VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

PŘEŽITÍ SILNĚJŠÍHO V MINIMALIZACI NEDETERMINISTICKÝCH KONEČNÝCH AUTOMATŮ

SURVIVAL OF THE FITTEST IN NONEDTERMINISTIC FINITE AUTOMATA MINIMIZATION

PROJEKTOVÁ PRAXE
PROJECT PRACTICE

AUTOR PRÁCE **MICHAL ŠEDÝ**
AUTHOR

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

BRNO 2021

Abstrakt

Používání již existujících minimalizačních metod, kterými jsou slučování stavů, odstraňování přechodů a saturace, zanechává v minimalizovaných nedeterministických konečných automatech (NKA) potenciálně minimalizovatelné podgrafy obsahující duplicitní informace. Práce přináší nové techniky umožňující minimalizaci také těchto fragmentů, které jsou tvořeny skupinou stavů, kde je celý jazyk jednoho stavu po částech reprezentován také ostatními stavy. Protože mezi takovými stavy neexistuje relace jazykové inkluze, nejsou saturace a odstraňování hran použitelné. Nově vyvinutá metoda umí tyto skupiny stavů efektivně minimalizovat. Pro minimalizaci počtu stavů v dříve zmíněných fragmentech automatu, je využit SAT solver Z3. Síla vyvinutých technik je porovnána s výsledky minimalizačního nástroje RABIT používajícího slučování, saturace a odstraňování přechodů.

Abstract

The techniques, such as state merging, transition pruning, and saturation, are used for the minimization of nondeterministic finite automata (NFA). These techniques can leave potentially minimizable subgraphs containing duplicate information in the automaton. The work aims to investigate new methods for the minimization of these automaton fragments, that consist of a group of states, where a whole language of one state is piecewise covered by the language of the other states. Because no language inclusion is between states of this group, saturation or transition pruning can not be applied. Algorithms developed by this thesis can effectively resolve these groups of states. The SAT solver Z3 is used for the minimization of already mentioned fragments of an automaton. The strength of the newly created techniques is compared with the results of the minimization tool RABIT, which uses merging, saturation, and transition pruning.

Klíčová slova

konečné automaty, nedeterministické konečné automaty, NKA, minimalizace, SAT solver, Z3 solver, ekvivalence stavů, ekvivalence jazyků, slučování stavů, quotienting, simulace

Keywords

finite automata, nondeterministic finite automata, NFA, minimization, SAT solver, Z3 solver, state equivalency, language equivalency, state merging, quotienting, simulation

Citace

ŠEDÝ, Michal. *Přežití silnějšího v minimalizaci nedeterministických konečných automatů.* Brno, 2021. Projektová praxe. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Mgr. Lukáš Holík, Ph.D.

Obsah

Kapitola 1

Úvod

Nedeterministické konečné automaty (NKA) prezentoval Michael Rabin a Dana Scott v [11]. Ve srovnání s deterministickými konečnými automaty (DKA) se vyznačují schopností přechodu do více následujících stavů na základě stejného přijatého znaku. Díky této vlastnosti umožňují NKA reprezentovat jazyk za pomocí menšího množství stavů a přechodů, než jeho deterministická varianta. Nedeterministické konečné automaty se často využívají pro reprezentaci regulárních jazyků.

Regulární jazyky jsou hojně využívány v aplikacích pro validaci dat, definici slovníků (například škodlivých úseků kódů), internetových vyhledávačích, genetice (pro vyhledávání sekvencí nukleových kyselin v DNA) [15], rozpoznávání řetězců v síťovém provozu a jeho klasifikaci, a mnoha dalších.

Příkladem využití NKA reprezentujícího slovník regulárních výrazů je kontrola síťového provozu. V důsledku se stále zvyšujícího objemu dat přenášených po sítí a také rychlosti přenosu, je potřeba zvyšovat také rychlost jejich skenování. Standardní softwarová řešení vyhledávající podřetězce, kterými mohou být jak zajímavé statistické údaje, tak škodlivý kód, nejsou při vysokých rychlostech přenosu použitelné. Pro rychlosti nad 100Gbps je potřeba vytvořit hardwarovou implementaci vyhledávání řetězců [7, 14]. Pomocí technik [12] je možné reprezentovat NKA přímo na FPGA. Z důvodu ušetření místa a prostředků, je vhodné původní automat minimalizovat. Původní automat by dokonce nemusel být bez minimalizace vůbec realizovatelný (z důvodu své velikosti).

Současné techniky minimalizace NKA založené na: slučování stavů [5], které prezentovali Lucian Ille a Sheng Yu, nebo saturaci a odstraňování přechodů [2] od Lorenzo Clemente a Richard Mayr, jsou velice efektivní. Nicméně existují případy, kdy minimalizace není známými metodami možná.

Práce zkoumá nové techniky pro minimalizaci doposud neřešitelných podčástí automatů. Jedná se o případ výskytu skupiny stavů v NKA, kde je celý jazyk jednoho stavu po částech reprezentován také ostatními stavy. Protože každý ze stavů může kromě části tohoto jazyka obsahovat také svůj jedinečný jazyk, nemusí se mezi jazyky stavů nacházet relace inkluze. Protože minimalizační techniky jako saturace a odstraňování stavu jsou prováděny právě na základě jazykové inkluze, je minimalizace těchto stavů vynechána. Za tímto účelem je použit SAT solver Z3 [9], který je hlavním nástrojem určujícím optimální postup při slučování stavů.

Kapitola 2 je věnována základním pojmům souvisejících s NKA a jejich minimalizaci. Jsou také zavedeny nové pojmy pro usnadnění popisu zkoumaných metod.

Po uvedení do problematiky automatů jsou v kapitole 3 definován cíl a základní myšlenky práce, zaměřující se na definici nového minimalizačního postupu umožňujícího minimalizovat ty části automatů, kde dosavadní metody selhávají.

Kapitola 4 je věnována kódování problému skupiny stavů a také algoritmům pro jejich nalezení a minimalizaci.

Poslední kapitola 5 prezentuje výsledky experimentů prováděných s vyvinutými algoritmy. Pro porovnání jsou stejné automaty minimalizovány také existujícím nástrojem RABIT [1].

Kapitola 2

Prerekvizity

Tato kapitola uvádí základní pojmy týkající se nedeterministických konečných automatů (NKA), jejich definice a notací potřebnou pro snazší porozumění následujícím kapitolám. Formální definice NKA (převzata z [13, s. 53]), spolu se souvisejícími pojmy, jakými jsou konfigurace automatu, přechod, dosažitelný nebo ukončující stav a reverzní automat jsou zavedeny v sekci 2.1. Definice konfigurace a přechodu je převzata z [8]. Hlavni vlastnosti automatu je jazyk, který definuje všechny akceptovatelné řetězce. Tyto definice specifikuje sekce 2.2. Jazyk není pouze vlastnosti celých automatů. Definice jazyka pro jednotlivé stavy automatu přináší sekce 2.3, jedná se o jazyk dopředný, zpětný a také speciální jazyky definované pro tuto práci. Sekce 2.4. je věnována definici simulace. Jedná se o aproximační metodu hojně využívanou k určování jazykových relací, z důvodu své vyšší rychlosti. V poslední sekci 2.5 je popsána technika slučování stavů.

2.1 Formální definice nedeterministického konečného automatu

Nedeterministické konečné automaty jsou odvozené od deterministickým. Obsahují stavy, přechody mezi nimi a abecedu znaků. Existují dvě speciální skupiny stavů, počáteční a koncové. Počáteční stav vytváří vstupní místo automatu. Čtení znaků vyhodnocovaného řetězce začíná vždy na tomto stavu. Oproti počátečnímu stavu reprezentují koncové stavy místa, která jako jediná v NKA označí čtený řetězec jako přijatý. Stavy jsou propojeny přechody s ohodnocením. Na základě přečteného symbolu bude proveden odpovídající přechod do nového stavu. Oproti deterministickým automatům, kdy je takovýto přechod možný vždy maximálně jede, může být u nedeterministických automatů, jak již název napovídá, po přečtení znaku proveden přechod do více následujících stavů. Z jednoho stavu může vést více hran přechodu se stejným ohodnocením. V případě nedeterministického přechodu se vyhodnocování rozdělí a řetězec je prohlášen za přijatý, pokud alespoň v jednom z vyhodnocování skončil v koncovém stavu.

Pro formální definici musíme zavést dodatečnou notaci. Pro jakoukoliv množinu stavů Q značíme $P(Q)$ jako kolekci všech podmnožin množiny Q. Pro abecedu Σ píšeme Σ_ϵ na místo $\Sigma \cup \{\epsilon\}$. Nyní můžeme s touto notací popsat funkci přechodů v NKA jako $\delta : Q \times \Sigma_\epsilon \longrightarrow P(Q)$.

Definice 2.1 ***Nedeterministický konečný automat*** *je pětice* $M = (Q, \Sigma, \delta, I, F)$*, kde*

1. Q *je konečná množina stavů*
2. Σ *je abeceda*
3. $\delta : Q_\epsilon \longrightarrow P(Q)$ *je přechodová funkce*
4. $I \subseteq Q$ *je množina počátečních stavů*
5. $F \subseteq Q$ *je množina koncových stavů*

Přechodová funkce δ specifikuje množinu pravidel přechodů R. Pro pravidlo r: $q \in \delta(p, a)$ budeme využívat zápis $pa \to q$, kde $p, q, \in Q$, $a \in \Sigma$ a $r \in R$. Toto pravidlo značí, že po přečtení vstupního znaku a ve stavu p dojde k přechodu do stavu q.

Definice 2.2 *Nechť* $M = (Q, \Sigma, \delta, I, F)$ *je NKA.* ***Konfigurace*** *na automatu* M *je řetězec* $\chi \in Q\Sigma^*$.

Konfigurace automatu zobrazuje informaci o aktuálním stavu (ne všech stavech, ve kterých se nedeterministický automat nachází) a zbývajícím řetězci na vstupu. Pokud se automat M nachází například ve stavu q a na vstupu je řetězec ab, pak má konfigurace tvar qab.

Definice 2.3 *Nechť* $M = (Q, \Sigma, \delta, I, F)$ *je NKA a* paw, qw *dvě konfigurace automatu* M*, kde* $p, q \in \Sigma_\epsilon$, $w \in \Sigma^*$*. Pokud existuje pravidlo* r*:* $pa \to q \in R$*, pak* M *může provést* ***přechod*** *z* qaw *do* qw *za použití* r*. Značíme* $paw \vdash qw\ [r]$ *nebo zjednodušeně* $paw \vdash qw$*.*

Nechť χ je konfigurace. M provede nula přechodů z χ do χ. Značíme $\chi \vdash^0 \chi\ [\epsilon]$ nebo zjednodušeně $\chi \vdash^0$.

