stavů vstupního NKA a $|S_{DKA}|$ počet stavů DKA. Pak spodní hranice počtu stavů hledaného NKA je určena hodnotou $\log_2 |S_{DKA}|$ (což se ještě musí zaokrouhlit na nejbližší vyšší celé číslo). Horní hranice počtu stavů hledaného NKA je určena hodnotou $|S_{NKA}|$-1 nebo hodnotou $|S_{DKA}|$-1 podle toho, která je menší. Během výpočtu začínáme na spodní hranici počtu stavů a pokud SAT solver takový NKA nenalezne, pak zvýšíme počet stavů o jedničku.

DKA, který má m stavů a a symbolů v abecedě, lze vyjádřit tabulkou 4.1. Význam hodnot v tabulce 4.1 je následující:

- f_i - hodnota 1, pokud stav i je koncový, jinak 0
- α_i - i-tý symbol abecedy automatu
- z_{jk} - číslo stavu, do kterého vede přechod ze stavu j po symbolu k. Pokud takový přechod neexistuje, má z_{jk} hodnotu 0.

Všechny sloupce tabulky 4.1 obsahují konkrétní hodnoty získané z DKA.

Koncový	Stav	α_1	α_2	...	α_a
f_1	1	z_{11}	z_{12}	...	z_{1a}
f_2	2	z_{21}	z_{22}	...	z_{2a}
...	...	...	...	...	...
f_m	m	z_{m1}	z_{m2}	...	z_{ma}

Tabulka 4.1: DKA

Nyní začíná hlavní cyklus algoritmu, kde hledáme NKA s určitým počtem stavů, který označíme n. Vycházíme z předpokladu, že DKA vznikl determinizací hledaného NKA a vyjádříme DKA druhou tabulkou 4.2. Význam hodnot v tabulce 4.2 je následující:

- f_i - hodnota 1, pokud stav i je koncový, jinak 0
- α_i - i-tý symbol abecedy automatu
- T_i - množina hodnot $t_{i1}\ t_{i2}\ ...\ t_{in}$:
  - t_{ij} - hodnota 1, pokud stav i DKA vznikl ze stavu j hledaného NKA při determinizaci, jinak 0
- U_{ik} - pokud $z_{ik} = p$ a p není 0, pak $U_{ik} = T_p$

Hodnoty f_i ve sloupci Koncový jsou dány předchozí tabulkou 4.1. Ostatní sloupce obsahují pouze názvy proměnných, kterým přiřadí konkrétní hodnoty až SAT solver.

Koncový	Stav	α_1	α_2	...	α_a
f_1	T_1	U_{11}	U_{12}	...	U_{1a}
f_2	T_2	U_{21}	U_{22}	...	U_{2a}
...	...	...	...	...	...
f_m	T_m	U_{m1}	U_{m2}	...	U_{ma}

Tabulka 4.2: Zakódovaný DKA

Dále vyjádříme hledaný NKA tabulkou 4.3. Význam hodnot v tabulce 4.3 je následující:

- g_i - hodnota 1, pokud stav i je koncový, jinak 0
- e_i - hodnota 1, pokud stav i je počáteční, jinak 0
- α_i - i-tý symbol abecedy automatu
- X_{jk} - množina hodnot $x_{jk1}\ x_{jk2}\ ...\ x_{jkn}$:
  - x_{jkl} - hodnota 1, pokud v hledaném NKA existuje přechod ze stavu j do stavu l po symbolu k, jinak 0

Sloupec Stav obsahuje konkrétní čísla stavů hledaného NKA. Ostatní sloupce obsahují opět pouze názvy proměnných, kterým přiřadí konkrétní hodnoty až SAT solver.

Koncový	Počáteční	Stav	α_1	α_2	...	α_a
g_1	e_1	1	X_{11}	X_{12}	...	X_{1a}
g_2	e_2	2	X_{21}	X_{22}	...	X_{2a}
...	...	...	...	...	...	...
g_n	e_n	n	X_{n1}	X_{n2}	...	X_{na}

Tabulka 4.3: Zakódovaný hledaný NKA

Poznámka: Výše popsané tabulky 4.2 a 4.3 algoritmus ve skutečnosti explicitně nevytváří. Jde spíše jen o reprezentaci pochopitelnou pro člověka. Algoritmus vytváří rovnou klauzule SAT problému, což bude popsáno dále.

Nyní jsme tedy uvnitř hlavního cyklu algoritmu, máme daný počet stavů n hledaného NKA a potřebujeme zakódovat jeho vztah s DKA. Pomocí klauzulí SAT problému potřebujeme vyjádřit podmínky týkající se počátečních stavů automatu, koncových stavů automatu a přechodů automatu.

Počáteční stavy automatu: Předpokládejme, že počátečním stavem DKA je například stav 1. Pak musí platit:

$$(t_{11} = e_1) \wedge (t_{12} = e_2) \wedge \ldots \wedge (t_{1n} = e_n)$$

Toto je jediná funkce proměnných e, jejichž ohodnocení se vlastně nehledá, protože je stejné jako ohodnocení proměnných t. Tyto klauzule se tedy do SAT problému vůbec nepřidávají.

Koncové stavy automatu: Pro každý řádek z tabulky 4.1 je potřeba přidat jedno pravidlo týkající se koncového stavu. Pokud $f_i = 1$, tedy stav i je koncový, tak musí platit:

$$(t_{i1} \wedge g_1) \vee (t_{i2} \wedge g_2) \vee \ldots \vee (t_{in} \wedge g_n)$$

To znamená, že stav i DKA musel vzniknout z alespoň jednoho koncového stavu hledaného NKA při determinizaci. Podobně pokud $f_i = 0$, tedy stav i není koncový, tak musí platit opak předchozího pravidla:

$$\neg ((t_{i1} \wedge g_1) \vee (t_{i2} \wedge g_2) \vee \ldots \vee (t_{in} \wedge g_n))$$

To znamená, že stav i DKA při determinizaci vznikl ze stavů hledaného NKA, z nichž ani jeden nebyl koncový.

Přechody automatu: Pro každé z_{jk} z tabulky 4.1 je potřeba přidat pravidla týkající se přechodů automatu. Pokud $z_{jk} = p$ a p není 0, pak musí platit:

$$t_{p1} \Leftrightarrow (t_{j1} \wedge x_{1k1}) \vee (t_{j2} \wedge g_{2k1}) \vee \ldots \vee (t_{jn} \wedge x_{nk1})$$
$$t_{p2} \Leftrightarrow (t_{j1} \wedge x_{1k2}) \vee (t_{j2} \wedge g_{2k2}) \vee \ldots \vee (t_{jn} \wedge x_{nk2})$$
$$\ldots$$
$$t_{pn} \Leftrightarrow (t_{j1} \wedge x_{1kn}) \vee (t_{j2} \wedge g_{2kn}) \vee \ldots \vee (t_{jn} \wedge x_{nkn})$$

Pokud naopak $z_{jk} = 0$, pak musí platit:

$$\neg ((t_{j1} \wedge x_{1k1}) \vee (t_{j2} \wedge g_{2k1}) \vee \ldots \vee (t_{jn} \wedge x_{nk1}))$$
$$\neg ((t_{j1} \wedge x_{1k2}) \vee (t_{j2} \wedge g_{2k2}) \vee \ldots \vee (t_{jn} \wedge x_{nk2}))$$
$$\ldots$$
$$\neg ((t_{j1} \wedge x_{1kn}) \vee (t_{j2} \wedge g_{2kn}) \vee \ldots \vee (t_{jn} \wedge x_{nkn}))$$

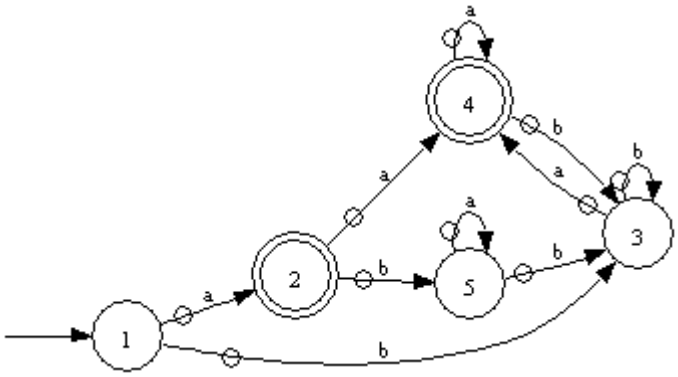
Nyní jsou všechny dostupné informace o hledaném NKA obsaženy v SAT problému, takže ho předáme k vyřešení SAT solveru. Pokud SAT solver nalezne řešení, pak z výsledných hodnot zjistíme informace o počátečních stavech, koncových stavech a přechodech hledaného NKA. Tento NKA vytvoříme a algoritmus končí. Pokud naopak SAT solver řešení nenalezne, pak pokračujeme dalším cyklem algoritmu, kde budeme hledat NKA s počtem stavů o 1 vyšším. Pokud se dostaneme až na horní hranici počtu stavů a žádné řešení nenalezneme, pak je výstupem algoritmu buď vstupní NKA, nebo DKA, podle toho, který má menší počet stavů.

Algoritmus 4. Redukce NKA pomocí metody s použitím *SAT solveru*:
Vstupem je NKA s počtem stavů $|S_{NKA}|$.

- Vytvoř DKA determinizací a minimalizací vstupního NKA. Počet stavů DKA je $|S_{DKA}|$.
- $n = \log_2 |S_{DKA}|$ zaokrouhlené na nejbližší vyšší celé číslo.
- Hlavní cyklus: dokud $n < \min(|S_{NKA}|, |S_{DKA}|)$:
  - Hledaný NKA s n stavy popiš tabulkami 4.2 a 4.3..
  - Pomocí tabulek 4.2 a 4.3 zakóduj vztahy DKA a hledaného NKA do klauzulí SAT problému.
  - Použij SAT solver na SAT problém.
  - Pokud SAT solver nalezl řešení:
    - Ukonči hlavní cyklus.
  - $n = n + 1$
- Pokud SAT solver nalezl řešení:
  - Vytvoř NKA podle informací v nalezeném řešení.
  - Vrať vytvořený NKA.
- Pokud SAT solver nenalezl řešení:
  - Pokud $|S_{NKA}| <= |S_{DKA}|$:
    - Vrať vstupní NKA.
  - Pokud $|S_{NKA}| > |S_{DKA}|$:
    - Vrať DKA.

4.4.2 Příklad SAT problému

Následuje příklad popisující zakódování problému pro SAT solver. Mějme DKA na obrázku 4.1, který vznikl determinizací vstupního NKA. Tento DKA vyjádříme tabulkou 4.4. Booleovské hodnoty v prvním sloupci udávají, zda je daný stav koncový. Ostatní hodnoty v tabulce jsou číselná označení jednotlivých stavů automatu.



