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 AUTOMATŮ

SURVIVAL OF THE STRONGER IN MINIMIZATION OF NONDETERMINISTIC AUTOMATONS

PROJEKTOVÁ PRAXE
PROJECT PRACTICE

AUTOR PRÁCE — **MICHAL ŠEDÝ**
AUTHOR

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

BRNO 2020

Abstrakt

Minimalizace nedeterministických automatů je vedena na základě výsledků výpočtů jazykových inkluzí. Tyto výpočty jsou náročné a pouhé minimum jejich výsledků poskytuje užitečné informace. Cenu výpočtů je možné snížit náhradou úplných algoritmů aproximační metodou (simulací). Nicméně využití získaných informací zůstává stejné. Tato práce přináší nové postupy minimalizace, které zvyšují využití získaných dat z výpočtu simulace. Definuje případy, kdy lze k minimalizaci použít stávající metodu slučování stavu, a to i v případě znalosti pouhé oboustranné jazykové inkluze. Hlavním přínosem jsou podmínky a algoritmy umožňující minimalizaci novým způsobem, a to odstraněním stavu. Použití odstraňování v kombinaci s dosavadními metodami využívajícími slučování výrazně zvýšilo efektivitu minimalizace a snížilo velikost výsledného automatu.

Abstract

The minimization of nondeterministic finite automatons is performed based on the results of language inclusions. This calculation is difficult and only a minimum of them are used in minimization. The price of the calculation can be decreased by using the approximation method (simulation). However, the usage of the results of language calculation is unchanged. This paper brings new approaches, which lead to increases in the usage of calculation. Defines cases, when states can be merged by the existing method, even if only bilateral language inclusion is known. The main benefits of this paper are conditions and algorithms, which allow to a minimized automaton by a new approach, by state deleting. Used of this approach and well-known state merging, led to a significant increase in minimization efficiency was achieved.

Klíčová slova

konečný automat, nedeterministický automat, minimalizace, jazyková inkluze, relace simulace, odstranění stavu

Keywords

finite automata, nondeterministic automata, minimization, language inclusion, simulation relation, state deleting

Citace

ŠEDÝ, Michal. *Přežití silnějšího v minimalizaci nedeterministických automatů*. Brno, 2020. 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) [7] se vyznačují schopností umožňující přechody do více stavů na základě stejného přijatého znaku. Deterministické automaty disponují pouze jedním přechodem pro znak. Nedeterministické konečné automaty umožňují reprezentovat větší jazyky za použití menšího množství stavů, než jejich deterministické varianty. Pro tuto důležitou vlastnost jsou nedeterministické automaty využívány při návrhu hardware reprezentujícího regulární výrazy, kde je kladen důraz na prostorovou náročnost a cenu. Jedná se například o síťová zařízení monitorující provoz na internetových linkách s rychlostmi pohybujícími se okolo 100Gbps a již kontrola prováděná procesorem v reálném čase není možná.

Na druhou stranu nedeterminizmus komplikuje minimalizaci automatu. U složitějších automatů nelze dokonce určit jejich nejmenší možnou formu. V současnosti existují účinné minimalizační algoritmy založené na metodě slučování ekvivalentních stavů, jak jej uvedli Ilie, Navaro a Yu [3]. Tyto algoritmy pracují na základě zkoumání jazykových inkluzí a na detekci dvou jazykově ekvivalentních stavů, které by mohly být sloučeny v jeden.

Úplný výpočet jazykových inkluzí bývá časově a prostorově náročný a v některých případech může vést k stavové explozi. Proto se používá aproximační metoda simulace [2]. Jedná se pouze o aproximaci, tedy některé existující jazykové inkluze nemusí být simulací detekovány. Následující práce určuje jazykové inkluze právě pomocí simulace.