Nechť existuje sekvence konfigurací $\chi_0, \chi_1, \ldots, \chi_n$ a $\chi_{i-1} \vdash \chi_i\ [r_i]$, kde $r_i \in R$ pro $i = 1, \ldots, n$, tedy: $\chi_0 \vdash \chi_1\ [r_1] \vdash \chi_2\ [r_2] \ldots \vdash_n [r_n]$. Pak M provede n přechodů z χ_0 do χ_n. Zapisujeme $\chi_0 \vdash^n \chi_n\ [r_1 \ldots r_n]$, nebo zjednodušeně $\chi_0 \vdash^n \chi_n$.

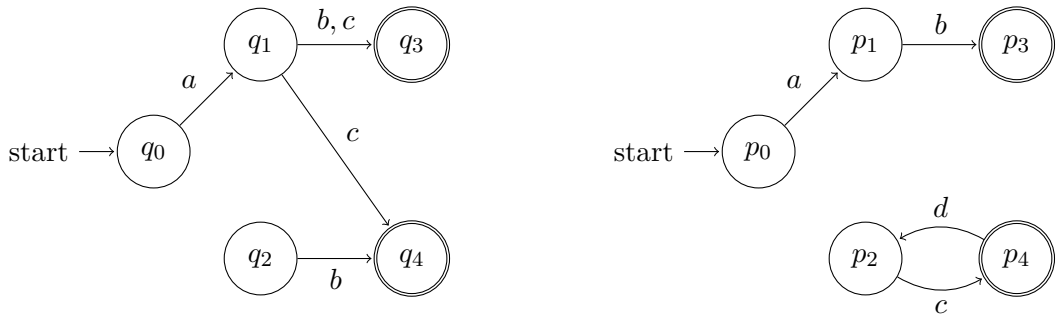
Pokud $\chi_0 \vdash^n \chi_n\ [\rho]$ pro $n \geq 1$, pak píšeme $\chi \vdash^+ \chi_n$ nebo zjednodušeně $\chi \vdash^+ \chi_n$.

Pokud $\chi_0 \vdash^n \chi_n\ [\rho]$ pro $n \geq 0$, pak píšeme $\chi \vdash^* \chi_n$ nebo zjednodušeně $\chi \vdash^* \chi_n$.

Již během vytváření automatu, ale spíše až během jeho úprav, mohou vzniknout stavy, jejichž existence nemá na výsledný jazyk automatu žádný vliv. Existují dva typy těchto stavů.

Prvním typ je způsoben neexistencí cesty z počátečního stavu. Tato nedostupnost může být způsobena absencí předchůdců stavů, nebo jeho izolací v nedostupném podgrafu automatu.

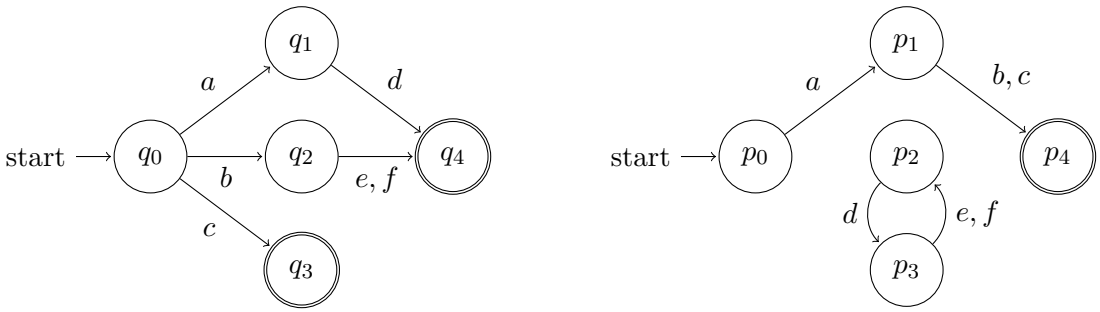
Definice 2.4 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$*. Pak stav* $q \in Q$ *je* ***dostupný****, pokud existuje* $w \in \Sigma^*$ *pro které platí* $q_0 w \vdash^* q$*, kde* $q_0 \in I$*. V opačném případě se jedná o nedostupný stav.*



Obrázek 2.1: Dva automaty s nedostupnými stavy. Pro levý automat se jedná o stav q_2 a pro pravý o stavy q_2 a q_4.

Druhým typem, je takzvaně neukončující stav. Jedná se o případ, kdy ze zkoumaného stavu nevede žádná cesta do koncového stavu. Toto odříznutí stavu může být způsobeno neexistenci následníka, nebo uzavřením stavu v neukončujícím podgrafu.

Definice 2.5 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$*. Stav* $q \in Q$ *je* ***ukončující****, pokud existuje* $w \in \Sigma^*$*, pro které platí* $qw \vdash^* f$*, kde* $f \in F$*. V opačném případě se jedná o neukončující stav.*



Obrázek 2.2: Dva NKA automaty s neukončujícími stavy. Pro levý automat to je stav q_3. V pravém automatu se jedná o stav q_2 a q_3.

Oba typy nepotřebných stavů mohou být odstraněny, protože se nenachází v žádné části automatu, která by definovala jeho jazyk. Eliminace nedostupných a neukončujících stavů je nejprimitivnější technikou minimalizace.

Pro usnadnění popisu algoritmů pracujících s předky stavů je vhodné definovat reverzní automat. Během procesu převodu jsou koncové a počáteční stavy prohozeny a směr všech přechodů obrácen.

2.2 Jazyky NKA

Jak již bylo dříve zmíněno, hlavní vlastnosti automatu je jazyk. Jazyk je množina řetězců definovaných topologií automatu. Může se jednat například o slovním úseků nebezpečného kódu. Následující definice přijímaný řetězce je převzata z [13, s. 54].

Definice 2.6 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$ *a řetězec* w *nad abecedou* Σ. *Pak říkáme, že* w *je* ***řetězcem přijímaný*** *automatem* M, *pokud jej můžeme zapsat ve tvaru* $w = y_1 y_2 \ldots y_n$, *kde* $y_i \in \Sigma_\epsilon$ *pro* $i = 1 \ldots n$ *a existuje taková sekvence stavů* $r_0, r_1, \ldots, r_n \in Q$, *pro kterou platí následující tři podmínky:*

1. $r_0 \in I$
2. $r_{i+1} = \delta(r_i, y_{i+1})$ *pro* $i = 0, \ldots, n\text{-}1$
3. $r_n \in F$

Podmínka 1. říká, že čtení musí začít v počátečním stavu. Podmínka 2. vyžaduje, aby pro každý stav r_i existoval přechod do stavu r_{i+1} po přečtení znaku y_{i+1}. Poslední podmínka 3. říká, že řetězec w je přijímaný automatem M, pokud poslední stav r_n je zároveň koncovým stavem automatu.

Definice 2.8 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$. ***Přijímaný jazyk*** *je definován jako* $L(M) = \{w \mid w \in \Sigma^*, q_w \vdash^* f, \text{ kde } q_0 \in I \text{ a } f \in F\}$.

Definice 2.9 *Nechť jsou definovány dva NKA* $M = (Q_M, \Sigma_M, \delta_M, I_M, F_M)$ a $N = (Q_N, \Sigma_N, \delta_N, I_N, F_N)$. *Automaty* M *a* N *jsou* ***ekvivalentní****, pokud platí rovnost* $L(M) = L(N)$.

Dva automaty M a N jsou na základě definice 2.9 ekvivalentní pokud pro každý řetězec $w_M \in L(M)$ existuje sekvence přechodů $w_M p_0 \vdash^* f_N$, kde $p_0 \in I_N$ a $f_N \in F_N$, a pro každý řetězec $w_N \in L(N)$ existuje sekvence přechodů $w_N q_0 \vdash^* f_M$ v M, kde $q_0 \in I_M$ a $f_M \in F_M$.

2.3 Jazyky stavů

Pro minimalizaci stavů NFA je potřebné znát vzájemné vztahy jejich jazyků. Nejvíce využívanou relací pro minimalizaci je ekvivalence. Může se jednat o ekvivalenci dopředných, zpětných, popřípadě obou těchto jazyků. Dalším důležitým vztahem je jazyková inkluze. Ta informuje o podobnosti dvou stavů, kde jeden stav je slabší a druhý silnější. Silnější stav pokrývá jazyk (dopředný, zpětný, atd.) slabšího stavu. Slabší stav bude eliminován a silnější přežije. Pro následující práci budou definovány speciální jazyky, kterými jsou: jazyk mezi stavy, ryzí jazyk, nebo jazyk definovaný na předem danou vzdálenost.

Základní skupinou jazyků stavů jsou jazyky dopředné a zpětné. Dopředný jazyk je definován jako množina řetězců nad abecedou automatu, pro které existuje sekvence přechodů (cesta) mezi počátečním stavem a stavem zkoumaným. Levý jazyk stavu $q \in Q$ budeme značit $\overleftarrow{L}(q)$ nebo zjednodušeně $\overleftarrow{q}$.

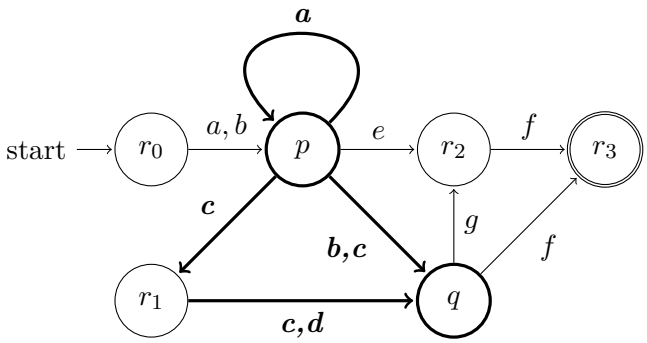
Definice 2.10 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$. ***Zpětný jazyk stavu*** $q \in Q$ *je definován jako* $\overleftarrow{L}(q) = \{w_l \mid w_l \in \Sigma^*, q_0 w_l \vdash^* q, \text{ kde } q_0 \in I\}$.

Dopředný jazyk je definován jako množina řetězců, pro které existuje cesta ze zkoumaného stavu do stavu koncového. Dopředný jazyk stavu $q \in Q$ budeme značit jako $\overrightarrow{L}(q)$ nebo zjednodušeně $\overrightarrow{q}$.

Definice 2.11 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$. ***Dopředný jazyk stavu*** $q \in Q$ *je definován jako* $\overrightarrow{L}(q) = \{w_r \mid w_r \in \Sigma^*, qw_l \vdash^* f, \text{ kde } f \in F\}$.

Do skupiny speciálních jazyků patří jazyk mezi stavy. Jak již bylo dříve řečeno definuje jej množina řetězců , pro které existuje sekvence přechodů mezi prvním a druhým stavem. Jazyk mezi stavy p a $q \in Q$ zapisujeme jako $L(p, q)$.

Definice 2.12 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$. ***Jazyk mezi stavy*** p *a* $q \in Q$ *je definován jako* $L(p, q) = \{w_b \mid w_b \in \Sigma^*, pw_l \vdash^* q\}$.