Obrázek 4.1: Příklad DKA pro SAT

Koncový	Stav	*a*	*b*
0	1	2	3
1	2	4	5
0	3	4	3
1	4	4	3
0	5	5	3

Tabulka 4.4 - DKA

Dále předpokládejme, že se v aktuálním kroku algoritmu snažíme o nalezení NKA se třemi stavy. Potom DKA popíšeme tabulkou 4.5 a hledaný NKA tabulkou 4.6. Červené hodnoty v tabulce 4.5 označují hodnoty převzaté z tabulky 4.4. Booleovská hodnota t_{ij} označuje, zda stav i v DKA vznikl ze stavu j v NKA při jeho determinizaci. Booleovská hodnota g_i označuje, zda stav i v NKA je koncový. Podobně booleovská hodnota e_i označuje, zda stav i v NKA je počáteční. Booleovská hodnota a_{ij} označuje, zda v NKA existuje přechod z i do j pomocí symbolu a (podobně pro symbol b).

Buňky s hodnotami t_{ij}, a_{ij} a b_{ij} obsahují vždy trojici proměnných, protože hledáme NKA se třemi stavy. Připomeňme, že všechny proměnné t_{ij}, g_i, e_i, a_{ij} a b_{ij} v tabulkách 4.5 a 4.6 jsou zatím pouze názvy proměnných vyjadřujících vztah DKA a hledaného NKA. Konkrétní hodnoty těmto proměnným přiřadí až SAT solver.

Koncový	Stav	a	b
0	$t_{11}t_{12}t_{13}$	$t_{21}t_{22}t_{23}$	$t_{31}t_{32}t_{33}$
1	$t_{21}t_{22}t_{23}$	$t_{41}t_{42}t_{43}$	$t_{51}t_{52}t_{53}$
0	$t_{31}t_{32}t_{33}$	$t_{41}t_{42}t_{43}$	$t_{31}t_{32}t_{33}$
1	$t_{41}t_{42}t_{43}$	$t_{41}t_{42}t_{43}$	$t_{31}t_{32}t_{33}$
0	$t_{51}t_{52}t_{53}$	$t_{51}t_{52}t_{53}$	$t_{31}t_{32}t_{33}$

Tabulka 4.5 - Zakódovaný DKA

Koncový	Počáteční	Stav	a	b
g_1	e_1	1	$a_{11}a_{12}a_{13}$	$b_{11}b_{12}b_{13}$
g_2	e_2	2	$a_{21}a_{22}a_{23}$	$b_{21}b_{22}b_{23}$
g_3	e_3	3	$a_{31}a_{32}a_{33}$	$b_{31}b_{32}b_{33}$

Tabulka 4.6 - Zakódovaný hledaný NKA

Nyní je potřeba zakódovat podmínky týkající se koncových stavů a přechodů hledaného NKA. Pro každý koncový stav DKA (hodnota 1 ve sloupci Koncový v tabulce 4.5) je potřeba udat, že vznikl z alespoň jednoho koncového stavu NKA, tedy:

$$(t_{21} \wedge g_1) \vee (t_{22} \wedge g_2) \vee (t_{23} \wedge g_3)$$
$$(t_{41} \wedge g_1) \vee (t_{42} \wedge g_2) \vee (t_{43} \wedge g_3)$$

Pro nekoncové stavy (hodnota 0 ve sloupci Koncový v tabulce 4.5) postupujeme obdobně, ale přidáme negaci:

$$\neg ((t_{11} \wedge g_1) \vee (t_{12} \wedge g_2) \vee (t_{13} \wedge g_3))$$
$$\neg ((t_{31} \wedge g_1) \vee (t_{32} \wedge g_2) \vee (t_{33} \wedge g_3))$$
$$\neg ((t_{51} \wedge g_1) \vee (t_{52} \wedge g_2) \vee (t_{53} \wedge g_3))$$

Pro ukázku zakódování přechodů hledaného NKA si vyberme první pravidlo z tabulky 4.5 (modře podbarvené buňky). Je potřeba zakódovat skutečnost, že pokud na pravé straně přechodu stav i v DKA vznikl ze stavu j v NKA (t_{ij} v pravé podbarvené buňce), pak na levé straně přechodu stav m v DKA vznikl z nějakého stavu n v NKA (t_{mn} v levé podbarvené buňce) a v NKA existuje přechod ze stavu n do stavu j. Pro vybraný přechod nám podle vzoru $t_{ij} \Leftrightarrow (t_{mn} \wedge a_{nj}) \vee \ldots$ vznikne:

$$t_{21} \Leftrightarrow (t_{11} \wedge a_{11}) \vee (t_{12} \wedge a_{21}) \vee (t_{13} \wedge a_{31})$$
$$t_{22} \Leftrightarrow (t_{11} \wedge a_{12}) \vee (t_{12} \wedge a_{22}) \vee (t_{13} \wedge a_{32})$$
$$t_{23} \Leftrightarrow (t_{11} \wedge a_{13}) \vee (t_{12} \wedge a_{23}) \vee (t_{13} \wedge a_{33})$$

Pro každý další ze zbývajících devíti přechodů v tabulce 4.5 vzniknou analogicky další tři pravidla, celkem tedy bude 30 pravidel. Nyní máme SAT problém tvořený 35 pravidly.

4.4.3 Převod do konjunktivní normální formy

V předchozích dvou podkapitolách jsme si ukázali, jak se tvoří pravidla SAT problému. SAT solver však přijímá pouze výrazy v konjunktivní normální formě. Redukční algoritmus se tedy musí ještě postarat o převod všech pravidel do konjunktivní normální formy. V této podkapitole popíšu, jak takový převod probíhá.

Pravidlo ve tvaru

$$(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee ... \vee (a_n \wedge b_n)$$

lze pomocí zavedení pomocných proměnných převést na tvar

$(c_1 \vee c_2 \vee ... \vee c_n) \wedge (\neg c_1 \vee a_1) \wedge (\neg c_1 \vee b_1) \wedge (\neg c_2 \vee a_2) \wedge (\neg c_2 \vee b_2) \wedge ... \wedge (\neg c_n \vee a_n) \wedge (\neg c_n \vee b_n)$.

Tato transformace sice přidává nové proměnné, ale velikost výrazu zvětšuje pouze lineárně. Princip převzat z [37].

Pravidlo ve tvaru

$$\neg ((a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee ... \vee (a_n \wedge b_n))$$

lze převést na tvar

$$(\neg a_1 \vee \neg b_1) \wedge (\neg a_2 \vee \neg b_2) \wedge ... \wedge (\neg a_n \vee \neg b_n).$$

Zde se jedná o pouhé odstranění negace z prvního pravidla použitím De Morganových zákonů.

Pravidlo ve tvaru

$$a \Leftrightarrow (b_1 \wedge c_1) \vee (b_2 \wedge c_2) \vee ... \vee (b_n \wedge c_n)$$

lze za pomoci zavedení nových proměnných převést na sérii pravidel ve tvaru

$$d \Leftrightarrow (b_2 \wedge c_2) \vee ... \vee (b_n \wedge c_n)$$

$$(\neg a \vee b_1 \vee d) \wedge (\neg a \vee c_1 \vee d) \wedge (a \vee \neg b_1 \vee \neg c_1) \wedge (a \vee \neg d).$$

Nově vzniklé pravidlo s ekvivalencí se převádí stejným způsobem dál. Princip převzat z [8].

4.5 Algoritmus 4: Prohledávání do šířky

Redukční algoritmus z článku [7] je určen především k redukci NKA s epsilon-přechody. Během této redukce dojde ke snížení počtu stavů NKA a zároveň k odstranění všech epsilon-přechodů. Autoři ale udávají, že jejich algoritmus lze použít i k redukci automatu bez epsilon-přechodů. Algoritmus funguje na principu prohledávání do šířky na automatu, kdy postupně prochází stavy automatu a pomocí nich vytváří nové stavy redukovaného automatu.

Algoritmus pracuje s frontou, ve které jsou stavy nově vytvářeného NKA, a dále se slovníkem *subsets*, kde klíče jsou indexy stavů nově vytvářeného NKA a hodnoty jsou množiny stavů původního NKA. Za běhu algoritmu je vytvářen nový NKA, který je na začátku prázdný a postupně jsou do něj přidávány stavy a přechody.

Nejdříve je vytvořen počáteční stav nového NKA s indexem 0, spočítá se epsilon-uzávěr počátečního stavu původního NKA a výsledná množina stavů je uložena do *subsets*(0). Stav 0 je poté vložen do fronty a začíná hlavní cyklus algoritmu, který probíhá, dokud jsou ve frontě nějaké stavy. V hlavním cyklu je pak vždy vybrán z fronty stav s a pro všechny symboly a abecedy automatu se provádí následující: Zjistí se množina stavů dosažitelných ze stavů v *subsets*(s) pomocí symbolu a. Poté se zjistí sjednocení epsilon-uzávěrů stavů zjištěných v předchozím kroku. Výsledná množina stavů *target* se rozdělí na dvě množiny *selfTarget* a *otherTarget*, kde v *selfTarget* jsou stavy, které jsou zároveň v *subsets*(s) a v *otherTarget* jsou ostatní stavy z množiny *target*. Množina *selfTarget* tedy vyjadřuje přechody, které vedou zpět do stejné množiny stavů, množina *otherTarget* vyjadřuje ostatní přechody. Množiny *selfTarget* a *otherTarget* jsou poté zpracovány každá zvlášť, ale stejným způsobem: Pokud daná množina již je v subsets, pak se do nového NKA přidá přechod (s, a, s_1), kde s_1 je stav, který je klíčem dané množiny v *subsets*. Pokud naopak daná množina ještě v *subsets* není, pak je vytvořen nový stav nového NKA s indexem o jedno větším než má předchozí stav a pod tímto indexem je daná množina uložena do slovníku *subsets*. Dále je nový stav vložen do fronty. Nakonec se do nového NKA přidá přechod (s, a, s_2), kde s_2 je nově vytvořený stav.

Takto postupně vznikne nový NKA bez epsilon-přechodů. Nakonec je ještě potřeba určit, které jeho stavy jsou koncové. Budou to všechny stavy *s*, pro které platí, že v *subsets*(*s*) je alespoň jeden koncový stav původního NKA.

Algoritmus 4. Redukce NKA pomocí metody *prohledávání do šířky*:

- Vytvoř počáteční stav nového NKA s indexem *ns* = 0.
- *subsets*(0) = epsilon-uzávěr počátečního stavu původního NKA.
- Vlož stav 0 do fronty.
- Dokud není fronta prázdná:
  - Vyber z fronty stav *s*.
  - Pro všechny symboly *a*:
    - *target* = sjednocení epsilon-uzávěrů stavů dosažitelných ze stavů v *subsets*(*s*) pomocí symbolu *a* v původním NKA.
    - *selfTarget* = stavy, které jsou zároveň v *target* i *subsets*(*s*).
    - *otherTarget* = stavy, které jsou v *target*, ale nejsou v *selfTarget*.
    - Pokud množina *selfTarget* není prázdná:
      - Pokud *selfTarget* je v *subsets*:
        - Najdi klíč s_1, pro který platí *subsets*(s_1) == *selfTarget*.
        - Do nového NKA přidej přechod (*s*, *a*, s_1).
      - Pokud *selfTarget* není v *subsets*:
        - Vytvoř nový stav nového NKA s indexem *ns* = *ns* + 1.
        - *subsets*(*ns*) = *selfTarget*
        - Vlož stav *ns* do fronty.
        - Do nového NKA přidej přechod (*s*, *a*, *ns*).
    - Pokud množina *otherTarget* není prázdná:
      - Pokud *otherTarget* je v *subsets*:
        - Najdi klíč s_1, pro který platí *subsets*(s_1) == *otherTarget*.
        - Do nového NKA přidej přechod (*s*, *a*, s_1).
      - Pokud *otherTarget* není v *subsets*:
        - Vytvoř nový stav nového NKA s indexem *ns* = *ns* + 1.
        - *subsets*(*ns*) = *otherTarget*
        - Vlož stav *ns* do fronty.
        - Do nového NKA přidej přechod (*s*, *a*, *ns*).
- V novém NKA označ jako koncový každý stav *s*, pro který platí, že v *subsets*(*s*) je alespoň jeden koncový stav původního NKA.