Tato práce přináší nový postup rozšiřující možnosti minimalizace automatů sloučením na stavy v pouhé jazykové inkluzi. Kromě ekvivalentních stavů lze slučovat i stavy v jazykové inkluzi. Dále navazuje na předešlou práci o minimalizaci odstraněním [8] a prezentuje algoritmy a podmínky umožňující větší minimalizující NKA pomocí odstranění jednoho z dvojice minimalizovaných stavů, určených simulací. Zavedení tohoto postupu vede k výraznému zmenšení výsledného automatu a zefektivnění využití dat získaných výpočtem jazykových inkluzí.

Kapitola 2 obsahuje základní definice potřebné k pochopení problematiky NKA. Postup minimalizace sloučením pouze při znalosti jazykové inkluze nalezne čtenář v kapitole 3. Podmínky včetně algoritmů umožňujících minimalizaci odstraněním prezentuje kapitola 4. Získané experimentální výsledky jsou k dispozici v poslední kapitole 5.

Kapitola 2

Konečné automaty

Tato kapitola přináší základní informace týkající se problematiky nedeterministických konečných automatů (NKA) a také notaci potřebnou k porozumění následující práce. Kapitola je převzata z [9], kde původní definice 2.1.-4. byly převzaty z [4, 5] a definice 2.8.-10. z [6].

2.1 Prerekvizity

Definice 2.1 *Nechť je dána abeceda* Σ*, pak rekurzivní definice* ***řetězce nad abecedou*** *má následující tvar:*

1. ϵ *je řetězcem nad abecedou* Σ *a značí prázdný řetězec.*
2. *Pokud je* w *řetězcem nad abecedou* Σ*, pak je také* aw*, kde* $a \in \Sigma$*, řetězcem nad abecedou* Σ*.*

Definice 2.2 *Nechť* Σ^* *značí množinu všech řetězců nad* Σ*. Každá podmnožina* $L \subseteq \Sigma^*$ *je* ***jazyk*** *nad* Σ*.*

Jazyk představuje libovolnou množinu řetězců, které vznikly kombinací znaků dané abecedy. Přestože řetězce a abeceda nabývají konečných délek, jazyk může být nekonečný. Mějme kupříkladu abecedu $\{0, 1\}$, pak nad touto abecedou může existovat řetězec ve tvaru 01001. Jazyk může být definován třeba jako množina všech řetězců se sudým počtem znaků 1.

Definice 2.3 ***nedeterministický konečný automat*** je pětice $M = (Q, \Sigma, \delta, q_0, F)$

1. Q *je konečná neprázdná množina stavů,*
2. Σ *je vstupní abeceda,*
3. $\delta : Q \times \Sigma_\epsilon \longrightarrow 2^Q$ *je přechodová funkce,*
4. $q_0 \in Q$ *je počáteční stav,*
5. $F \subseteq Q$ *je konečná neprázdná množina koncových stavů.*

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

Definice 2.4 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak* ***konfigurace*** *je řetězec* $\chi \in Q\Sigma^*$

Konfigurace automatu znázorňuje informaci o aktuálním stavu, ve kterém se automat nachází, a zbytku vstupního řetězce čekajícího na zpracování. Například pokud se automat M nachází ve stavu q a na vstupu zbývá řetězec ab, pak nabývá konfigurace tvaru qab.

Definice 2.5 *Nechť* paw *a* qw *jsou dvě konfigurace NKA* M*, kde* $p, q \in Q$*,* $a \in \Sigma \cup \{\epsilon\}$ *a* $w \in \Sigma^*$*. Nechť* $r = pa \rightarrow q \in R$ *je pravidlo, potom* M *může provést* ***přechod*** *z* paw *do* qw *za použití* r*, zapsáno* $paw \vdash qw$ $[r]$ *nebo zjednodušeně* $paw \vdash qw$*.*

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

Nechť χ_0, χ_1, ...,χ_n, je sekvence konfigurací pro $n \leq 1$ a $\chi_{i-1} \vdash \chi_i$ $[r_i]$, $r_i \in R$ pro všechna $i = 1, \ldots, n$, což znamená: $\chi_0 \vdash \chi_1$ $[r_1] \vdash \chi_2$ $[r_2] \ldots \vdash \chi_n$ $[r_n]$. 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 nějaké $n \geq 1$, pak jej značíme $\chi_0 \vdash^+ \chi_n$ $[\rho]$.