Obrázek 2.3: Jazyk mezi stavy p a q $L(p, q)$.

Na tvar řetězců jazyka mezi stavy nemá vliv uspořádání dvojice krajních stavů, ale topologie automatu. Reverzní jazyk mezi stavy q a p nelze získat pouhým prohozením operandů. Pro reverzní jazyk musí být nejdříve zreverzován celý automat.

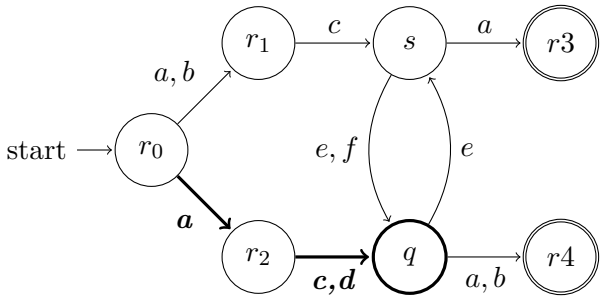
Dalším speciálním jazykem je ryzí jazyk. Jedná se o speciální jazyk, který je definován množinou řetězců nad abecedou automatu, pro které existuje skupina přechodů mezi krajními stavy (počátečním a stavem zkoumaným v případě zpětného jazyka, nebo zkoumaným stavem a koncovým stavem pro dopředný jazyk, nebo dvěma zkoumanými stavy) bez použití specifikovaného stavu. Tato podmínka neplatí pro krajní stavy (krajní stav může být zakázaným). Například ryzí zpětný jazyk stavu q, který nepoužívá stav r, je množina řetězců, pro které existuje cesta z počátečního stavu do stavu q, a to bez použití stavu r.

Definice 2.13 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$ *a* $q, s \in Q$. ***Ryzí zpětný jazyk*** *stavu* q *bez použití* s *značený* $\overleftarrow{L}(q, \overline{s})$ *je definován jako množina všech řetězců* w *ve tvaru* $w = y_1y_2 \ldots y_n$, *kde* $y_i \in \Sigma$ *pro* $i = 1 \ldots n$ *a existuje taková sekvence stavů,* $r_0r_1r_2 \ldots r_n \in Q \backslash \{s\}$, *pro kterou platí následující tři podmínky:*

1. $r_0 \in I$
2. $r_{i+1} = \delta(r_i, y_{i+1})$ *pro* $i = 0, \ldots, n\text{-}1$
3. $r_n = q$

Definice 2.14 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$ *a* $q, s \in Q$. ***Ryzí dopředný jazyk*** *stavu* q *bez použití* s *značený* $\overrightarrow{L}(q, \overline{s})$ *je definován jako množina všech řetězců* w *ve tvaru* $w = y_1 y_2 \ldots y_n$, *kde* $y_i \in \Sigma$ *pro* $i = 1 \ldots n$ *a existuje taková sekvence stavů,* $r_0 r_1 r_2 \ldots r_n \in Q \backslash \{s\}$, *pro kterou platí následující tři podmínky:*

1. $r_0 = q$
2. $r_{i+1} = \delta(r_i, y_{i+1})$ *pro* $i = 0, \ldots, n\text{-}1$
3. $r_n \in F$



Obrázek 2.4: Ryzí zpětný jazyk $\overleftarrow{L}(q, \overline{s})$ automatu $M_{2.5}$.

Definice 4.15 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$ *a* $p, q, s \in Q$. ***Ryzí jazyk mezi stavy*** p *a* q *bez použití* s *značený* $L(p, q, \overline{s})$ *je definován jako množina všech řetězců* w *ve tvaru* $w = y_1 y_2 \ldots y_n$, *kde* $y_i \in \Sigma$ *pro* $i = 1 \ldots n$ *a existuje taková sekvence stavů,* $r_0 r_1 r_2 \ldots r_n$, *kde* r_0, $r_n \in Q$ *a* $r_i \in Q \backslash \{s\}$, *pro* $i = 1 \ldots n-1$, *pro kterou platí následující tři podmínky:*

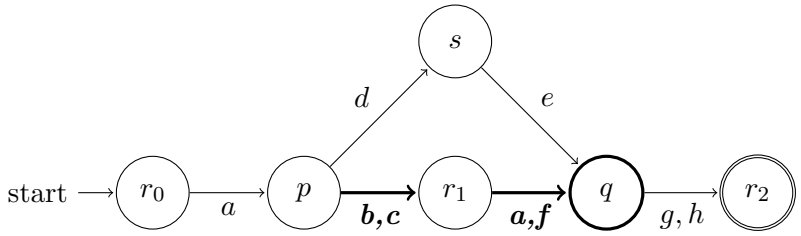
1. $r_0 = p$
2. $r_{i+1} = \delta(r_i, y_{i+1})$ *pro* $i = 0, \ldots, n-1$
3. $r_n = q$

Posledním speciálním jazykem je jazyk na fixní vzdálenost. Jedná se o standardní jazyk obohacený o skutečnost, že jeho řetězce jsou definovány pouze stavy v maximální vzdálenosti n od zkoumaného stavy. Jazyk s fixní vzdálenosti se bude hojně využívat pro upřesnění dopředného, nebo zpětného jazyka. Pro zpětný jazyk definuje řetězce o délce n, které jsou sufixy jednotlivých řetězců celého zpětného jazyka stavu. Obráceně pro dopředný jazyk jsou řetězce prefixy jednotlivých řetězců z dopředného jazyka stavu.

Definice 2.16 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$ a $\overleftarrow{L}(q)$ *zpětný jazyk stavu* $q \in Q$. ***Zpětný jazyk stavu na fixní vzdálenost*** n *je definován jako* $\overleftarrow{L_n}(q) = \{w' \mid \|w'\| = Max(n, \|w\| \in \overleftarrow{L}(q)), \exists\, x \in \Sigma^*$ pro které platí $w'x = w\}$.

Definice 2.17 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$ a $\overrightarrow{L}(q)$ *zpětný jazyk stavu* $q \in Q$. ***Dopředný jazyk stavu na fixní vzdálenost*** n *je definován jako* $\overrightarrow{L_n}(q) = \{w' \mid \|w'\| = Max(n, \|w\| \in \overrightarrow{L}(q)), \exists\, x \in \Sigma^*$ pro které platí $xw' = w\}$.

Všechny speciální vlastnosti jazyků mohou být kombinovány. Může být vytvořen například rytí zpětný jazyk s fixní vzdálenosti. Obrázek 2.5 ukazuje ryzí zpětný jazyk na fixní vzdálenost 2, $\overleftarrow{L_2}(q, \overline{s}) = \{ba, af, ca, cf\}$.



Obrázek 2.5: Ryzí zpětný jazyk na fixní vzdálenost 2, $\overleftarrow{L_2}(q, \overline{s})$, atuomatu $M_{2.6}$.

2.4 Simulace

Pro výpočet jazykových inkluzí, a v konečném důsledku jazykových ekvivalencí, existují dva přístupy. První metoda je založena na konstrukci podmnožina (*subset construction*). Jedná se o algoritmus využívající transformaci NKA na DKA, který je dále vstupem výpočtu ekvivalence pro deterministické automaty. Během převodu může dojít k explozi počtu stavů. Pokud je Q množina všech stavů původního nedeterministického konečného automatu, pak potenční množina $P(Q)$ má velikost $2^{\|Q\|}$, tedy výslední DKA může obsahovat až $2^{\|Q\|}$ stavů [10]. Druhým způsobem výpočtu jazykové inkluze jsou metody založené na simulaci. Simulace je aproximační metoda rozhodnutelná v polynomiálním čase [3] a je tedy často efektivnější než metody založené na konstrukci podmnožin. V mnoha případech se využívá pro výpočet jazykových inkluzí právě simulace. Bohužel simulace je nekompletní. Simulace implikuje jazykovou inkluzi, ale opačné tvrzení neplatí.

Definice 1.18 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$. ***Simulace*** *je kvaziuspořádání* $\preceq \subseteq Q \times Q$, *například* $p \preceq r$ *pouze pokud platí:*

1. $p \in F \implies r \in F$
2. pro každý přechod $pa \to p'$, existuje přechod $ra \to r'$ takový, že $p' \preceq r'$

První podmínka vyjadřuje, že pokud je stav q koncovým, pak musí být koncový i stav p. Druhá podmínka říká, že pro každý přechod z p do p' pomocí znaku a musí existovat přechod z q do q' pomocí stejného znaku. Nad stavy p' a q' jsou pak dále rekurzivně znovu aplikována pravidla 1 a 2. Pokud jsou veškeré podmínky splněny, pak je stav p simulován stavem q.

2.5 Slučování stavů

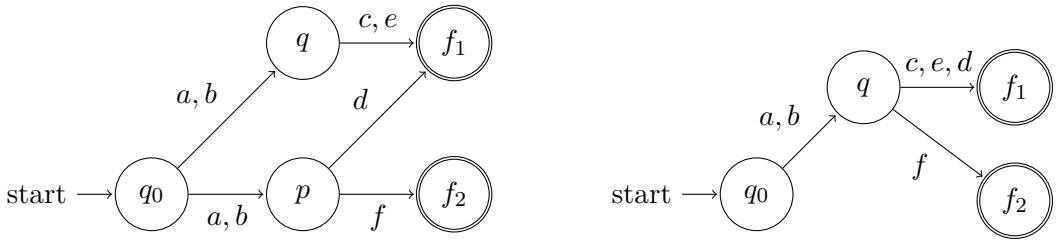
Nejznámější a historicky nejstarší technikou minimalizující nedeterministické konečné automaty je slučování stavů, které jako první publikovali Llie a Yu [5]. Metoda pracuje na principu slučování ekvivalentních stavů. Může se jednat o zpětnou, dopřednou, nebo oboustrannou jazykovou ekvivalenci. Sloučení může být také provedeno pouze na základě oboustranné jazykové inkluze. Podmínky umožňující slučování uvádí teorém 2.20 [4]. O ekvivalenci může být rozhodnuto na základě relace simulace. Pokud jsou dva stavy ekvivalentní, budou sloučeny, přičemž bude sloučen jeden do druhého (například p do q, kde q zůstává a p zmizí). Při sloučení jsou všechny hrany přechodů zdrojového stavu (bude zahozen) přesměrovány na cílová stav (stav zůstává po minimalizaci).