5 Implementace

Pro implementaci diplomového projektu byl zvolen programovací jazyk Python. Vlastní projekt není určen k samostatnému použití, ale jako knihovna pro framework Netbench 1.5 [41], který je také napsán v programovacím jazyce Python. Při redukci je vstupem regulární výraz, případně více regulárních výrazů uložených na jednotlivých řádcích v textovém souboru. Netbench zařídí převod regulárního výrazu na konečný automat, případně další pomocné funkce, a knihovna pro redukci se pak už může zabývat pouze redukcí samotnou.

Projekt byl spouštěn a testován pouze na serveru ant-3, protože na vlastním notebooku se systémem Windows XP se nepodařilo Netbench zprovoznit. Konkrétně se jednalo o problém s parserem regulárních výrazů napsaným v jazyce C, který Netbench využívá.

5.1 Netbench 1.5

V této podkapitole jsou popsány součásti frameworku Netbench, které implementovaná redukční knihovna využívá. Mezi hlavní potřebné součásti patří třída `msfm_parser`, která slouží pro zpracování regulárního výrazu na konečný automat pomocí funkce `get_nfa()`. Tato funkce využívá volání parseru regulárních výrazů napsaného v jazyce C. Další důležitou částí je třída `nfa_data`. Je to struktura, která obsahuje vlastní konečný automat, konkrétně:

- `states` - slovník stavů automatu.
- `alphabet` - slovník symbolů abecedy. Pod jedním indexem může být množina více různých symbolů. Případný symbol epsilon se nachází pod indexem -1.
- `start` - index počátečního stavu.
- `transitions` - množinu přechodů automatu, kde přechod je trojice (s_1, a, s_2).
- `final` - množinu koncových stavů automatu.
- `Flags` - slovník příznaků, které určují různé vlastnosti automatu.

Nejdůležitější součástí je třída `b_Automaton`, která reprezentuje NKA a obsahuje mnoho funkcí pro práci s ním. Tato třída v sobě obsahuje NKA ve formě `nfa_data`. Mezi důležité funkce patří:

- `create_from_nfa_data()` - pomocí výše popsaného parseru vytvoří NKA z jednoho regulárního výrazu ze vstupního souboru.
- `create_by_parser()` - pomocí výše popsaného parseru vytvoří jeden velký NKA ze všech regulárních výrazů ve vstupním souboru.
- `get_automaton()` - vrátí strukturu `nfa_data` reprzentující konečný automat. Bez parametrů nebo s parametrem *True* vrátí jeho kopii. S parametrem *False* vrátí odkaz na něj, takže pak přepisujeme přímo původní data.
- `remove_epsilons()` - odstraní z automatu epsilon-přechody a nastaví příznak `"Epsilon Free"`.
- `resolve_alphabet()` - upraví abecedu tak, aby žádné dvě množiny symbolů neobsahovaly stejný symbol a nastaví příznak `"Alphabet collision free"`.
- `reduce_by_simulation()` - redukuje automat pomocí Netbenchové metody *simulation*.
- `reduce_by_bisimulation()` - redukuje automat pomocí Netbenchové metody *bisimulation*.

Poslední použitou součástí je třída `b_dfa`, která obsahuje funkce.

- `determinise()` - vytvoří DKA z NKA a nastaví příznak `"Deterministic"`.
- `minimise()` - minimalizuje DKA a nastaví příznak `"Minimal"`.

5.2 Součásti projektu

Zde jsou popsány jednotlivé součásti knihovny implementované v rámci diplomové práce. Knihovna obsahuje tyto soubory:

- `main.py` - hlavní funkce knihovny, která slouží ke spouštění redukčních metod za účelem jejich testování.
- `equivalences.py` - funkce pro redukci NKA pomocí algoritmu *equivalences*.
- `preorders.py` - funkce pro redukci NKA pomocí algoritmu *preorders*.
- `sat.py` - funkce pro redukci NKA pomocí algoritmu využívajícího *SAT solver*.
- `bfs.py` - funkce pro redukci NKA pomocí algoritmu *prohledávání do šířky*.
- `minisat` - spustitelný SAT solver.

Předpokládá se umístění všech těchto souborů do jedné složky ve frameworku Netbench. Bez něj nelze hlavní program spustit.

5.3 Funkce hlavního programu main.py

Hlavní program slouží k testování redukčních algoritmů. Program načte vstupní soubor, který obsahuje jeden nebo více regulárních výrazů, každý na samostatném řádku. Zpracovávání regulárních výrazů může proběhnout dvěma způsoby: buď je z každého výrazu vytvořen jeden samostatný konečný automat, nebo je ze všech výrazů v souboru vytvořen jeden společný konečný automat. S každým konečným automatem pak lze provést jednu z akcí:

- Redukovat ho pomocí jednoho z algoritmů implementovaných v rámci diplomové práce:
  - equivalences
    - pravá ekvivalence $\equiv_R$
    - levá ekvivalence $\equiv_L$
  - preorders
  - SAT solver
  - prohledávání do šířky
- Redukovat ho pomocí jednoho z Netbenchových redukčních algoritmů (pro porovnání):
  - simulation
  - bisimulation
- Program také může pouze vypsat údaje o automatu a skončit.

Pokud pouze vypisujeme údaje o automatu, pak program vypíše na výstup:

- název vstupního souboru
- identifikaci, že jde o vypsání (parametr `-p`)
- pro každý konečný automat program vypíše:
  - pořadové číslo automatu
  - počet stavů
  - počet přechodů
  - počet symbolů abecedy

Při použití redukčního algoritmu program vypíše na výstup:

- název vstupního souboru
- identifikaci redukční metody
- pro každý konečný automat program vypíše:
  - pořadové číslo automatu
  - počet stavů před redukcí
  - počet přechodů před redukcí
  - počet stavů po redukci
  - počet přechodů po redukci
  - dobu trvání redukce

Poznámka: Počet symbolů abecedy se při redukci nemění.

5.4 Funkce redukčních algoritmů

Redukční algoritmy dostávají na vstupu konečný automat ve formátu `nfa_data`, který redukcí upraví na nový automat. Abeceda automatu je ve struktuře `nfa_data` uložena ve slovníku `alphabet`, kde hodnoty nejsou jednotlivé symboly, ale množiny symbolů. Všechny redukční algoritmy vyžadují, aby ve vstupním automatu nebyl žádný symbol ve více různých množinách. Toho se dosáhne zavoláním funkce `resolve_alphabet()` před vlastním zavoláním redukční metody. Dále všechny algoritmy, kromě *prohledávání do šířky*, požadují na vstupu automat bez epsilon-přechodů. To zařídí zavolání funkce `remove_epsilons()`, opět před zavoláním vlastní redukční metody.

5.5 Použitý SAT solver

Pro redukční metodu využívající SAT solver byl vybrán SAT solver *MiniSat* [42], konkrétně verze 1.14 [43]. Dále pak byly ještě vyzkoušeny různé verze SAT solveru *zChaff* [44], který pro testy použili autoři článku [8]. Ty však na serveru ant-3 nefungovaly správně, končily s chybou `Segmentation fault`. Funkční byla pouze verze 5.13 [45], ale ta byla pomalejší než *MiniSat*. Nakonec byl tedy použit SAT solver *MiniSat*. Ten přijímá vstupní SAT problém ve vstupním textovém souboru a výsledek vypisuje do výstupního textového souboru. SAT problém musí být v konjunktivní normální formě.

Formát vstupního souboru je následující:

- Komentářové řádky začínající znakem `c`.
- Řádek specifikující SAT problém ve formátu: `p cnf <proměnné> <klauzule>`
  - Řetězec `cnf` značí, že jde o problém v konjunktivní normální formě.
  - Hodnota `<proměnné>` udává počet proměnných v SAT problému.
  - Hodnota `<klauzule>` udává počet klauzulí, jejichž konjunkcí je tvořen SAT problém.
- Následující řádky obsahují jednotlivé klauzule, které jsou disjunkcí proměnných a jejich negací. Každá proměnná je reprezentována svým pořadovým číslem. Negace proměnné je vyjádřena znaménkem mínus před číslem proměnné. Konec každé klauzule je označen znakem `0`. Jedna klauzule tedy může být zapsána přes více řádků souboru.

Příklad zakódování: SAT problém ve tvaru

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_1 \vee \neg x_4 \vee \neg x_5)$$

lze zakódovat například takto:

```
c ukazka SAT problemu
p cnf 5 3
1 -2 0
2 3 0
1 -4
-5 0
```

Formát výstupního souboru je následující: Pokud řešení nebylo nalezeno, pak soubor obsahuje pouze jeden řádek s řetězcem `UNSAT`. Pokud řešení nalezeno bylo, pak soubor obsahuje dva řádky - první s řetězcem `SAT` a druhý s ohodnocením proměnných. Každá proměnná je reprezentována svým pořadovým číslem. Pokud jí byla přiřazena hodnota *False*, pak má před svým číslem znaménko mínus, pokud hodnota *True*, pak je číslo bez znaménka. Řádek s ohodnocením proměnných končí znakem `0`.