Pokud $\chi_0 \vdash^n \chi_n$ $[\rho]$ pro nějaké $n \geq 0$, pak jej značíme $\chi_0 \vdash^* + \chi_n$ $[\rho]$.

Definice 2.6 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$ *a řetězec* w *nad abecedou* Σ*. Pak říkáme, že* w *je* ***řetězcem akceptovaným*** *[7] 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. $r_0 = q_0$
2. $r_{i+1} = \delta(r_i, y_{i+1})$ *pro* $i = 0, \ldots, n$-1
3. $r_n \subseteq F$

Podmínka 1. říká, že automat musí začít ve stavu q_0. Podmínka 2. vyžaduje, aby pro každý stav r_i existoval přechod do stavu r_{i+1} za přečtení znaku y_{i+1}. Poslední podmínka 3. říká, že řetězec w je akceptovatelný automatem M, pokud poslední stav r_n je zároveň koncovým stavem automatu.

Definice 2.7 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak* ***jazyk přijímaný*** *konečným automatem M je definován:* $L(M) = \{w |\ w \in \Sigma^*, q_0 w \vdash^* f,$ kde $f \in F\}$

Přijímaným jazykem NKA je taková množina řetězců, pro které platí, že po zpracování všech znaků se z počátečního stavu přes odpovídající hrany přechodů dostane automat do stavu koncového. V případě nedeterministického přechodu, kdy se může pomocí jednoho znaku přejít do více stavů, dojde k rozdělení vyhodnocování. Řetězec je poté akceptován, pokud alespoň jedna cesta dospěje do koncového stavu.

Definice 2.8 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak stav* $q \in Q$ *je* ***dostupným stavem****, pokud existuje* $w \in \Sigma^*$*, pro který platí* $q_0 w \vdash^* q$*. Jinak je nedostupný.*

Stavy automatu dělíme na dostupné a nedostupné v závislostí na existenci dopředné cesty mezi počátečním stavem a stavem zkoumaným. Pokud taková cesta neexistuje, pak se jedná o nedostupný stav. Nedostupné stavy nemají na funkci automatu žádný vliv, a proto

je možné tyto stavy odstranit. Přítomnost nedostupných stavů je pro minimalizaci výhodou. Metoda odstranění stavu, která je prezentována v kapitole 3, se snaží svým přístupem právě takové stavy vytvářet.

Definice 2.9 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak stav* $q \in Q$ *je* ***ukončujícím stavem****, pokud existuje* $w \in \Sigma^*$*, pro který platí* $qw \vdash^* f$*, kde* $f \in F$*. Jinak je neukončující.*

Stavy automatu je také možné členit do dvou kategorii podle existence dopředné cesty mezi zkoumaným stavem a stavem koncovým. Existence takové cesty přisuzuje stavu vlastnost ukončujícího stavu. Neukončující stavy nijak funkčně NKA neovlivňují, proto mohou být smazány. Obdobně jako u nedostupných stavů je i tato vlastnost využívána při aplikování minimalizační metody odstraněním.

Definice 2.10 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak* ***pravý jazyk stavu*** $q \in Q$ *konečného automatu* M *je definován:* $\overrightarrow{L}(q) = \{w_r |\ w_r \in \Sigma^*, qw_r \vdash^* f\}$

Pravý jazyk stavu q je množina všech řetězců, pomocí kterých lze dosáhnout z daného stavu do stavu koncového.