Definice 2.19 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$ *a* $p, q \in Q$. ***Automat se sloučenými stavy*** p *do* q *je definován jako* $M' = (Q', \Sigma, \delta', I', F')$, kde

1. $Q' = Q \backslash \{p\}$ *je konečná množina stavů*
2. Σ *je abeceda*
3. $\delta'(s,a) = \begin{cases} \delta(s,a) \setminus \{p\} \cup \{q\}, & \text{pro } p \in \delta(s,a), \\ \delta(p,a), & \text{pro } s = q, \\ \delta(s,a), & \text{jinak.} \end{cases}$
4. $'I = I \setminus \{p\}$ *je konečná množina počátečních stavů*
5. $F' = F \setminus \{p\}$ *je konečná množina koncových stavů*

Teorém 2.20 *Dva stavy* p *a* q *z automatu* M *mohou být sloučeny, pokud platí alespoň jedna z následujících podmínek:*

1. $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q) \wedge \overleftarrow{L}(q) \subseteq \overleftarrow{L}(p)$,
2. $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overrightarrow{L}(q) \subseteq \overrightarrow{L}(p)$,
3. $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q) \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$, *v tomto případě je* p *slabý stav a je sloučen do* q.



Obrázek 2.6: Automat $M_{2.7}$ (nalevo) a jeho minimalizovaná verze se stavem p sloučeným do q.

Kapitola 3

Cíle a hlavní myšlenky

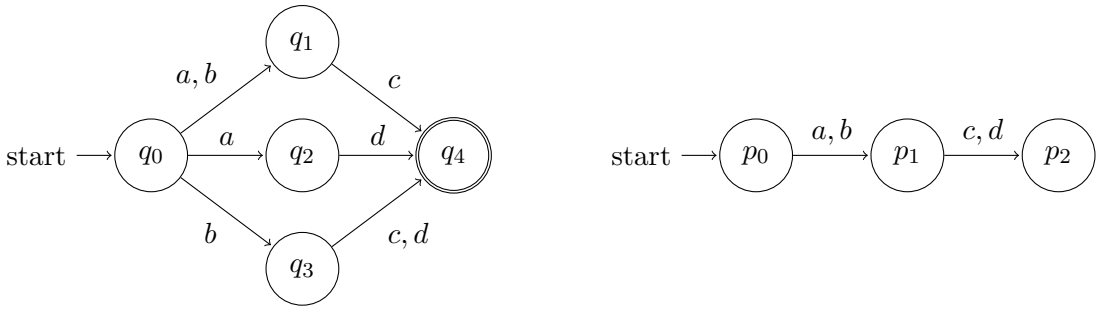
Cílem práce je vyvinout nové postupy minimalizace automatů na základě podobnosti jazyka množiny stavů (nemusí se nutně jednat o inkluzi). Definice těchto případů a jejich využití pro minimalizaci jsou uvedeny v úvodní sekci. Tato kapitola také popisuje již existující řešení, která jsou schopna tyto případy částečně řešit [2]. Jedná se o odstraňování přechodů, který umožňuje na základě jazykové inkluze mezi stavy odstraňovat jednotlivé přechody automatu, a tím vytváří nedostupné, nebo neukončující stavy. Další metodou je saturace, která obohacuje automat o duplicitní přechody a tím vytváří více potenciálně odstranitelných přechodů.

I přes svou minimalizační sílu nejsou stávající techniky všemocné. Existují speciální případy automatů, nebo jen jejich částí, které jsou teoreticky minimalizovatelné, ale současné postupy založené na jazykových relacích je nedovedou řešit.

3.1 Cíle práce

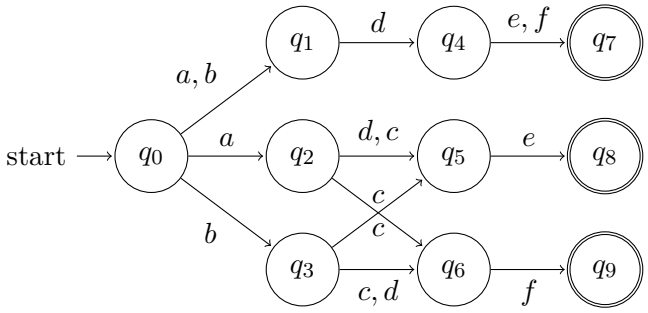
Cílem práce je prozkoumat nové metody, které by byly schopné řešit minimalizaci automatu s využitím podobnosti jazyků množiny stavů.

Hlavní myšlenkou je nalezení duplicitního kódování jazyka jdoucího přes skupinu příbuzných stavů.



Obrázek 3.1: Automat $M_{3.1}$ (nalevo) a jeho minimalizovaná verze (napravo).

Základní případ množiny stavů S jsou stavy $\{q_1, q_2, q_3\}$ z automatu $M_{3.1}$. Tyto stavy se nacházejí v podgrafu automatu s jedním zdrojem a cílem. Jedná se o podgraf 1:1. Dalším případem jsou části automatu se vztahem 1:N, popřípadě N:1. V případě 1:N, který je vyobrazen automatem $M_{3.2}$, existuje pouze jeden společný zdroj stavů z množiny S a vice cílů více cílů. Při vztahu N:1 je situace opačná.



Obrázek 3.2: Automat $M_{3.2}$ s relací 1:N.

3.2 Existující řešení

Mezi nejznámější a doposud nejefektivnější techniky minimalizace automatů doplňující slučování stavů patří odstraňování přechodů (transition pruning) a saturace. Obě tyto techniky vytváří nedostupné nebo neukončující stavy a tím umožňují jejich budoucí odstranění.

Následující text obsahuje definice případů, kdy je tyto metody možné aplikovat. Definice a teorémy publikované v [2] jsou pro snazší porozumění zjednodušeny. To však nic neubírá na jejich správnosti.

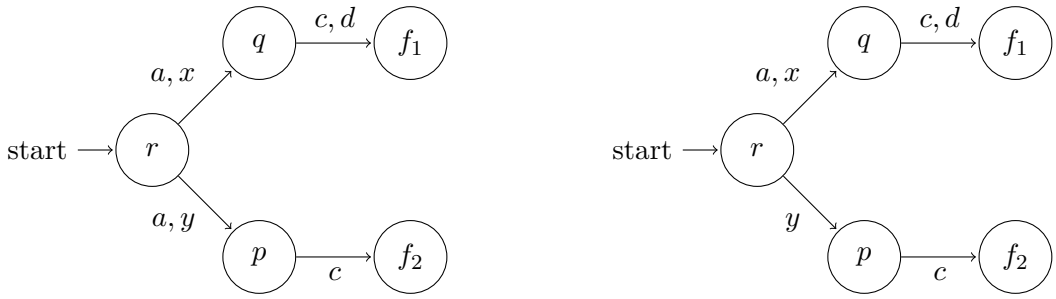
Odstraňování přechodů
Základní myšlenkou odstraňování přechodů je existence lepšího přechodu (jazykově silnějšího), který dokáže převzít jeho funkci. Protože jsou přechody pouze odstraňovány, platí pro jazyk výsledného automatu $L(A) \subseteq L(B)$, kde B je automat A po odstranění hran.

Definice 3.1 *Nechť je dán NKA $M = (Q, \Sigma, \delta, I, F)$.* ***Automat s odstraněným přechodem*** *$pa{\to}r$, kde $p, r \in Q$ a $a \in \Sigma$, je $M' = (Q, \Sigma, \delta', I, F)$ zapsáno $Prune(M, pa \to r)$ kde*

1. *Q je konečná množina stavů*
2. *Σ je abeceda*
3. *δ': $\delta \backslash (p, a, r)$ je funkce přechodů*
4. *$I \subseteq Q$ je konečná množina počátečních stavů*
5. *$F \subseteq Q$ je konečná množina koncových stavů*

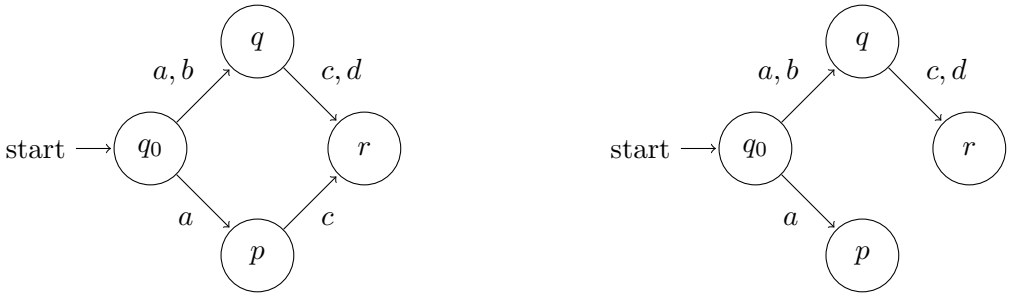
Následující teorémy ukazují případy, kdy je možné použít odstraňování přechodů. Důkazy těchto teorémů, provedeny na Büchiho automatech, jsou uvedeny v [2, s. 16–20].

Teorém 3.2 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$, $p, q, r \in Q$ *a* $a \in \Sigma$. *Přechod* $ra \to p$ *může být odstraněn, pokud existuje přechod* $ra \to q$ *a platí* $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$.



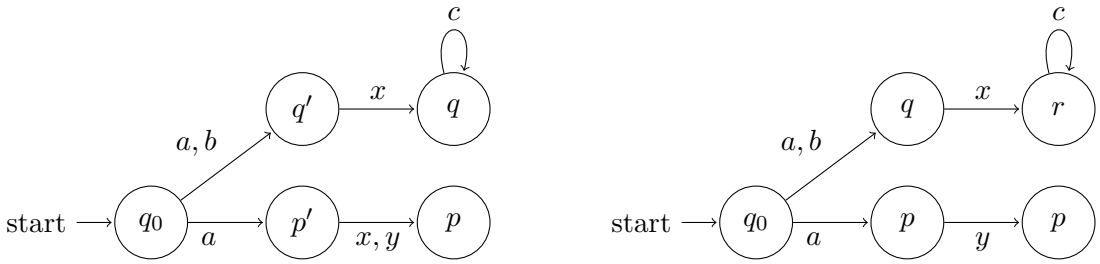
Obrázek 3.3: Automat $M_{3.3}$ (nalevo) a automat $Prune(M_{3.3}, ra \to p)$ (na pravo). Přechod byl odstraněn na základě teorému 3.2

Teorém 3.3 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$, $p, q, r \in Q$ *a* $a \in \Sigma$. *Přechod* $pa \to r$ *může být odstraněn, pokud existuje přechod* $qa \to r$ *a platí* $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$.



Obrázek 3.4: Automat $M_{3.4}$ (nalevo) a automat $Prune(M_{3.4}, pa \to r)$ (napravo). Přechod byl odstraněn na základě teorému 3.3.

Teorém 3.4 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$, $p, q, p', q' \in Q$ *a* $a \in \Sigma$. *Přechod* $q'a \to p'$ *může být odstraněn, pokud existuje přechod* $pa \to q$ *a platí* $\overleftarrow{L}(p') \subseteq \overleftarrow{L}(p) \wedge \overrightarrow{L}(q') \subseteq \overrightarrow{L}(q)$.



Obrázek 3.5: Automat $M_{3.5}$ (nalevo) a automat $Prune(M_{3.5}, qa \to p')$ (napravo). Odstranění přechodu bylo provedeno na základě teorému 3.4.