Následuje ještě *příklad zakódování řešení* SAT problému z předchozího příkladu. Program *MiniSat* nalezl toto ohodnocení proměnných:

$$x_1 = False$$
$$x_2 = False$$
$$x_3 = True$$
$$x_4 = False$$
$$x_5 = False.$$

Toto řešení je ve výstupním souboru zakódováno takto:

```
SAT
-1 -2 3 -4 -5 0
```

5.6 Spuštění hlavního programu

Hlavní program se spouští z příkazové řádky tímto způsobem:

```
python main.py <join> <method> inputFile
python main.py -h
```

Parametr `<join>`:
`-j` - spojit všechny regulární výrazy v souboru do jednoho NKA
`-nj` - nespojovat, vytvořit jeden NKA pro každý regulární výraz

Parametr `<method>`:
`-p` - pouze vypsat údaje o NKA
`-eqR` - redukce metodou equivalences (pravá ekvivalence $\equiv_R$)
`-eqL` - redukce metodou equivalences (levá ekvivalence $\equiv_L$)
`-pre` - redukce metodou preorders
`-sat` - redukce metodou se SAT solverem
`-bfs` - redukce metodou s prohledáváním do šířky
`-simulation` - redukce Netbenchovou metodou simulation
`-bisimulation` - redukce Netbenchovou metodou bisimulation

`inputFile` - vstupní soubor s regulárními výrazy
`-h` - výpis nápovědy

6 Výsledky experimentů

Tato kapitola se zabývá výsledky testů redukčních algoritmů na regulárních výrazech používaných v systémech pro ochranu moderních sítí. Testy byly provedeny pro všechny algoritmy implementované v rámci diplomové práce a pro porovnání i pro dva redukční algoritmy z knihovny Netbench - *simulation* a *bisimulation*. Pro testy byly použity množiny regulárních výrazů získané z programu Snort. Ty jsou uloženy v jednotlivých souborech ve složce `netbench/pattern_match/rules/Moduly/` knihovny Netbench. Některé soubory obsahovaly regulární výrazy, ze kterých vznikaly příliš velké konečné automaty. Pro testy algoritmů tedy byly vybrány pouze ty soubory, u kterých byly doby běhů algoritmů (alespoň některých) časově přijatelné. Vzhledem k počtu testovaných souborů byla časová přijatelnost běhu algoritmu maximálně v řádu několika hodin. Následuje seznam vybraných souborů:

```
backdoor.rules.pcre
chat.rules.pcre
ddos.rules.pcre
dos.rules.pcre
finger.rules.pcre
ftp.rules.pcre
imap.rules.pcre
info.rules.pcre
misc.rules.pcre
mysql.rules.pcre
netbios.rules.pcre
nntp.rules.pcre
p2p.rules.pcre
policy.rules.pcre
pop2.rules.pcre
pop3.rules.pcre
rpc.rules.pcre
shellcode.rules.pcre
specific-threats.rules.pcre
spyware-put.rules.pcre
sql.rules.pcre
telnet.rules.pcre
tftp.rules.pcre
voip.rules.pcre
web-activex.rules.pcre
web-cgi.rules.pcre
web-frontpage.rules.pcre
web-iis.rules.pcre
web-php.rules.pcre
```

S těmito soubory byly provedeny dva druhy testů. V *testech se samostatnými automaty* byl vytvořen a redukován konečný automat pro každý regulární výraz každého souboru. *Testy se sloučenými automaty* byly provedeny s automaty, z nichž každý vznikl ze všech regulárních výrazů v daném souboru. Při všech testech byly pro každý automat zaznamenány tyto údaje:

- počet stavů automatu před redukcí
- počet přechodů automatu před redukcí
- počet stavů automatu po redukci
- počet přechodů automatu po redukci
- doba trvání redukce

Přestože se tato práce zabývá algoritmy pro redukci počtu stavů, byly při testech zaznamenány i údaje o počtech přechodů. V tomto ohledu lze zkoumat, jak závisí redukce počtu přechodů na redukci počtu stavů.

6.1 Testy se samostatnými automaty

V první fázi testování byly provedeny testy se samostatnými automaty z každého regulárního výrazu vstupního souboru. Vlastnosti automatů vzniklých z použitých vstupních souborů shrnuje tabulka 6.1, která obsahuje tyto údaje:

- soubor - první část názvu souboru (bez `.rules.pcre`)
- výrazy - počet regulárních výrazů v souboru
- počet stavů - rozsah a medián počtu stavů automatů vzniklých z jednotlivých regulárních výrazů
- počet přechodů - rozsah a medián počtu přechodů automatů vzniklých z jednotlivých regulárních výrazů
- počet symbolů - rozsah a medián počtu symbolů automatů vzniklých z jednotlivých regulárních výrazů

Medián byl zvolen jako více vypovídající hodnota než průměr, protože některé soubory obsahují hodně automatů s menším počtem stavů a pár automatů s hodně velkým počtem stavů.

		počet stavů			počet přechodů			počet symbolů		
soubor	výrazy	min	med	max	min	med	max	min	med	max
backdoor	154	3	24	104	3	40	296	2	13	26
chat	14	6	12	29	7	35	94	5	8	21
ddos	1	7	7	7	10	10	10	2	2	2
dos	3	28	49	107	106	137	329	7	7	21
finger	1	2	2	2	1	1	1	1	1	1
ftp	35	5	11	212	4	32	2 212	3	7	12
imap	39	9	107	1 046	28	620	7 193	6	8	18
info	1	14	14	14	14	14	14	13	13	13
misc	17	10	34	536	9	107	8 240	5	14	22
mysql	3	25	28	79	59	183	711	14	14	17
netbios	16	3	21	285	2	35	564	1	2	14
nntp	12	28	40	1 032	133	264	8 200	6	8	13
p2p	1	31	31	31	64	64	64	19	19	19
policy	10	10	22	38	19	49	146	4	10	17
pop2	2	7	262	262	7	1 541	1 541	6	6	6
pop3	16	8	16	267	8	65	2 315	5	6	9
rpc	1	5	5	5	4	4	4	2	2	2
shellcode	3	28	30	39	69	105	142	19	21	22
specific-threats	4	23	65	122	68	102	1 806	2	16	22
spyware-put	460	7	25	122	7	53	984	6	15	34
sql	3	16	19	534	48	67	7 754	5	14	17
telnet	2	19	402	402	144	510	510	4	7	7
tftp	1	478	478	478	1 907	1 907	1 907	4	4	4
voip	38	6	21	263	16	47	1 808	3	9	22
web-activex	474	69	69	136	185	211	286	24	30	32
web-cgi	10	11	22	531	42	108	5 177	9	15	18
web-frontpage	1	307	307	307	2 113	2 113	2 113	7	7	7
web-iis	12	6	63	1 065	5	139	17 523	5	15	28
web-php	16	6	20	55	5	46	491	5	11	17

Tabulka 6.1: Vlastnosti vzniklých automatů

Během testů se ukázalo především to, že některé redukční algoritmy jsou výrazně pomalejší než jiné. U těch pomalejších se proto podařilo získat výsledky pouze u souborů generujících automaty s přijatelnou velikostí. Pokud byly automaty moc velké (vzhledem ke schopnostem algoritmu), pak by výpočet neskončil v rozumném čase. Následující tabulka 6.2 shrnuje údaje o tom, pro který soubor a který redukční algoritmus se výpočet podařil. Úspěšný běh je označen celkovým časem běhu algoritmu pro příslušný soubor. Jednotkou času je sekunda.

soubor	eqR	eqL	pre	sat	bfs	simulace	bisimulace
backdoor	1,277	1,055	4177,708	-	0,745	1,883	1,347
chat	0,046	0,043	27,032	-	0,046	0,039	0,036
ddos	0,001	0,001	0,015	0,257	0,002	0,001	0,000
dos	0,392	0,063	1228,604	-	0,058	1,944	1,457
finger	0,000	0,000	0,000	0,010	0,001	0,000	0,000
ftp	52,460	0,754	-	-	1,223	478,163	941,318
imap	-	30,611	-	-	5,756	-	-
info	0,003	0,002	0,222	-	0,002	0,001	0,001
misc	424,136	1,248	-	-	4,446	11078,033	21925,636
mysql	0,366	0,046	241,513	-	0,087	2,018	3,383
netbios	12,648	0,243	-	-	0,163	109,839	199,079
nntp	-	8,424	-	-	7,281	-	-
p2p	0,010	0,010	8,146	-	0,006	0,006	0,006
policy	0,111	0,051	51,564	-	0,054	0,297	0,240
pop2	20,822	0,150	-	-	0,215	250,247	494,512
pop3	47,016	0,397	-	-	0,631	555,802	1097,936
rpc	0,001	0,001	0,005	0,013	0,001	0,000	0,000
shellcode	0,039	0,030	41,514	-	0,028	0,052	0,042
specific-threats	1,252	0,090	2144,468	-	0,265	12,114	21,252
spyware-put	4,168	3,674	-	-	3,038	9,657	5,261
sql	412,759	1,081	-	-	3,233	-	-
telnet	13,061	0,521	-	-	0,088	50,170	-
tftp	103,028	0,532	-	-	0,348	1853,801	-
voip	81,337	0,952	-	-	1,005	831,514	-
web-activex	45,943	29,737	-	-	29,257	120,654	44,227
web-cgi	645,825	2,438	-	-	3,125	-	-
web-frontpage	37,725	0,227	-	-	0,353	1209,685	1096,978
web-iis	-	7,992	-	-	17,521	-	-
web-php	0,171	0,082	119,472	-	0,103	0,548	0,879

Tabulka 6.2: Úspěšnost redukčních algoritmů

V následujících tabulkách jsou popsány výsledky jednotlivých redukčních algoritmů. Všechny tabulky obsahují tyto sloupce:

- soubor - první část názvu souboru (bez `.rules.pcre`)
- výrazy - počet regulárních výrazů v souboru
- redukce stavů - údaje o úspěšné redukci počtu stavů:
  - úspěch - počet úspěšně redukovaných automatů (vzhledem k počtu stavů) z celkového počtu automatů
  - průměr - průměrný počet stavů, o který je redukovaný automat menší než původní automat. Tento průměr se počítá pouze z úspěšných redukcí počtu stavů.
  - nejlepší - největší počet stavů, o který byl nějaký automat redukován
- redukce přechodů - údaje o úspěšné redukci počtu přechodů:
  - úspěch - počet úspěšně redukovaných automatů (vzhledem k počtu přechodů) z celkového počtu automatů
  - průměr - průměrný počet přechodů, o který je redukovaný automat menší než původní automat. Tento průměr se počítá pouze z úspěšných redukcí počtu přechodů.
  - nejlepší - největší počet přechodů, o který byl nějaký automat redukován
- čas - průměrný čas strávený výpočtem na jednom automatu. Tento průměr se počítá ze všech (i neúspěšných) pokusů.

Tabulka 6.3 shrnuje výsledky testů redukčního algoritmu *equivalences (pravá ekvivalence $\equiv_R$)*. Tento algoritmus se ukázal být celkem rychlý a účinný.

		redukce stavů			redukce přechodů			
soubor	výrazy	úspěch	průměr	nejlepší	úspěch	průměr	nejlepší	čas [s]
backdoor	154	80	1,725	20	80	20,425	121	0,008
chat	14	14	1,071	2	14	8,143	19	0,003
ddos	1	0	-	-	0	-	-	0,001
dos	3	3	8,333	16	3	51,667	127	0,131
finger	1	0	-	-	0	-	-	0,000
ftp	35	15	1,733	3	15	12,600	20	1,499
info	1	1	1,000	1	0	-	-	0,003
misc	17	14	4,857	9	14	28,429	60	24,949
mysql	3	3	5,667	6	3	32,667	54	0,122
netbios	16	12	3,000	14	1	98,000	98	0,791
p2p	1	0	-	-	0	-	-	0,010
policy	10	9	1,444	3	9	14,444	32	0,011
pop2	2	0	-	-	0	-	-	10,411
pop3	16	2	1,000	1	2	8,500	9	2,939
rpc	1	0	-	-	0	-	-	0,001
shellcode	3	3	5,667	7	3	33,000	41	0,013
specific-threats	4	4	2,000	3	3	15,333	33	0,313
spyware-put	460	423	1,603	35	418	21,859	176	0,009
sql	3	3	2,333	4	3	17,667	37	137,586
telnet	2	2	55,000	101	2	88,500	105	6,531
tftp	1	1	1,000	1	1	4,000	4	103,028
voip	38	20	5,450	43	17	39,412	289	2,140
web-activex	474	474	10,658	15	474	80,901	91	0,097
web-cgi	10	10	2,400	3	10	32,800	51	64,583
web-frontpage	1	0	-	-	0	-	-	37,725
web-php	16	15	4,533	6	15	19,867	42	0,011