Definice 2.11 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak* ***levý jazyk stavu*** $q \in Q$ *konečného automatem* M *je definován:* $\overleftarrow{L}(q) = \{w_l |\ w_l \in \Sigma^*, q_0 w_l \vdash^* q\}$

Definice 2.12 *Automaty* M *a* N *jsou* ***ekvivalentní automaty****, pokud* $L(M) = L(N)$*.*

Dva automaty M a N jsou si ekvivalentní tehdy, pokud se rovnají jejich jazyky. Z tohoto tvrzení plyne, že všechny řetězce přijímané automatem M musí být zároveň přijímany automatem N a obráceně.

Definice 2.13 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak* ***simulace*** [2] *nad* M *je kvaziuspořádáním* $\preceq \subseteq Q \times Q$*, například* $p \preceq q$*, pro které platí*:

1. $q \in F \implies p \in F$ a zároveň
2. pro každé $pa \to p'$ existuje $qa \to q'$ a zároveň pletí $p' \preceq q'$.

Podmínka 1. vyjadřuje, že pokud je stav q koncovým, pak musí být koncový i stav p. Podmínka 2. 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.

Tvrzení 2.13 Existence simulace $r \preceq q$ implikuje $L(r) \subseteq L(q)$. Pokud existuje nad stavy p a q relace simulace, pak je jazyk stavu p podmnožinou jazyka stavu q. Opačné tvrzení ale neplatí. Případ, kdy je vypočítaná čistě jazyková inkluze, nemusí nutně znamenat, že zde existuje i relace simulace. Simulace může tedy některé jazykové inkluze "přehlédnout".

Kapitola 3

Minimalizace sloučením

Standartní způsob minimalizace NKA spočívá ve slučování jazykově ekvivalentních stavů [3]. Tato kapitola uvádí podmínky, díky nimž lze slučování aplikovat také na stavy, které jsou pouze v oboustranné jazykové inkluzi.

3.1 Automat se sloučenými stavy

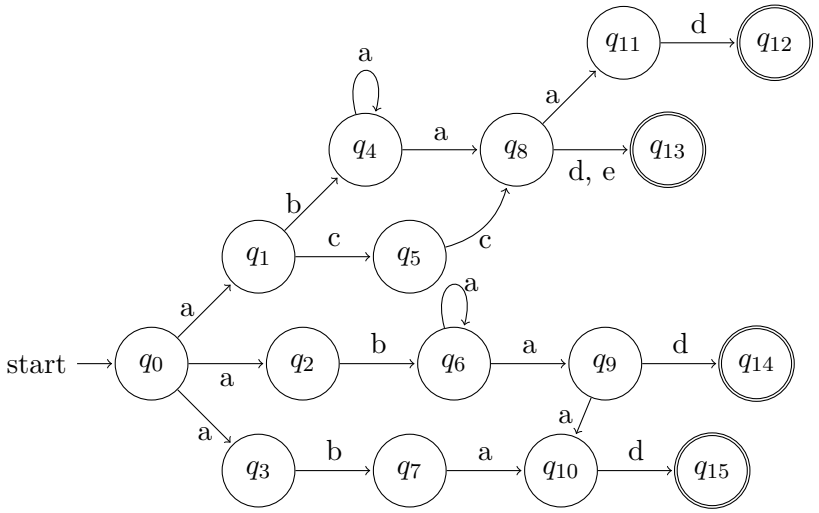
Minimalizace sloučením stavů vyžaduje převedení všech přechodů, které minimalizované stavy obsluhovaly, na nový stav vzniklý jejich sloučením. V praxi je slabší, nebo v případě rovnosti mohutnosti jazyků stavů náhodně zvolený stav, odstraněn a hrany přechodů jdoucí přes tento odstraněný stav jsou přesměrovány do zbylého silného stavu.