Saturace
Tato technika umožňuje přidávat nové hrany přechodů do automatu, a to bez změny jeho jazyka. Během procesu minimalizace jsou postupně aplikovány metody slučování stavů a odstraňování hran. V určitém okamžiku již nelze dále automat minimalizovat. Díky saturaci je v automatu možné zduplikovat hrany a tím umožnit další minimalizaci. Saturace je provedena pouze tehdy, pokud již daný (lepší) přechod existuje. Jedná se o analogii s odstraňováním hran.

Definice 3.5 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$ *a* $p, q \in Q$. *Pokud platí* $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$, *pak* ***dopředná saturace*** *stavu* p *stavem* q, *značeno* $\overrightarrow{Sat}(M, q, p)$, *změní funkci přechodu* δ *následovně:*

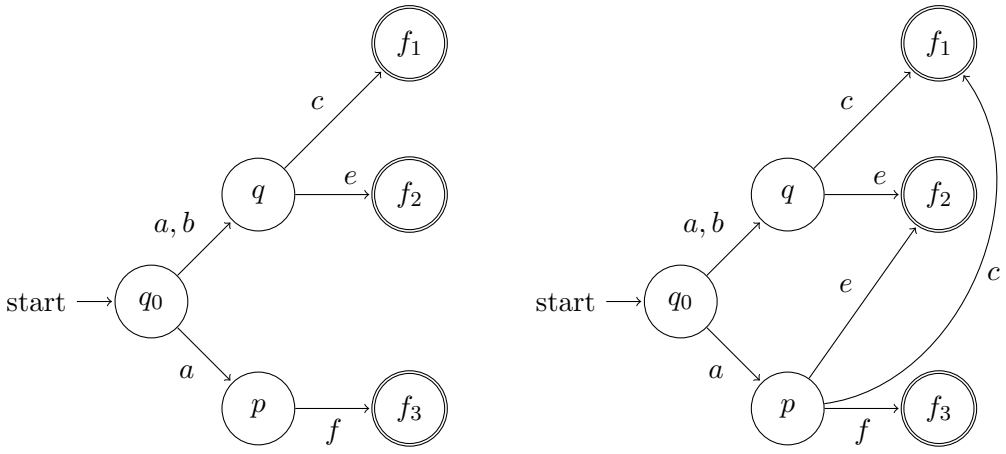
$$\delta'(s,a) = \begin{cases} \delta(q,a), & \text{pro } s = p, \\ \delta(s,a), & \text{jinak.} \end{cases}$$

Dopředná saturace je provedena pouze při zpětné jazykové inkluzi. Saturovaný stav je obohacen o všechny dopředné hrany vedoucí ze silnějšího stavu do jeho následníků. Příklad dopředné saturace je na obrázku 3.6.

Definice 3.6 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, I, F)$ *a* $p, q \in Q$. *Pokud platí* $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$, *pak* ***zpětná saturace*** *stavu* p *stavem* q, *značeno* $\overleftarrow{Sat}(M, q, p)$, *změní funkci přechodu* δ *následovně:*

$$\delta'(s,a) = \begin{cases} \delta(s,a) \setminus \{p\} \cup \{q\}, & \text{pro } p \in \delta(s,a), \\ \delta(s,a), & \text{jinak.} \end{cases}$$

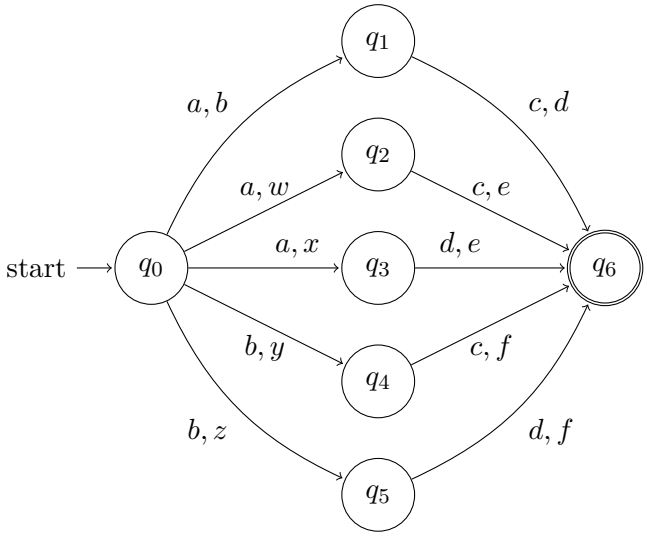
Zpětná saturace je provedena pouze při dopředné jazykové inkluzi. Saturovaný stav je obohacen o všechny zpětné hrany vedoucí do silnějšího stavu z jeho předchůdců.



Obrázek 3.6: Automat $M_{3.6}$ (nalevo) a automat $\overrightarrow{Sat}(M_{3.6}, q, p)$ (napravo).

3.3 Limity existujících řešení

Přestože jsou dosavadní metody velice robustní, stále existují případy, které nedovedou řešit (minimalizovat). Právě tyto doposud neminimalizovatelné případy jsou hlavním bodem této práce. V následujících odstavcích je demonstrován příklad minimalizace, který je pro současné metody neřešitelný.



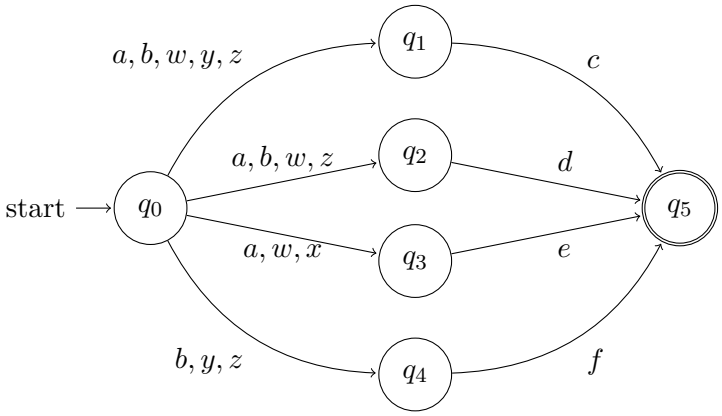
Obrázek 3.7: Automat $M_{3.7}$ není doposud známými metodami (slučování, saturace, odstraňování přechodů) řešitelný.

Slučování stavů není možné aplikovat na automat zobrazeny na obrázku 3.9. Slučování vyžaduje existenci dvou ekvivalentních stavů, nebo stavů v oboustranné jazykové inkluzi, které by mohly být sloučeny. V automatu neexistuje dvojice, která by podmínkám vyhovovala. Zároveň není možné provést ani saturaci automatu, protože saturace vyžaduje také relaci jazykové ekvivalence. Žádná taková dvojice se zde nenachází. Protože odstraňování stavů vyžaduje v jedné ze svých podmínek existenci jazykové inkluze, není možné aplikovat pro minimalizaci ani odstraňování.

Pozornému čtenáři je již určitě zřejmé, že jazyk jdoucí přes stav q_1 je celý pokryt jazykem množiny stavů $S = \{q_2, q_3, q_4, q_5\}$. A tudíž není pro jazyk automatu potřebný. Stav q_1 může být odstraněn. Vidíme, že minimalizace je velice triviální, ale známé metody ji nezvládají.

3.4 Hlavní myšlenky

V úvodu této sekce je představena naivní myšlenka minimalizace složitých podgrafů, která v sobě nese logickou úvahu pro budoucí postupy. Hlavní technika minimalizace vychází z vlastností metody slučování stavů. Na jejím základě je stanoven minimalizační teorém dovolující, popřípadě zakazující vzájemné slučování již jednou sloučených stavů.



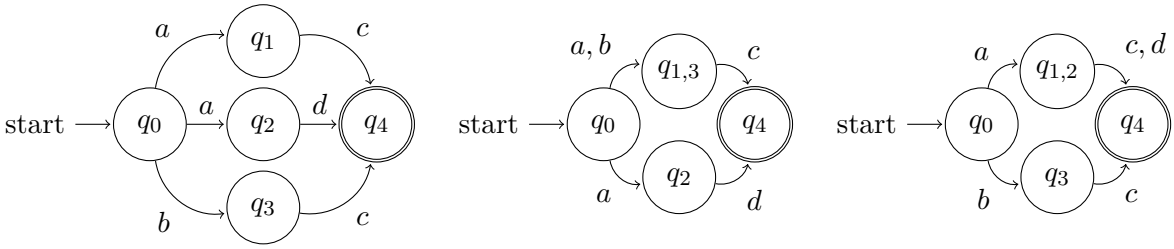
Obrázek 3.8: Naivní postup minimalizace automatu z obrázku 3.7.

Protože původní dopředné hrany směřující ze stavů množiny S automatu $M_{3.7}$ byly ohodnoceny 4 různými znaky, mohl být jejich počet snížen na tuto hodnotu. Hrany vycházející z počátečního stavu do stavu množiny S musely být přepočítány, aby byl zachován jazyk automatu. Počet stavů se snížil o jeden a počet hran klesl také o jednu. Z předchozí sekce ale víme, že pro automat existuje efektivnější rozložení s 16 hranami.

Teorém 3.7 *Množina stavů S může být minimalizována na $MIN(\|\overleftarrow{L}_1(S)\|, \|\overrightarrow{L}_1(S)\|)$ stavů pro podgraf 1:1, $\|\overleftarrow{L}_1(S)\|$ pro podgraf 1:N, nebo $\|\overrightarrow{L}_1(S)\|$ stavů pro podgraf N:1.*

Teorém 3.4 říká, že každá množina stavů S automatu M může být minimalizována na n stavů, kde n je minimum z počtu unikátních ohodnocení hran vstupujících a vystupujících z množiny S v závislosti na typu podgrafu

Slučování stavů nespecifikuje nejoptimálnější pořadí prováděných slučování. Různá pořadí mohou vést k různě efektivním řešením. Ille and Yo ukázali příklad [6] různých výsledku slučování (obrázek 3.9).



Obrázek 3.9: Původní NKA (nalevo) a jemu odpovídající výsledky slučování různých stavů.