Tabulka 6.3: Výsledky algoritmu *equivalences (pravá ekvivalence $\equiv_R$)*

Tabulka 6.4 shrnuje výsledky testů redukčního algoritmu *equivalences (levá ekvivalence $\equiv_L$)*. Tento algoritmus byl oproti ostatním hodně rychlý, což bylo zřejmě způsobeno tím, že většinou žádné stavy ke sloučení nenašel a proto rychle skončil. Pro redukci regulárních výrazů získaných z programu Snort se algoritmus ukázal být naprosto nevhodný.

		redukce stavů			redukce přechodů			
soubor	výrazy	úspěch	průměr	nejlepší	úspěch	průměr	nejlepší	čas [s]
backdoor	154	3	4,333	9	3	4,667	9	0,007
chat	14	1	1,000	1	1	1,000	1	0,003
ddos	1	0	-	-	0	-	-	0,001
dos	3	1	1,000	1	1	2,000	2	0,021
finger	1	0	-	-	0	-	-	0,000
ftp	35	0	-	-	0	-	-	0,022
imap	39	0	-	-	0	-	-	0,785
info	1	0	-	-	0	-	-	0,002
misc	17	7	1,286	3	7	2,571	5	0,073
mysql	3	0	-	-	0	-	-	0,015
netbios	16	0	-	-	0	-	-	0,015
nntp	12	0	-	-	0	-	-	0,702
p2p	1	0	-	-	0	-	-	0,010
policy	10	0	-	-	0	-	-	0,005
pop2	2	0	-	-	0	-	-	0,075
pop3	16	0	-	-	0	-	-	0,025
rpc	1	0	-	-	0	-	-	0,001
shellcode	3	0	-	-	0	-	-	0,010
specific-threats	4	1	1,000	1	1	2,000	2	0,023
spyware-put	460	5	7,600	16	5	9,400	16	0,008
sql	3	0	-	-	0	-	-	0,360
telnet	2	0	-	-	0	-	-	0,261
tftp	1	0	-	-	0	-	-	0,532
voip	38	5	7,000	10	5	7,400	12	0,025
web-activex	474	0	-	-	0	-	-	0,063
web-cgi	10	0	-	-	0	-	-	0,244
web-frontpage	1	0	-	-	0	-	-	0,227
web-iis	12	3	1,000	1	3	8,333	12	0,666
web-php	16	0	-	-	0	-	-	0,005

Tabulka 6.4: Výsledky algoritmu *equivalences (levá ekvivalence $\equiv_L$)*

Tabulka 6.5 shrnuje výsledky testů redukčního algoritmu *preorders*. Tento algoritmus dával o něco lepší výsledky než algoritmus *equivalences (pravá ekvivalence* $\equiv_R$*)*, byl však výrazně pomalejší. Proto také zvládl zpracovat menší množství souborů.

		redukce stavů			redukce přechodů			
soubor	výrazy	úspěch	průměr	nejlepší	úspěch	průměr	nejlepší	čas [s]
backdoor	154	80	1,850	29	80	20,563	121	27,128
chat	14	14	1,143	3	14	8,214	19	1,931
ddos	1	0	-	-	0	-	-	0,015
dos	3	3	8,667	16	3	52,000	127	409,535
finger	1	0	-	-	0	-	-	0,000
info	1	1	1,000	1	0	-	-	0,222
mysql	3	3	5,667	6	3	32,667	54	80,504
p2p	1	0	-	-	0	-	-	8,146
policy	10	9	7,444	28	9	34,889	115	5,156
rpc	1	0	-	-	0	-	-	0,005
shellcode	3	3	5,667	7	3	33,000	41	13,838
specific-threats	4	4	2,250	4	3	15,667	33	536,117
web-php	16	15	4,533	6	15	19,867	42	7,467

Tabulka 6.5: Výsledky algoritmu *preorders*

Tabulka 6.6 shrnuje výsledky testů redukčního algoritmu využívajícího *SAT solver*. Tento algoritmus se ukázal být naprosto nepoužitelným.Velikost kódovaného SAT problému totiž extrémně roste s velikostí automatu a SAT solver pak potřebuje obrovské množství času k vyřešení problému. V praxi se algoritmus dá použít na automaty o velikosti maximálně zhruba kolem 10 stavů. V článku [8] jsem později zjistil, že jeho autoři ho testovali na automatech s maximálně 13 stavy a 5 symboly abecedy. Na regulární výrazy používané v knihovně Netbench je tedy tento algoritmus nepoužitelný.

		redukce stavů			redukce přechodů			
soubor	výrazy	úspěch	průměr	nejlepší	úspěch	průměr	nejlepší	čas [s]
ddos	1	0	-	-	0	-	-	0,257
finger	1	0	-	-	0	-	-	0,010
rpc	1	0	-	-	0	-	-	0,013

Tabulka 6.6: Výsledky algoritmu využívajícího *SAT solver*

Tabulka 6.7 shrnuje výsledky testů redukčního algoritmu *prohledávání do šířky*. Tento algoritmus byl suverénně nejrychlejší a zároveň s nejhoršími výsledky. Většinou se mu nepodařilo zmenšit počet stavů v automatech a pokud ano, tak se někdy dokonce stalo, že vytvořil automat s větším počtem přechodů než měl předtím. Tyto záporné hodnoty změny počtu přechodů do tabulky zahrnuty nejsou. Autoři algoritmu tvrdili, že ho lze použít pro redukci počtu stavů i na automaty bez epsilon-přechodů. Toto se však na regulárních výrazech používaných v knihovně Netbench nepotvrdilo.

		redukce stavů			redukce přechodů			
soubor	výrazy	úspěch	průměr	nejlepší	úspěch	průměr	nejlepší	čas [s]
backdoor	154	3	3,667	8	1	1,000	1	0,005
chat	14	1	1,000	1	1	1,000	1	0,003
ddos	1	0	-	-	0	-	-	0,002
dos	3	1	1,000	1	1	2,000	2	0,019
finger	1	0	-	-	0	-	-	0,001
ftp	35	0	-	-	0	-	-	0,035
imap	39	0	-	-	0	-	-	0,148
info	1	0	-	-	0	-	-	0,002
misc	17	1	2,000	2	0	-	-	0,262
mysql	3	0	-	-	0	-	-	0,029
netbios	16	0	-	-	0	-	-	0,010
nntp	12	0	-	-	0	-	-	0,607
p2p	1	0	-	-	0	-	-	0,006
policy	10	0	-	-	0	-	-	0,005
pop2	2	0	-	-	0	-	-	0,108
pop3	16	0	-	-	0	-	-	0,039
rpc	1	0	-	-	0	-	-	0,001
shellcode	3	0	-	-	0	-	-	0,009
specific-threats	4	1	1,000	1	1	2,000	2	0,066
spyware-put	460	4	8,000	15	0	-	-	0,007
sql	3	0	-	-	0	-	-	1,078
telnet	2	0	-	-	0	-	-	0,044
tftp	1	0	-	-	0	-	-	0,348
voip	38	5	5,600	9	3	5,333	7	0,026
web-activex	474	0	-	-	0	-	-	0,062
web-cgi	10	0	-	-	0	-	-	0,313
web-frontpage	1	0	-	-	0	-	-	0,353
web-iis	12	0	-	-	0	-	-	1,460
web-php	16	0	-	-	0	-	-	0,006

Tabulka 6.7: Výsledky algoritmu *prohledávání do šířky*

Tabulka 6.8 shrnuje výsledky testů Netbenchového redukčního algoritmu *simulation*. Tento algoritmus dával stejné výsledky jako algoritmus *equivalences (pravá ekvivalence $\equiv_R$)*. Je tedy zřejmě založen na podobném principu. Algoritmus *simulation* však často spotřeboval více času než algoritmus *equivalences (pravá ekvivalence $\equiv_R$)*.

		redukce stavů			redukce přechodů			
soubor	výrazy	úspěch	průměr	nejlepší	úspěch	průměr	nejlepší	čas [s]
backdoor	154	80	1,725	20	80	20,425	121	0,012
chat	14	14	1,071	2	14	8,143	19	0,003
ddos	1	0	-	-	0	-	-	0,001
dos	3	3	8,333	16	3	51,667	127	0,648
finger	1	0	-	-	0	-	-	0,000
ftp	35	15	1,733	3	15	12,600	20	13,662
info	1	1	1,000	1	0	-	-	0,001
misc	17	14	4,857	9	14	28,429	60	651,649
mysql	3	3	5,667	6	3	32,667	54	0,673
netbios	16	12	3,000	14	1	98,000	98	6,865
p2p	1	0	-	-	0	-	-	0,006
policy	10	9	1,444	3	9	14,444	32	0,030
pop2	2	0	-	-	0	-	-	125,124
pop3	16	2	1,000	1	2	8,500	9	34,738
rpc	1	0	-	-	0	-	-	0,000
shellcode	3	3	5,667	7	3	33,000	41	0,017
specific-threats	4	4	2,000	3	3	15,333	33	3,029
spyware-put	460	423	1,603	35	418	21,859	176	0,021
telnet	2	2	55,000	101	2	88,500	105	25,085
tftp	1	1	1,000	1	1	4,000	4	1853,801
voip	38	20	5,450	43	17	39,412	289	21,882
web-activex	474	474	10,658	15	474	80,901	91	0,255
web-frontpage	1	0	-	-	0	-	-	1209,685
web-php	16	15	4,533	6	15	19,867	42	0,034

Tabulka 6.8: Výsledky Netbenchového algoritmu *simulation*

Tabulka 6.9 shrnuje výsledky testů Netbenchového redukčního algoritmu *bisimulation*. Tento algoritmus dával většinou stejné výsledky jako algoritmy *simulation* a *equivalences (pravá ekvivalence* $\equiv_R$*)*. Někdy byl o něco málo lepší. Pravděpodobně tedy opět vychází z podobných principů jako dva zmíněné algoritmy. Algoritmus *bisimulation* byl ještě o něco pomalejší než zmíněné dva algoritmy.

		redukce stavů			redukce přechodů			
soubor	výrazy	úspěch	průměr	nejlepší	úspěch	průměr	nejlepší	čas [s]
backdoor	154	97	1,680	20	97	17,186	121	0,009
chat	14	14	1,071	2	14	8,143	19	0,003
ddos	1	0	-	-	0	-	-	0,000
dos	3	3	8,333	16	3	51,667	127	0,486
finger	1	0	-	-	0	-	-	0,000
ftp	35	17	1,647	3	17	11,706	20	26,895
info	1	1	1,000	1	0	-	-	0,001
misc	17	14	4,929	9	14	28,500	60	1289,743
mysql	3	3	5,667	6	3	32,667	54	1,128
netbios	16	12	3,000	14	1	98,000	98	12,442
p2p	1	0	-	-	0	-	-	0,006
policy	10	9	1,444	3	9	14,444	32	0,024
pop2	2	0	-	-	0	-	-	247,256
pop3	16	3	1,000	1	3	7,667	9	68,621
rpc	1	0	-	-	0	-	-	0,000
shellcode	3	3	5,667	7	3	33,000	41	0,01
specific-threats	4	4	2,250	4	3	20,333	48	5,313
spyware-put	460	423	1,622	35	420	21,852	176	0,011
web-activex	474	474	10,658	15	474	80,901	91	0,093
web-frontpage	1	1	1,000	1	1	7,000	7	1096,978
web-php	16	15	4,533	6	15	19,867	42	0,055