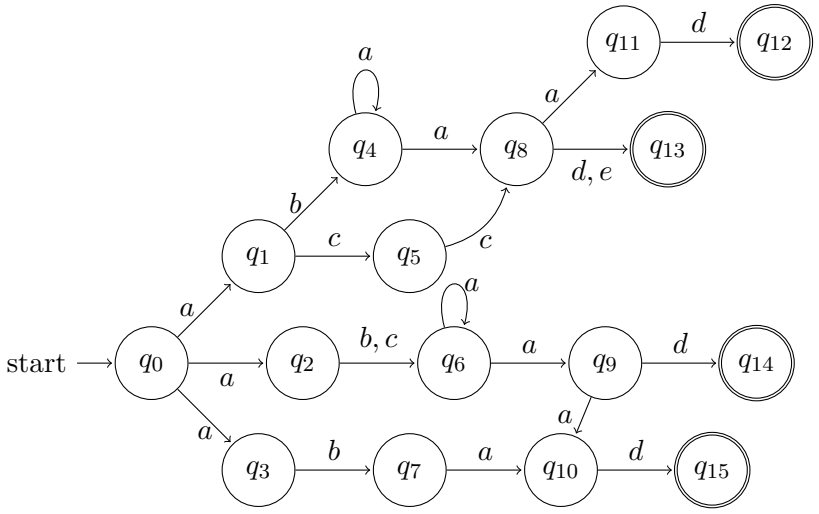
Během slučování stavů na základě zpětné ekvivalence jazyků, je levý jazyk nově vzniklého stavu ekvivalentní s původními stavy. Tato ekvivalence umožňuje použití sloučeného stavu v dalších minimalizaci založených na starých výpočtech levé jazykové inkluze. Při slučování stavů se může (a při pouhé levá jazykové ekvivalenci je to pravidlem) změnit pravý jazyk výsledného stavu (do kterého bylo slučováno). Slučování na základě starých výpočtů pravé jazykové inkluze již není bezpečné, a tudíž možné.

Teorem 3.8 *Pokud dva stavy p a q jsou slučovány, kde p je slučováno do q, na základě ekvivalence zpětných jazyků, pak nový zpětný jazyk stavu q′ je ekvivalentní s předchozími stavy p a q. Dopředný jazyk může být změněn, bez dopadu na jazyk automatu. Nově vytvořený stav q′ může být použit v minimalizacích založených na starých výpočtech zpětné ekvivalence všude tam, kde by byl použit stav p nebo q.*

Teorem 3.9 *Pokud dva stavy p a q jsou slučovány, kde p je slučováno do q, na základě ekvivalence dopředných jazyků, pak nový dopředný jazyk stavu q′ je ekvivalentní s předchozími stavy p a q. Zpětný jazyk může být změněn, bez dopadu na jazyk automatu. Nově vytvořený stav q′ může být použit v minimalizacích založených na starých výpočtech dopředné ekvivalence všude tam, kde by byl použit stav p nebo q.*

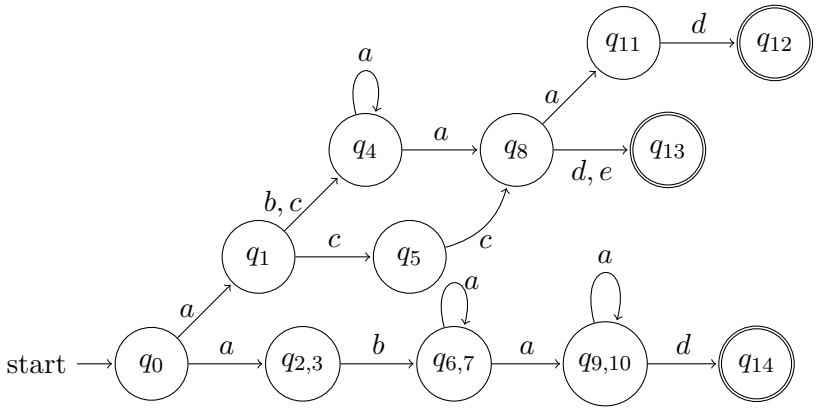
Teorem 3.10 *Pokud dva stavy p a q jsou slučovány, kde p je slučováno do q, na základě oboustranné ekvivalence jazyků, pak nový zpětný i dopředný jazyk stavu q′ je ekvivalentní s předchozími stavy p a q. Nově vytvořený stav q′ může být použit v minimalizacích založených na starých výpočtech zpětné nebo dopředné ekvivalence všude tam, kde by byl použit stav p nebo q.*

Nechť je dán NKA $M_{3.10} = (Q, \Sigma, \delta, I, F)$. Pokud z výsledků výpočtů jazykových inkluzí plyne, že $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$, pak je možné stav p sloučit do stavu q a vznikne automat se sloučenými stavy. Výsledný stav vzniklý sloučením již ale není možné dále použít k příštím minimalizacím, protože při slučování a převádění přechodů mohlo dojít ke změně levého, nebo pravého jazyka silnějšího stavu. Celkový jazyk stavů zůstává ale beze změny, viz. automat $M_{3.10}$ a $M'_{3.10}$.



Obrázek 3.10: Automat $M_{3.10}$, sloučí se stavy q_9 a q_{10}.

Z automatu $M_{3.10}$ lze vidět, že je možno libovolně sloučit buď q_8 s q_9, nebo q_9 s q_{10}. Co již ale není na první pohled zřejmé, je nemožnost sloužit q_9 s q_{10} a poté q_8 s q_9, viz. automat $M'_{3.10}$.



Obrázek 3.11: Automat $M'_{3.10}$ po drobných úpravách a sloučení stavů q_9 a q_{10} původního automatu $M_{3.10}$.

Z automatu $M'_{3.10}$ je vidět, že sloučení stavů q_9 a q_{10} neovlivnilo výsledný jazyk automatu a dokonce ani celkový jazyk původního silného stavu q_9, tedy celé větve. Změnil se ale levý jazyk stavu q_9. Výsledky jazykových simulací, na jejichž základě je řízena minimalizace, již nejsou platné pro nově vzniklý sloučený stav $q_{9,10}$, a proto jej musíme z další minimalizace s použitím starých výsledků simulace vyloučit. Je zřejmé, že pokus o sloučení stavů q_8 a q_9 by vedl v současné podobě automatu $M'_{3.10}$ ke změně jazyka automatu.

Teorém 3.11 *Pokud dva stavy p a q jsou slučovány, kde p je slučováno do q, na základě oboustranné jazykové inkluze, pak nový zpětný i dopředný jazyk stavu q' může být změněn. Nově vytvořený stav q' nesmí již být dále použit v minimalizacích založených na starých výpočtech zpětné nebo dopředné ekvivalence (popřípadě inkluze).*

Během slučování může být jeden stav v relaci ekvivalence zpětných jazyků s jednou skupinou stavů a v relaci ekvivalence dopředných jazyků s jinou skupinou stavů. Tato vlastnost může dále platit i pro ostatní stavy. Hlavní otázkou je: "Jak tyto stavy vhodně sloučit, aby byl výsledek nejefektivnější?". Na základě teorémů 3.8–3.11 víme, že stavy sloučené podle zpětné jazykové ekvivalence nemohou být dále slučovány se stavy podle staré pravé jazykové ekvivalence. Jak sloučit stavy, které patří do obou skupin, aby budoucí slučování bylo nejoptimálnější? Technologií, využitou pro tento optimalizační problém je Max SAT solver Z3 [9], který určí nejvhodnější postup slučování. Pro zvýšení množství ekvivalentních stavů je vhodné vybrané části automatu upravit tak (přidáním stavů a přechodů) aby hrany obsahovaly právě jedno ohodnocení.

Kapitola 4

Maximalizace minimalizace

Kapitola popisuje jednotlivé postupy pro zefektivnění minimalizace automatů. Jak již bylo řečeno, neexistuje způsob jak určit takové pořadí slučovaných stavů, aby byl výsledný automat nejminimálnější. Pro předpověď slučování je využit Z3 solver. Proces je uveden v sekci 4.1. Vytvořenou techniku lze také využít pro minimalizaci doposud neřešitelných podgrafů, jejichž vyhledávání je popsáno v sekci 4.2. Nalezené skupiny stavů (podgrafy) představují svou velikostí lokální minimum v minimalizaci. Abychom se dostali z tohoto minima je potřeba skupinu stavů roznásobit. Algoritmus roznásobení je uveden v sekci 4.3. Konec této kapitoly se věnuje algoritmům využívajících solver pro minimalizaci nedeterministických konečných automatu.

4.1 Využití Z3 při sloučení stavů

Pokud jeden stav tvoří zpětně ekvivalentní dvojice s jednou skupinou stavů a zároveň dopředně ekvivalentní dvojice s jinou skupinou, pak existují dvě možnosti jeho budoucího slučování. Stavy, které jsou spolu sloučeny na základě zpětné ekvivalence mohou být dále (na základě starých výpočtů ekvivalence) slučovány pouze se zpětně ekvivalentními stavy. Obrácené tvrzení platí pro stavy sloučené na základě dopředné ekvivalence. Pro výběr nejvhodnějších slučovaných párů, které zajistí maximalizaci počtu provedených sloučení je použit Max SAT solver.

Kódování SAT solveru
Pro Z3 solver je zvoleno následující kódování. qBr značí sloučení stavu q a stavu r podle zpětné jazykové ekvivalence. qFs značí sloučení stavů q a s podle dopředné jazykové ekvivalence.

Protože použití stavu při sloučení podle zpětné ekvivalence vylučuje použití v minimalizacích podle dopředné ekvivalence a na opak, jsou pro každý výskyt stavu v obou ekvivalencích zavedena pravidla: $qBr \implies \neg qFs$ (při sloučení stavu q a r na základě zpětné ekvivalence již nemůže být provedeno sloučení na stavů q a s na základě dopředné ekvivalence). Výjimkou v těchto pravidlech jsou identické dvojice stavů, které stojí před a za implikací. Sloučení qBr a následné rFq (qFr) je povoleno.

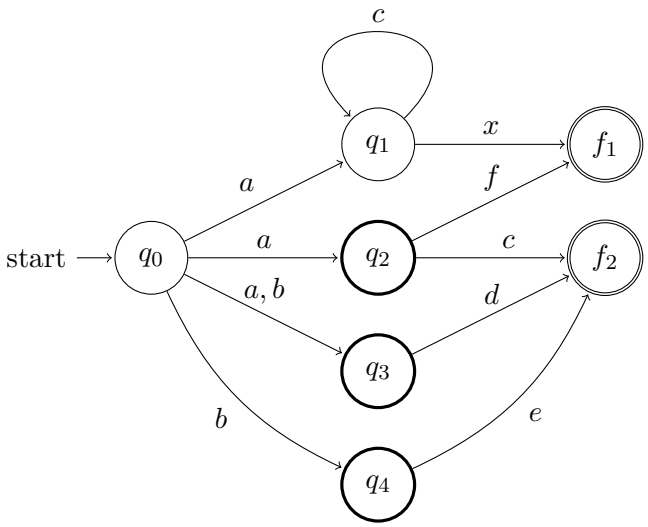
Nechť B_{eq} je množina dvojic stavů (p, q), kde $\overleftarrow{p} \subseteq \overleftarrow{q} \wedge \overleftarrow{p} \supseteq \overleftarrow{q}$ a F_{eq} je množina dvojic stavů (r, s), kde $\overrightarrow{r} \subseteq \overrightarrow{s} \wedge \overrightarrow{r} \supseteq \overrightarrow{s}$. V množinách reprezentujeme dvojici stavů vždy pouze jednou. Pro solver jsou použity proměnné $B_{vars} = \{pBq \mid (p, q) \in B_{eq}\}$ a $F_{vars} = \{rFs \mid (r, s) \in F_{eq}\}$. Dále jsou definovány podmínky $pBq \implies \neg p'Fq'$, kde $\exists\ (p', q') \in F_{eq}$, pro které $(p', q') = (r, x) \vee (x, r)$, kde $r \in \{p, q\}$ a $x \in Q \backslash \{p, q\}$. Na základě těchto omezujících pravidel, ohodnotí Max SAT solver proměnné tak, aby byl proveditelný maximální počet sloučení.

Příklad kódování
Nechť jsou dány množiny ekvivalentních dvojic stavů: $B_{eq} = \{(q_1, q_2)\}$, $F_{eq} = \{(q_2, q_3), (q_2, q_4), (q_3, q_4)\}$. Vznikají proměnné: q_1Bq_2, q_2Fq_3, q_2Fq_4, q_3Fq_4 a pravidla: $q_1Bq_2 \implies \neg q_2Fq_3$, $q_1Bq_2 \implies \neg q_2Fq_4$.

4.2 Stavy vhodné pro roznásobení

Techniku optimalizace slučování stavů lze použít na speciální množinu stavů v podgrafu 1:1, 1:N, nebo N:1. Tedy s jedním společným zdrojem a cílem, případně s právě jedním společným zdrojem a více cíli (nebo opačně).

Definice 4.1 *Nechť je dán NKA $M = (Q, \Sigma, \delta, I, F)$. Množina stavů S je **vhodná pro roznásobení**, pokud $\forall\ s \in S$ platí $pred(s) = x \vee suc(s) = y$, kde $x \neq s \wedge y \neq s$ a $x, y \in Q$.*



Obrázek 4.1: Automat $M_{4.1}$ se skupinou stavů $\{q_2, q_3\ q_4\}$ vhodnou pro roznásobení.

V automatu $M_{4.1}$ je skupina stavů $\{q_2, q_3\ q_4\}$ ohodnocena jako vhodná pro roznásobení. Do této skupiny není zařazen stav q_1, protože smyčka nad tímto stavem způsobuje, že je svým předchůdcem a také následníkem, což je v rozporu s definicí.

Aby nebyly roznásobovány již minimální skupiny stavů, lze na základě teorému 3.7 vybrat pouze ty skupiny, jejich roznásobení může přinést optimálnější řešení.

Algorithm 1 Algoritmus pro nalezení množin stavů vhodných k roznásobení

Vstup: NKA M = $(Q, \Sigma, \delta, I, F)$
Výstup: množina množin stavů vhodných k roznásobení

```
1:  S = set()                                   ▷ inicializace množiny množin stavů
2:  for centr ∈ Q do                ▷ vyhledávání společného stavu (předka, následníka)
3:      tmp = set()
4:      if ‖succ(center)‖ > 1 then                ▷ vyhledávání podgrafů 1:N nebo 1:1
5:          for s ∈ succ(center) do
6:              if s ≠ center ∧ pred(s) == 1 then
7:                  tmp = tmp ∪ {s}
8:              end if
9:          end for
10:     end if
11:     if ‖pred(center)‖ > 1 then                ▷ vyhledávání podgrafů N:1 nebo 1:1
12:         for s ∈ pred(center) do
13:             if s ≠ center ∧ succ(s) == 1 then
14:                 tmp = tmp ∪ {s}
15:             end if
16:         end for
17:     end if
18: end for
19: return S
```

Skupiny stavů vhodných k roznásobení jsou vyhledávány podle společných předchůdců, případně následníků. Pokud z některého stavu vedou hrany přechodu do více následníku, jsou tito následníci potenciálními hledanými stavy. Po splnění podmínky, že jejich jediným předchůdcem je právě jeden stav jsou prohlášeny za vhodné k roznásobení.

4.3 Roznásobení skupiny stavů

Pro efektivnější fungování metody slučování stavů založené na výsledcích Z3 solveru je potřebné roznásobit stavy minimalizované množiny S. V některých případech se může množina S nacházet v lokálním dále neminimalizovatelném minimu, a pro nalezení přesnějšího minima je nutno z něj vystoupat výše (zvýšit množství stavů).

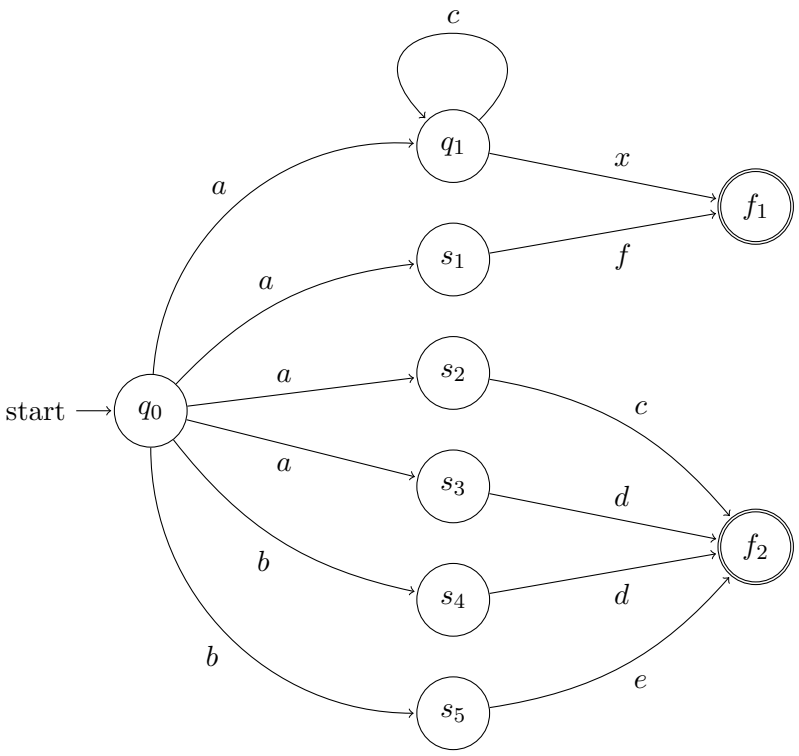
Po roznásobení množiny stavů budou tyto stavy nahrazeny z pravidla vyšším počtem stavů, které definují stejný jazyk. Nově vzniklé stavy obsahují právě jednu vstupní a maximálně jednu výstupní hranu.

Pro každý stav s z roznásobované množiny jsou vytvořeny nové stavy. Pokud je s počátečním čí koncovým stavem, budou mít tuto vlastnost také všichni jeho náhradníci (nově vzniklé stavy). Na nové stavy jsou navázány přechody tak, aby pokryly celý jazyk stavu s. Stav s je na konec z automatu odstraněn.

Algorithm 2 Algoritmus pro roznásobení množiny stavů S

Vstup: NKA M = $(Q, \Sigma, \delta, I, F)$ a množina stavů S

for $s \in S$ **do**
  $newS = set()$ ▷ inicializace množiny nových stavů
  $newS = createStates(count = \|\overleftarrow{L_1}(s) \cdot \overrightarrow{L_1}(s)\|)$ ▷ nové stavy nahrazující s
  if $s \in F$ **then**
    $F = F \cup newS$
  end if
  if $s \in I$ **then**
    $I = I \cup newS$
  end if
  $Q = Q \cup newS$
  for $Bwd \in \overleftarrow{L_1}(s) \cup \{\epsilon\}$ **do** ▷ ϵ provede cyklus i při prázdném jazyku
    for $Fwd \in \overrightarrow{L_1}(s) \cup \{\epsilon\}$ **do**
      $state = newS.pop()$
      for $pred \in pred(s)$ **do**
        $\delta(pred, Bwd) = \delta(pred, Bwd) \cup \{state\}$
      end for
      if $Fwd == \epsilon$ **then** ▷ zpětný jazyk je prázdný
        $\delta(state, Fwd) = succ(s)$
      end if
    end for
  end for
  $Q = Q \backslash \{s\}$ ▷ odstranění již roznásobeného stavu
end for



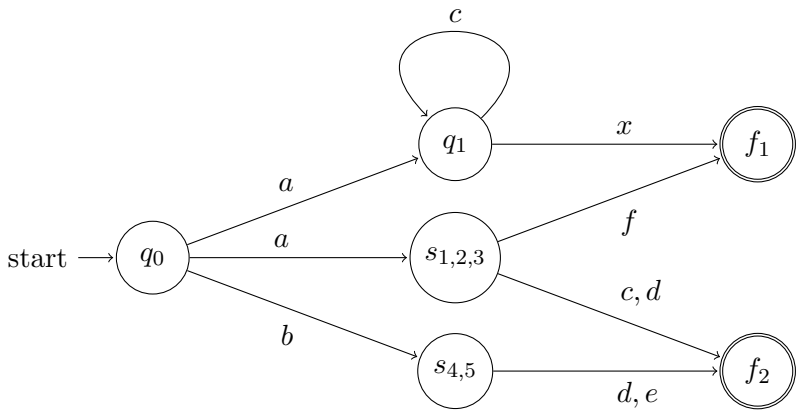
Obrázek 4.2: Roznásobení množiny stavů $\{q_2, q_3\ q_4\}$ z automatu $M_{4.1}$.

4.4 Použití Z3 na skupinu stavů

Na již roznásobeném automatu $M_{4.1}$, který je na obrázku 4.2 bude provedena minimalizace sloučením s pomocí Max SAT solveru. Na základě jazykových ekvivalencí sestavíme zpětné a dopředné ekvivalentní páry a následně definujeme proměnné optimalizovaného problému.

$B_{eq} = \{(s_1, s_2), (s_1, s_3), (s_2, s_3), (s_4, s_5)\}$
$F_{eq} = \{(s_3, s_4)\}$
Proměnné: s_1Bs_2, s_1Bs_3, s_2Bs_3, s_4Bs_5 a s_3Fs_4.
Podle pravidel uvedených v sekci 4.1 jsou definovány následující omezující podmínky:
$s_1Bs_3 \implies \neg s_3Fs_4$
$s_2Bs_3 \implies \neg s_3Fs_4$
$s_4Bs_5 \implies \neg s_3Fs_4$

Protože cílem Max SAT solveru je maximalizovat počet proměnný ohodnocených pravdou (True). Bude výsledek následující: $s_1Bs_2 = True$, $s_1Bs_3 = True$, $s_2Bs_3 = True$, $s_4Bs_5 = True$ a $s_3Fs_4 = False$. Maximální minimalizace bude dosaženo při sloučení stavů podle celé množiny B_{eq}.



Obrázek 4.3: Automat $M_{4.1}$ minimalizovaný pomocí roznásobení a sloučení podle Z3 solveru.