Tabulka 6.9: Výsledky Netbenchového algoritmu *bisimulation*

6.2 Testy se sloučenými automaty

Ve druhé fázi testování byly provedeny testy se sloučenými automaty, kde každý automat vznikl sloučením všech regulárních výrazů ze vstupního souboru. Tím samozřejmě narostla velikost automatů, takže z časových důvodů byly pro testy vybrány jen některé ze souborů použitých v první fázi. Také byly samozřejmě vynechány soubory, které obsahují pouze jeden regulární výraz. Vlastnosti vzniklých sloučených automatů shrnuje tabulka 6.10, která obsahuje tyto údaje:

- soubor - první část názvu souboru (bez `.rules.pcre`)
- počet stavů - počet stavů sloučeného automatu
- počet přechodů - počet přechodů sloučeného automatu
- počet symbolů - počet symbolů sloučeného automatu

soubor	počet stavů	počet přechodů	počet symbolů
backdoor	3 800	33 489	99
chat	186	1 243	38
dos	182	1 911	33
ftp	1 794	64 045	43
imap	5 770	118 806	37
misc	1 266	59 097	66
mysql	130	1 499	23
netbios	621	10 205	18
nntp	2 573	59 386	25
policy	205	3 265	43
pop2	268	1 804	7
pop3	905	15 620	20
shellcode	95	330	23
specific-threats	244	4 305	34
sql	567	13 841	28
telnet	420	720	10
voip	1 851	60 609	51
web-cgi	1 306	42 005	46
web-iis	2 381	110 618	67
web-php	322	4 250	57

Tabulka 6.10: Vlastnosti vzniklých sloučených automatů

I zde samozřejmě opět došlo k tomu, že pomalejší redukční algoritmy dokázaly v rozumném čase zpracovat jen některé soubory. Tím spíš, že velikost zpracovávaných automatů zde oproti první fázi testů výrazně nasrostla. Tabulka 6.11 tedy opět shrnuje údaje o tom, pro který soubor a který redukční algoritmus se výpočet podařil. Úspěšný běh je označen celkovým časem běhu algoritmu pro příslušný soubor. Jednotkou času je sekunda.

soubor	eqR	eqL	pre	sat	bfs	simulace	bisimulace
backdoor	-	716,553	-	-	95,323	-	-
chat	0,341	0,262	-	-	0,255	1,568	1,536
dos	1,552	0,266	-	-	0,404	16,452	13,492
ftp	-	26,919	-	-	242,078	-	-
imap	-	-	-	-	967,581	-	-
misc	3122,240	13,360	-	-	572,082	-	-
mysql	0,749	0,102	1921,683	-	0,183	6,199	9,606
netbios	136,509	2,065	-	-	2,973	-	-
nntp	-	76,670	-	-	221,262	-	-
policy	1,027	0,386	-	-	0,897	11,315	12,444
pop2	23,803	0,177	-	-	0,273	297,071	585,660
pop3	388,700	3,168	-	-	20,470	-	-
shellcode	0,087	0,064	441,738	-	0,029	0,139	0,139
specific-threats	3,487	0,309	-	-	5,249	50,329	73,475
sql	678,340	1,198	-	-	10,552	-	-
telnet	25,209	1,065	-	-	0,117	62,943	-
voip	-	38,315	-	-	218,750	-	-
web-cgi	2641,547	12,046	-	-	87,159	-	-
web-iis	-	73,704	-	-	803,411	-	-
web-php	5,189	1,268	-	-	1,461	23,385	20,355

Tabulka 6.11: Úspěšnost redukčních algoritmů na sloučených automatech

V následujících tabulkách jsou popsány výsledky jednotlivých redukčních algoritmů na sloučených automatech. Všechny tabulky obsahují tyto sloupce:

- soubor - první část názvu souboru (bez `.rules.pcre`)
- redukce stavů - počet stavů, o který je redukovaný automat menší než původní automat
- redukce přechodů - počet přechodů, o který je redukovaný automat menší než původní automat
- čas - čas strávený výpočtem

Tabulka 6.12 shrnuje výsledky testů redukčního algoritmu *equivalences (pravá ekvivalence* $\equiv_R$*)*. Algoritmus byl i na sloučených automatech stále celkem úspěšný a přiměřeně rychlý.

soubor	redukce stavů	redukce přechodů	čas [s]
chat	56	596	0,341
dos	24	532	1,552
misc	434	23 234	3 122,240
mysql	16	62	0,749
netbios	269	3 840	136,509
policy	14	90	1,027
pop2	1	0	23,803
pop3	537	9 691	388,700
shellcode	18	44	0,087
specific-threats	9	72	3,487
sql	7	32	678,340
telnet	108	188	25,209
web-cgi	43	749	2 641,547
web-php	128	464	5,189

Tabulka 6.12: Výsledky algoritmu *equivalences (pravá ekvivalence* $\equiv_R$*)*

Tabulka 6.13 shrnuje výsledky testů redukčního algoritmu *equivalences (levá ekvivalence $\equiv_L$)*. Oproti samostatným automatům, kde skončil velkým neúspěchem, dokázal algoritmus na sloučených automatech někdy dosáhnout i lepší redukce než algoritmus *equivalences (pravá ekvivalence $\equiv_R$)* a byl také stále výrazně rychlejší.

soubor	redukce stavů	redukce přechodů	čas [s]
backdoor	413	10 871	716,553
chat	38	445	0,262
dos	2	70	0,266
ftp	93	311	26,919
misc	115	1 676	13,360
mysql	28	123	0,102
netbios	249	4 343	2,065
nntp	11	11	76,670
policy	42	1 466	0,386
pop2	4	4	0,177
pop3	18	18	3,168
shellcode	24	133	0,064
specific-threats	1	2	0,309
sql	1	56	1,198
telnet	1	20	1,065
voip	146	518	38,315
web-cgi	42	1 213	12,046
web-iis	14	1 459	73,704
web-php	25	1 622	1,268

Tabulka 6.13: Výsledky algoritmu *equivalences (levá ekvivalence $\equiv_L$)*

Tabulka 6.14 shrnuje výsledky testů redukčního algoritmu *preorders*. Na sloučených automatech, které mají větší velikost než samostatné automaty, se už naplno projevila větší časová náročnost algoritmu. Našel sice opět lepší řešení než předchozí dva algoritmy, dokázal však v rozumném čase zpracovat pouze dva soubory s nejmenšími automaty.

soubor	redukce stavů	redukce přechodů	čas [s]
mysql	33	168	1 921,683
shellcode	36	152	441,738

Tabulka 6.14: Výsledky algoritmu *preorders*

Algoritmus využívající *SAT solver* již nešel na sloučené automaty vzhledem k jejich velikosti vůbec použít.

Tabulka 6.15 shrnuje výsledky testů redukčního algoritmu *prohledávání do šířky*. Tento algoritmus opět patřil k nejrychlejším a zpracoval skoro všechny soubory. Dokonce se mu i dařily nějaké redukce počtu stavů, ovšem stále se mu stávalo, že přitom zvýšil počet přechodů. To je v tabulce označeno zápornými hodnotami.

soubor	redukce stavů	redukce přechodů	čas [s]
backdoor	256	-7 515	95,323
chat	27	-43	0,255
dos	0	0	0,404
ftp	48	-3 600	242,078
imap	202	3 120	967,581
misc	0	0	572,082
mysql	24	34	0,183
netbios	236	3 282	2,973
nntp	11	11	221,262
policy	4	-200	0,897
pop2	3	-3	0,273
pop3	0	0	20,470
shellcode	23	109	0,029
specific-threats	0	0	5,249
sql	0	0	10,552
telnet	0	0	0,117
voip	107	-4 485	218,750
web-cgi	29	609	87,159
web-iis	0	0	803,411
web-php	0	0	1,461

Tabulka 6.15: Výsledky algoritmu *prohledávání do šířky*

Tabulka 6.16 shrnuje výsledky testů Netbenchového redukčního algoritmu *simulation*. Ten byl opět pomalejší než algoritmus *equivalences (pravá ekvivalence* $\equiv_R$*)* a na rozdíl od testů na samostatných automatech dával tentokrát i horší výsledky.

soubor	redukce stavů	redukce přechodů	čas [s]
chat	9	275	1,568
dos	23	532	16,452
mysql	14	62	6,199
policy	4	89	11,315
pop2	0	0	297,071
shellcode	14	34	0,139
specific-threats	7	72	50,329
telnet	108	188	62,943
web-php	53	341	23,385

Tabulka 6.16: Výsledky Netbenchového algoritmu *simulation*

Tabulka 6.17 shrnuje výsledky testů Netbenchového redukčního algoritmu *bisimulation*. Ten dával tentokrát stejné výsledky jako algoritmus *simulation*, byl ale o něco pomalejší.

soubor	redukce stavů	redukce přechodů	čas [s]
chat	9	275	1,536
dos	23	532	13,492
mysql	14	62	9,606
policy	4	89	12,444
pop2	0	0	585,660
shellcode	14	34	0,139
specific-threats	8	104	73,475
web-php	53	341	20,355

Tabulka 6.17: Výsledky Netbenchového algoritmu *bisimulation*

6.3 Shrnutí výsledků testů

Tato kapitola shrnuje výsledky všech předchozích testů redukčních algoritmů. Nejdříve jsou uvedeny shrnující tabulky 6.18 a 6.19., které obsahují tyto sloupce:

- algoritmus - použitý redukční algoritmus
- redukce stavů - údaje o úspěšné redukci počtu stavů:
  - úspěch - celkový počet úspěšně redukovaných automatů (vzhledem k počtu stavů).
  - stavy - celkový součet počtů stavů, o které jsou redukované automaty menší než původní automaty.
- redukce přechodů - údaje o úspěšné redukci počtu přechodů:
  - úspěch - celkový počet úspěšně redukovaných automatů (vzhledem k počtu přechodů).
  - přechody - celkový součet počtů přechodů, o které jsou redukované automaty menší než původní automaty.
- čas - celkový čas strávený výpočtem na všech automatech (i neúspěšné pokusy o redukci).

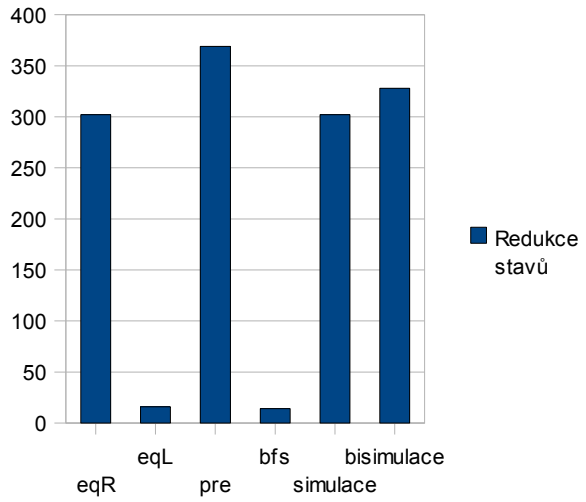
Tabulka 6.18 shrnuje výsledky testů na samostatných automatech. Z porovnání je vynechán algorimus využívající *SAT solver*, který zvládl zpracovat pouze 3 soubory. Ostatní algoritmy jsou porovnány pouze na těch souborech, na kterých všechny skončily úspěchem. Je to těchto 13 souborů:

```
backdoor.rules.pcre
chat.rules.pcre
ddos.rules.pcre
dos.rules.pcre
finger.rules.pcre
info.rules.pcre
mysql.rules.pcre
p2p.rules.pcre
policy.rules.pcre
rpc.rules.pcre
shellcode.rules.pcre
specific-threats.rules.pcre
web-php.rules.pcre
```

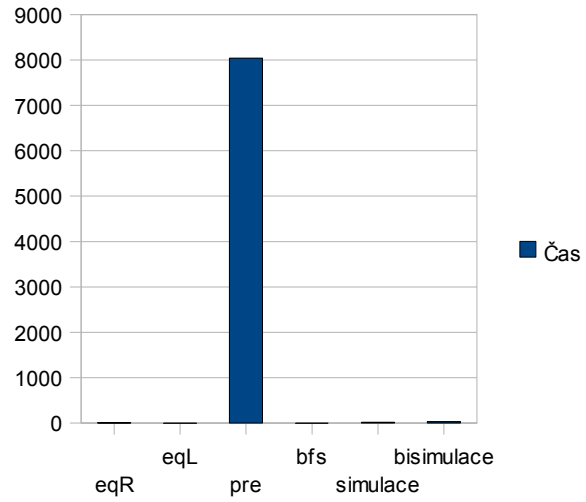
	redukce stavů		redukce přechodů		
algoritmus	úspěch	stavy	úspěch	přechody	čas [s]
eqR	132	302	130	2 574	3,669
eqL	6	16	6	19	1,474
pre	132	369	130	2 772	8 040,263
bfs	6	14	4	6	1,398
simulace	132	302	130	2 574	18,903
bisimulace	149	328	147	2 622	28,643

Tabulka 6.18: Shrnutí výsledků testů na samostatných automatech

Graf 6.1 zobrazuje porovnání redukce počtu stavů z tabulky 6.18. Graf 6.2 zobrazuje porovnání času výpočtu z tabulky 6.18.



Graf 6.1: Porovnání redukce stavů na samostatných automatech



Graf 6.2: Porovnání doby výpočtu na samostatných automatech

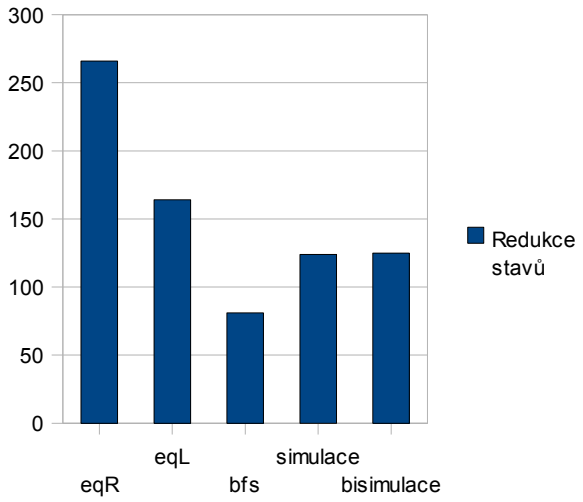
Tabulka 6.19 shrnuje výsledky testů na sloučených automatech. Z porovnání je vynechán algoritmus *preorders*, který zvládl zpracovat pouze 2 soubory, a algoritmus využivající *SAT solver*, který nezpracoval žádný soubor. Ostatní algoritmy jsou porovnány pouze na těch souborech, na kterých všechny skončily úspěchem. Je to těchto 8 souborů:

```
chat.rules.pcre
dos.rules.pcre
mysql.rules.pcre
policy.rules.pcre
pop2.rules.pcre
shellcode.rules.pcre
specific-threats.rules.pcre
web-php.rules.pcre
```

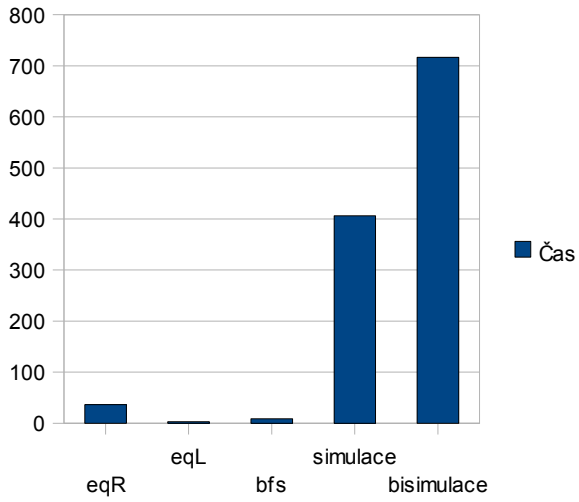
	redukce stavů		redukce přechodů		
algoritmus	úspěch	stavy	úspěch	přechody	čas [s]
eqR	8	266	7	1 860	36,235
eqL	8	164	8	3 865	2,834
bfs	5	81	2	143	8,751
simulace	7	124	7	1 405	406,458
bisimulace	7	125	7	1 437	716,703

Tabulka 6.19: Shrnutí výsledků testů na sloučených automatech

Graf 6.3 zobrazuje porovnání redukce počtu stavů z tabulky 6.19. Graf 6.4 zobrazuje porovnání času výpočtu z tabulky 6.19.



Graf 6.3: Porovnání redukce stavů na sloučených automatech



Graf 6.4: Porovnání doby výpočtu na sloučených automatech

Experimenty s redukčními algoritmy odhalily různé jejich vlastnosti. Jako nejrychlejší a zároveň nejméně účinné se ukázaly být algoritmy *prohledávání do šířky* a *equivalences (levá ekvivalence* $\equiv_L$*)*. Na samostatných automatech byly prakticky nepoužitelné, na sloučených automatech už nějakou úspěšnost měly. Nevýhodou algoritmu *prohledávání do šířky* však bylo to, že někdy sice snížil počet stavů, ale zároveň zvýšil počet přechodů automatu.

Algoritmus využívající *SAT solver* se ukázal jako naprosto nepoužitelný na jakékoliv jiné než úplně nejmenší automaty. A i na nich se mu žádná redukce počtu stavů nepodařila.

Algoritmus *preorders* ukázal své kvality i slabiny. Dokázal dosáhnout nejlepších redukcí, byl ale výrazně pomalý.

Algoritmy *equivalences (pravá ekvivalence* $\equiv_R$*)*, *simulation* a *bisimulation* jsou zřejmě založeny na podobných principech a proto také dávaly podobné výsledky. Nejrychlejší z nich byl algoritmus *equivalences (pravá ekvivalence* $\equiv_R$*)*, druhý byl algoritmus *simulation* a nejpomalejší byl algoritmus *bisimulation*.

Během testů se ukázalo, že redukce počtu stavů obvykle vede také k redukci počtu přechodů automatu. Jediný algoritmus, který se chová jinak, je algoritmus *prohledávání do šířky*, který někdy počet přechodů automatu dokonce zvýšil. Dále se také ukázalo, že sloučené automaty se dají lépe redukovat. Je to pravděpodobně způsobeno tím, že jsou o dost větší než samostatné automaty, a tím pádem nabízejí větší prostor k redukci.

Celkově se jako nejlepší algoritmus z hlediska redukce počtu stavů ukázal algoritmus *preorders*, který však byl velmi pomalý. Jako nejpoužitelnější algoritmus bych označil algoritmus *equivalences (pravá ekvivalence* $\equiv_R$*)*, který dával velmi dobré výsledky v rozumném čase.

6.4 Časová náročnost SAT solveru

Tato kapitola ilustruje, proč algoritmus využívající *SAT solver* nedokázal zpracovat skoro žádné konečné automaty kromě těch úplně nejmenších. Časovou náročnost běhu algoritmu si ukážeme na příkladu redukce NKA, který má 15 stavů, 15 přechodů a 12 symbolů abecedy. (Konkrétně jde o poslední regulární výraz ze souboru `chat.rules.pcre`.) Redukční algoritmus hledá NKA, který má velikost v rozmezí určeném velikostí vstupního NKA a z něj vytvořeného DKA. V tomto konkrétním příkladě se hledá NKA o velikosti od 4 do 11 stavů. Pokud SAT solver NKA s n stavy nenalezne, pokračuje algoritmus v dalším cyklu hledáním NKA s n+1 stavy. Tabulka 6.20 shrnuje údaje o jednotlivých cyklech algoritmu a obsahuje tyto sloupce:

- stavy - počet stavů hledaného NKA
- proměnné - počet proměnných SAT problému
- klauzule - počet klauzulí SAT problému
- čas - doba běhu SAT solveru

stavy	proměnné	klauzule	čas [s]
4	392	2 885	0,004
5	610	4 506	0,006
6	876	6 487	0,074
7	1 190	8 828	0,895
8	1 552	11 529	22,763
9	1 962	14 590	515,772
10	2 420	18 011	9 819,818
11	2 926	21 792	mnoho

Tabulka 6.20: Ukázka běhu redukčního algoritmu využívajícího *SAT solver*

Z tabulky 6.20 je jasně vidět, že doba běhu SAT solveru exponenciálně roste s počtem stavů hledaného NKA. Prakticky je pak algoritmus schopen zpracovat vstupní NKA o velikosti maximálně zhruba kolem 10 stavů. Pro větší automaty pak již celkový čas neúnosně narůstá.

7 Závěr

V rámci této diplomové práce byly nastudovány různé metody pro redukci nedeterministických konečných automatů. Ukázalo se, že většina článků je spíše teoretického rázu, a prakticky použitelných algoritmů mnoho není. Pro implementaci bylo vybráno několik algoritmů založených na různých principech. Implementované algoritmy byly otestovány na regulárních výrazech používaných v knihovně Netbench. Pět implementovaných redukčních algoritmů bylo také porovnáno se dvěma redukčními algoritmy z knihovny Netbench. Výsledky experimentů jsou následující:

Algoritmus využívající *SAT solver*, který se zdál být nadějný, nakonec dopadl nejhůř, protože dokázal zpracovat pouze velmi malé automaty, maximálně zhruba kolem 10 stavů. Velikost SAT problému extrémně narůstá s počtem stavů automatu, což činí algoritmus prakticky nepoužitelným, alespoň co se týká regulárních výrazů používaných v knihovně Netbench. Algoritmus *prohledávání do šířky* dopadl také velmi špatně. Byl sice nejrychlejší, ale málokdy dokázal úspěšně redukovat počet stavů automatu. A navíc někdy dokonce i zvýšil počet přechodů automatu.