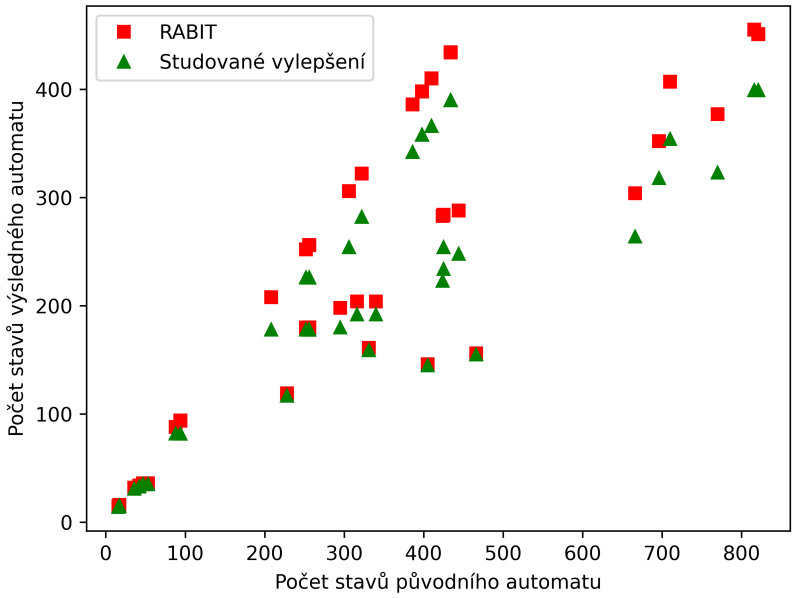
Z minimalizovaného automatu $M_{4.1}$ na obrázku 4.3 je patrné, že pokud by bylo slučování stavů prováděno bez výsledků solveru, mohli by být jako první sloučeny stavy s_3 a s_4 na základě dopředné jazykové ekvivalence. Výsledek minimalizace by nebyl tak efektivní. Automat by obsahovat o jeden stav více.

Kapitola 5

Experimentální výsledky

Výsledky porovnání efektivity minimalizace implementovaného algoritmu pracujícího na základě Z3 solveru a existujícího nástroje RABIT, jsou uvedeny v této kapitole. RABIT používá pro minimalizaci techniky slučování stavů, odstraňování hran přechodů a saturaci. Porovnávaný algoritmus obohacuje tyto postupy o optimalizační sílu Max Sat Z3 solveru. Testy byly prováděny na 99 automatech abstraktního regulárního model chackingu[1] vybraných podle teorému 3.7. Automaty obsahují celkem 33 011 stavů. Porovnávání bylo zaměřeno na výsledný počet stavů minimalizovaného automatu a počet provedených minimalizací (odebrání stavů).

5.1 Výsledný počet stavů



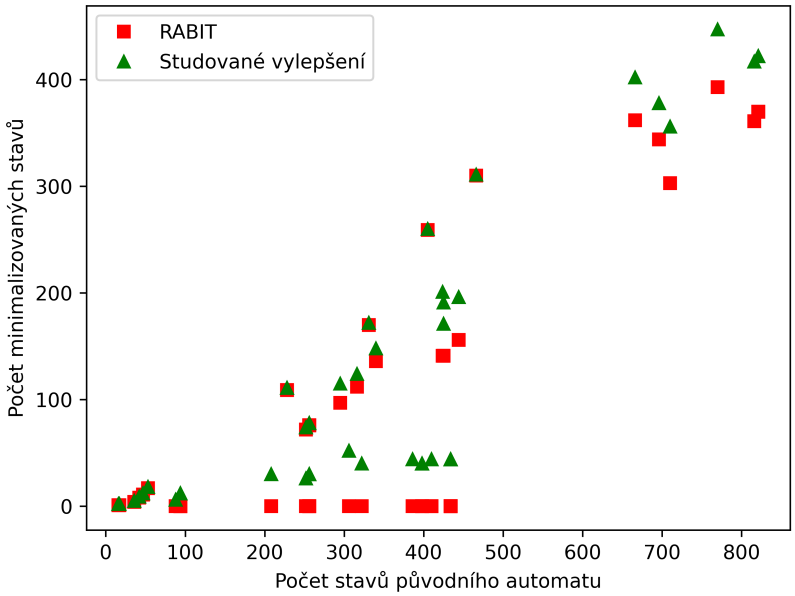
Obrázek 5.1: Porovnaní výsledného počtu stavů.

[1] dostupné z http://www.fit.vutbr.cz/~holik/pub/ARMCautomata.tar.gz

Při zaměření na velikost výsledného automatu (počet stavů), obsahovaly výstupy metody RABIT celkem 25 094 stavů. Oproti tomu testovaný algoritmus využívající Z3 solver minimalizoval automaty až na 22 530 stavů. V průměru minimalizoval vyvinutý algoritmus automat o dalších 10,2%.

5.2 Množství provedených minimalizací

Druhým zkoumaným statistickým údajem je množství stavů, o které byl výsledný automat minimalizován. Výstupy metody RABIT minimalizovaly automaty z původních 33 011 stavů na 25 094, tedy o 7 917 stavů. Algoritmus využívající solver minimalizoval automaty až na 22 530, což je o 2 564 stavů více než metoda RABIT. Vyvinutý postup minimalizoval v průměru o 32,4% více stavů než RABIT.



Obrázek 5.2: Porovnaní počtu minimalizovaných stavů.

Kapitola 6

Závěr

Protože nedeterministické konečné automaty disponují vlastnosti umožňující přechod z jednoho stavu do více následníku na základě stejného znaku, reprezentují jazyk menším množstvím stavů, než automaty deterministické, kde existuje vždy pouze jeden přechod. Nevýhodou nedeterministických konečných automatů je jejich obtížná minimalizace.

V současné době existují efektivní algoritmy pro minimalizaci nedeterministických automatů. Jsou jimi slučování stavů, odstraňování hran a saturace. Všechny tyto techniky implementuje nástroj RABIT, se kterým je vyvíjený minimalizační algoritmus porovnáván.

V automatu může existovat skupina stavů, kde je jazyk jednoho stavu postupně, ale celý, reprezentován ostatními stavy dané množiny. V takovém případě se může stát, že existující postupy nedovedou automat minimalizovat. Práce přináší techniku, která umožňuje minimalizaci takové skupiny stavů.

Vyvíjená minimalizační technika staví na vlastnosti slučování stavů. Po sloučení dvojice stavů, například podle jejich zpětné jazykové ekvivalence je možné výsledný stav použit ve slučování na základě zpětné jazykové ekvivalence všude tam, kde by byl použit některý ze stavů této dvojice. Takový stav ovšem není možné sloučit dále na základě dopředné jazykové ekvivalence založené na výpočtu jazykových relací před původním sloučením stavů. Pokud je některý stav možné sloučit na základě zpětné a zároveň dopředné jazykové ekvivalence, je využit Z3 solver pro určení nejoptimálnějšího postupu slučování, který povede na maximální počet sloučených párů.

Pro minimalizaci skupiny stavů, která se nachází v lokálním minimu je zapotřebí roznásobit jednotlivé stavy. Vznikne větší skupiny nových stavů se stejným jazykem, jako měla původní. Nyní ovšem má každý stav maximálně jednu vstupní a výstupní hranu. Nad těmito roznásobenými stavy je provedeno sloučení na základě výsledků Z3 solveru, které vede k maximální efektivitě.

Technika založená na Max SAT Z3 solveru, v kombinaci s dosavadními metodami, kterými jsou odstraňování hran přechodů a saturace, minimalizuje automat ve srovnání s nástrojem RABIT až o 32.4% stavů více.

Budoucím cílem je zobecnit postup minimalizace automatu založeného na podobnosti jazyka množiny stavů na podobnost jazyka množiny podgrafů, kde jazyk jednoho podgrafu je celý po částech obsažen v jazycích podgrafů z dané množiny. Dalším bodem je nalézt efektivnější kódování problému optimalizace slučování, aby byl výpočet SAT solveru rychlejší.

Literatura

[1] Chen, Y.-F. a Mayr, R. *RABIT/Reduce: Tools for language inclusion testing and reduction of nondeterministic Büchi automata and NFA*. Dostupné z: http://languageinclusion.org/doku.php?id=tools.

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

[3] Dill, D. L., Hu, A. J. a Wong Toi, H. Checking for language inclusion using simulation preorders. In: Larsen, K. G. a Skou, A., ed. *Computer Aided Verification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, s. 255–265. ISBN 978-3-540-46763-2.

[4] Ilie, L., Navarro, G. a Yu, S. On NFA reductions. In:. Leden 2004, sv. 3113, s. 112–124. DOI: 10.1007/978-3-540-27812-2_11. ISBN 978-3-540-22393-1.

[5] Ilie, L. a Yu, S. Algorithms for Computing Small NFAs. In:. Srpen 2002, s. 328–340. DOI: 10.1007/3-540-45687-2_27. ISBN 978-3-540-44040-6.

[6] Ilie, L. a Yu, S. Reducing NFAs by invariant equivalences. *Theor. Comput. Sci.* Září 2003, sv. 306, s. 373–390. DOI: 10.1016/S0304-3975(03)00311-6.

[7] Kim, H. a Choi, K.-I. A Pipelined Non-Deterministic Finite Automaton-Based String Matching Scheme Using Merged State Transitions in an FPGA. *PLOS ONE*. Říjen 2016, sv. 11. DOI: 10.1371/journal.pone.0163535.

[8] Meduna, A. a Lukáš, R. *Formal Languages and Compilers: Models for regular languages*. Božetěchova 1/2, 612 00 Brno-Královo Pole: FIT Brno University of Technology, 2017.

[9] Moura, L. de a Bjørner, N. Z3: an efficient SMT solver. In:. Duben 2008, sv. 4963, s. 337–340. DOI: 10.1007/978-3-540-78800-3_24. ISBN 978-3-540-78799-0.

[10] Norton, D. Algorithms for testing equivalence of finite automata, with a grading tool for JFLAP. Duben 2009, s. 10.

[11] Rabin, M. a Scott, D. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*. Duben 1959, sv. 3, s. 114–125. DOI: 10.1147/rd.32.0114.

[12] Sidhu, R. a Prasanna, V. Fast Regular Expression Matching Using FPGAs. In: *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing*

Machines. Los Alamitos, CA, USA: IEEE Computer Society, únor 2001, s. 227 – 238. ISBN 0-7695-2667-5.

[13] Sipser, M. Nondeterminism. In: *Introduction to the Theary of Computation*. 2. vyd. Boston: Thomson Course Technology, 2006. ISBN 0-534-95097-3.

[14] Sourdis, I. a Pnevmatikatos, D. Fast, Large-Scale String Match for a 10Gbps FPGA-Based Network Intrusion Detection System. In:. Září 2003, sv. 2778, s. 880–889. DOI: 10.1007/978-3-540-45234-8_85. ISBN 978-3-540-40822-2.

[15] Subramanian, S. a Thomas, T. Regular expression based pattern extraction from a cell - Specific gene expression data. *Informatics in Medicine Unlocked*. Listopad 2019, sv. 17, s. 100269. DOI: 10.1016/j.imu.2019.100269.