O něco lépe dopadl algoritmus *equivalences (levá ekvivalence $\equiv_L$)*, který sice příliš nedokázal redukovat počty stavů samostatných automatů, ale zaznamenal jistou úspěšnost na sloučených automatech. Celkově byl ale o dost horší než algoritmus *equivalences (pravá ekvivalence $\equiv_R$)*, přestože je založený na úplně stejném principu. Automaty vzniklé z regulárních výrazů používaných v knihovně Netbench tedy zřejmě mají nějaké vlastnosti, kvůli kterým se pro jejich redukci hodí daleko lépe algoritmus *equivalences (pravá ekvivalence $\equiv_R$)*.

Jako nejúčinnější z hlediska redukce počtu stavů se ukázal algoritmus *preorders*. Jeho nevýhodou však je velké množství času potřebného k výpočtu. Použití tohoto algoritmu lze tedy doporučit, pokud potřebujeme především co nejvíce redukovat daný automat a nezáleží nám tolik na čase, který k tomu bude potřeba.

Obecně nejpoužitelnějším algoritmem se stal algoritmus *equivalences (pravá ekvivalence $\equiv_R$)*. Ten dával velmi dobré výsledky v krátkém čase. Lze ho tedy doporučit pro obecné použití, kdy nám záleží jak na redukci, tak i na čase. Netbenchové algoritmy *simulation* a *bisimulation* jsou zřejmě založeny na podobném principu, protože dávaly většinou podobné výsledky jak algoritmus *equivalences (pravá ekvivalence $\equiv_R$)*. Doba jejich běhu však byla obvykle delší.

Jako nejlepší se tedy nakonec ukázaly algoritmy založené na příbuzných principech - *equivalences* a *preorders*. Co se týče dalšího zlepšení vlastností těchto algoritmů, mohlo by se pokračování práce v této oblasti zabývat především implementací vylepšených verzí těchto algoritmů. Popisy vylepšených verzí lze nalézt v článcích [5] a [6]. Další možností by mohlo být zkombinování různých přístupů do jednoho vylepšeného algoritmu.

Literatura

[1] Ilie, Lucian, Yu, Sheng. *Algorithms for Computing Small NFAs.* Proceedings of the 27th International Symposium of Mathematical Foundations of Computer Science: 328–340, Springer-Verlag Berlin, 2002.

[2] Ilie, Lucian, Yu, Sheng. *Reducing NFAs by invariant equivalences.* Theoretical computer science 306, Issue 1-3, Elsevier Science Publishers, 2003.

[3] Champarnaud, Jean-Marc, Coulon Fabien. *NFA reduction algorithms by means of regular inequalities.* Theoretical computer science 327: 241-253, Elsevier Science Publishers, 2004.

[4] Champarnaud, Jean-Marc, Coulon Fabien. *NFA reduction algorithms by means of regular inequalities - correction.* 2005.

[5] Ilie, Lucian, Navarro, Gonzalo, Yu, Sheng. *On NFA reductions.* Lecture Notes in Computer Science 3113: 112-124, Springer-Verlag Berlin, 2004.

[6] Ilie, Lucian, Solis-Oba, Roberto, Yu, Sheng. *Reducing the size of NFAs by using equivalences and preorders.* Lecture Notes in Computer Science 3537: 310-321, Springer-Verlag Berlin, 2005.

[7] Becchi, Michaela, Crowley, Patrick. *Efficient Regular Expression Evaluation: Theory to Practice.* Proceedings of the 4th ACM/IEEE Symposium of Architectures for Networking and Communications Systems, ACM New York, 2008.

[8] Geldenhuys, Jaco, van der Merwe, Brink, van Zijl, Lynette. *Reducing Nondeterministic Finite Automata with SAT Solvers.* Lecture Notes in Computer Science 6062: 81-92, Springer-Verlag Berlin, 2010.

[9] Kameda, Tsunehiko, Weiner, Peter. *On the State Minimization of Nondeterministic Finite Automata.* IEEE Transactions on Computers 19, Issue 7, IEEE Computer Society Washington, 1970.

[10] Matz, Oliver, Potthoff, Andreas. *Computing Small Nondeterministic Finite Automata.* Workshop on Tools and Algorithms for the Construction and Analysis of Systems, 1995.

[11] Gouveia, Hugo, Moreira, Nelma, Reis, Rogério. *Small NFAs from Regular Expressions: Some Experimental results.* 2010.

[12] Melnikov, Boris Feliksovich, Tsyganov, Andrey Vladimirovich, Bulychov, Oleg Ivanovich. *A Multi-heuristic Algorithmic Skeleton for Hard Combinatorial Optimization Problems.* Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization 1, IEEE Computer Society Washington, 2009.

[13] Almeida, Marco, Moreira, Nelma, Reis, Rogério. *On the performance of automata minimization algorithms.* Universidade do Porto, 2007.

[14] Melnikov, Boris Feliksovich, Melnikova, A. A.. *Edge-minimization of non-deterministic finite automata.* Journal of Applied Mathematics and Computing 8, Number 3: 469-479, Korean Society for Computational & Applied Mathematics and Korean SIGCAM, 2001.

[15] Melnikov, Boris Feliksovich. *Multiheuristic approach to discrete optimization problems.* Cybernetics and System Analysis 42, Number 3: 335-341, Kluwer Academic Publishers, 2006.

[16] Melnikov, Boris Feliksovich. *A new algorithm of the state-minimization for the nondeterministic finite automata.* Journal of Applied Mathematics and Computing 6, Number 2: 277-290, Korean Society for Computational & Applied Mathematics and Korean SIGCAM, 1999.

[17] Melnikov, Boris Feliksovich. *Once more about the state-minimization for the nondeterministic finite automata.* Journal of Applied Mathematics and Computing 7, Number 3: 655-662, Korean Society for Computational & Applied Mathematics and Korean SIGCAM, 2000.

[18] Sengoku, Hiroaki. *Minimization of Nondeterministic Finite Automata.* Master Thesis, Department of Information Science, Faculty of Engineering, Kyoto University, 1992.

[19] Polák, Libor. *Minimalization of NFA Using the Universal Automaton.* Lecture Notes in Computer Science 3317: 325-326, Springer-Verlag Berlin, 2005.

[20] Gramlich, Gregor, Schnitger, Georg. *Minimizing NFA's and Regular Expressions.* Lecture Notes in Computer Science 3404: 399-411, Springer-Verlag Berlin, 2005.
[21] Gruber, Hermann, Holzer, Markus. *Computational Complexity of NFA Minimization for Finite and Unary Languages.* 2007.
[22] Grunsky, Igor, Kurganskyy, Oleksiy, Potapov, Igor. *On a Maximal NFA Without Mergible States.* Lecture Notes in Computer Science 3967: 202-210, Springer-Verlag Berlin, 2006.
[23] Holzer, Markus, Kutrib Martin. *Nondeterministic Finite Automata - Recent Results on the Descriptional and Computational Complexity.* Lecture Notes in Computer Science 5148: 1-16, Springer-Verlag Berlin, 2008.
[24] Holzer, Markus, Kutrib, Martin. *Descriptional and Computational Complexity of Finite Automata.* Proceedings of the 3rd International Conference on Language and Automata Theory and Applications: 23-42, Springer-Verlag Berlin, 2009.
[25] Câmpeanu, Cezar, Sântean, Nicolae, Yu, Sheng. *Mergible States in Large NFA.* Theoretical Computer Science - Insightful theory 330, Issue 1: 23-34, Elsevier Science Publishers, 2006.
[26] Câmpeanu, Cezar, Sântean, Nicolae, Yu, Sheng. *Large NFA Without Mergible States.* Journal of Automata, Languages and Combinatorics, Otto-von-Guericke-Universität Magdeburg, 2005.
[27] Matz, Oliver, Miller, Axel, Potthoff, Andreas, Thomas, Wolfgang, Valkema, Erich. *Report on the Program AMoRE.* Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel, 1995.
[28] Xing, Guangming. *Minimized Thompson NFA.* 2004.
[29] Jiang, Tao, Ravikumar, Bala. *Minimal NFA problems are hard.* Lecture Notes in Computer Science 510: 629-640, Springer-Verlag Berlin, 1991.
[30] Češka, Milan, Vojnar, Tomáš, Smrčka, Aleš. *Teoretická informatika - Studijní opora.* 2009.
[31] *Equivalence relation (Wikipedia)* [online]. Poslední modifikace: 31. prosince 2010. [cit. 2011-01-08]. Dostupné na URL: <http://en.wikipedia.org/wiki/Equivalence_relation>.
[32] *Preorder (Wikipedia)* [online]. Poslední modifikace: 20. října 2010. [cit. 2011-01-08]. Dostupné na URL: <http://en.wikipedia.org/wiki/Preorder>.
[33] *Kvaziuspořádání (Wikipedie)* [online]. Poslední modifikace: 8. září 2010. [cit. 2011-01-08]. Dostupné na URL: <http://cs.wikipedia.org/wiki/Kvaziuspo%C5%99%C3%A1d%C3%A1n%C3%AD>.
[34] Paige, Robert, Tarjan, Robert Endre. Three Partition Refinement Algorithms. SIAM Journal on Computing 16, Issue 6: 973-989, Society for Industrial and Applied Mathematics Philadelphia, 1987.
[35] *The international SAT Competitions web page* [online]. [cit. 2011-05-14]. Dostupné na URL: <http://www.satcompetition.org>.
[36] *Boolean satisfiability problem (Wikipedia)* [online]. Poslední modifikace: 18. dubna 2011. [cit. 2011-05-14]. Dostupné na URL: <http://en.wikipedia.org/wiki/Boolean_satisfiability_problem>.
[37] *Conjunctive normal form (Wikipedia)* [online]. Poslední modifikace: 21. dubna 2011. [cit. 2011-05-14]. Dostupné na URL: <http://en.wikipedia.org/wiki/Conjunctive_normal_form>.
[38] *Matrix addition (Wikipedia)* [online]. Poslední modifikace: 9. srpna 2010. [cit. 2011-05-14]. Dostupné na URL: <http://en.wikipedia.org/wiki/Matrix_addition>.
[39] *Matrix multiplication (Wikipedia)* [online]. Poslední modifikace: 25. dubna 2011. [cit. 2011-05-14]. Dostupné na URL: <http://en.wikipedia.org/wiki/Matrix_multiplication>.
[40] *Transpose (Wikipedia)* [online]. Poslední modifikace: 31. ledna 2011. [cit. 2011-05-14]. Dostupné na URL: <http://en.wikipedia.org/wiki/Transpose>.
[41] *Accelerated Network Technologies - Netbench* [online]. [cit. 2011-05-14]. Dostupné na URL: <http://merlin.fit.vutbr.cz/ant/netbench/index.html>.
[42] *MiniSat* [online]. [cit. 2011-05-14]. Dostupné na URL: <http://minisat.se/MiniSat.html>.
[43] *MiniSat (download)* [online]. [cit. 2011-05-14]. Dostupné na URL: <http://minisat.se/downloads/MiniSat_v1.14.2006-Aug-29.src.zip>.

[44] *zChaff* [online]. [cit. 2011-05-14]. Dostupné na URL: <http://www.princeton.edu/~chaff/zchaff.html>.
[45] *zChaff (download)* [online]. [cit. 2011-05-14]. Dostupné na URL: <http://www.princeton.edu/~chaff/zchaff/zchaff.2004.5.13.tar.gz>.

Seznam příloh

Příloha 1. CD se zdrojovými texty a dokumentací.