Definice 3.1 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$, *který je minimalizován. Ke sloučení jsou vybrány stavy* p, $q \in Q$. *Stav* p *bude sloučen do stavu* q. *Pak* ***automat vzniklý sloučením stavů*** *je definován jako* $M' = (Q', \Sigma, \delta', q_0, F')$, *kde*

1. $Q' = Q \backslash \{p\}$,
2. Σ *se nemění,*
3. $\delta'(a, s') = (\delta(a, s) \cap Q') \cup (\delta(a, p) \Leftrightarrow s' = q) \cup (\{\mathrm{q}\} \Leftrightarrow p \in \delta(a, s))$,
   pro s, p, $q \in Q$, $a \in \Sigma_\epsilon$, $s' \in Q'$
4. $q_0 \in Q'$ *zůstává počátečním stavem,*
5. $F' = F \cap Q'$.

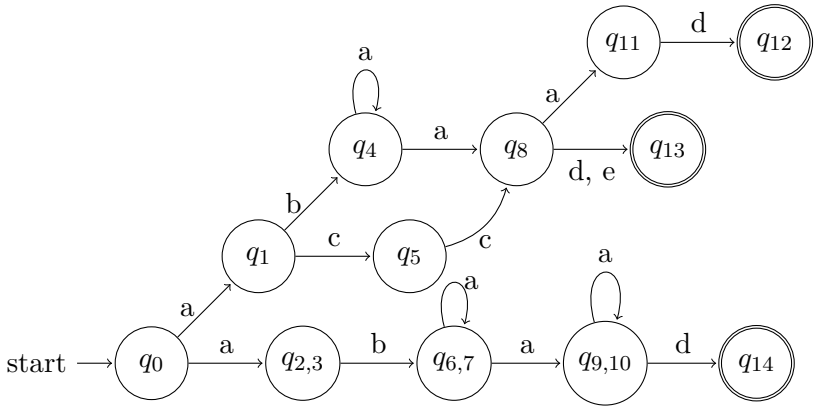
3.2 Sloučení na základě jazykové inkluze

Nechť je dán NKA M = $(Q, \Sigma, \delta, q_0, F)$. Pokud z výsledků výpočtů jazykových inkluzí vedených simulací 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_1 a M_1'.



Obrázek 3.1: automat M_1, sloučí se q_9 a q_{10}

Z automatu M_1 lze vidět, že je možno libovolně sloučit buď q_8 s q_9, nebo q_8 s q_{10}. Co již ale není na první pohled zřejmé, je nemožnost sloužit q_8 s q_{10} a poté q_8 s q_9, viz. automat M_1'.



Obrázek 3.2: automat M_1', sloučení stavů q_9 a q_{10} z upraveného automatu M_1

Z automatu M_1' 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_4 a q_9 by vedl v současné podobě automatu M_1' ke změně jazyka automatu.

Algorithm 1: Slučování při jazykové inkluzi

Inicializujte množinu $CLOSED$
$SIM_L \leftarrow$ simulace levých jazyků stavů automatu
$SIM_R \leftarrow$ simulace pravých jazyků stavů automatu
for all $q \succeq p \in SIM_R$ **do**
  if $q \succeq p \in SIM_L \wedge q \notin CLOSED \wedge p \notin CLOSED$ **then**
    $automat$.del(p)
    $CLOSED$.add(p)
  end if
end for

Každá dvojice stavů q a p, která je podle výsledků simulace v oboustranné jazykové inkluzi, $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$, může být minimalizovana sloučením stavů. K další minimalizaci se již nesmí použit ani jeden z těchto stavů.

Kapitola 4

Minimalizace odstraněním

Minimalizace odstraněním stavu přináší větší úsporu ve velikosti automatu. Ve srovnání se slučováním stavů je možné stavy zbylé po minimalizaci odstraněním za dodržení podmínek znovu použít k další minimalizaci, a to s použitím původní informace o jazykových inkluzích získaných z výpočtu simulace. Podkapitola s definicemi je převzata z dřívější práce o minimalizaci [10].

4.1 Definice pro odstranění

Definice 4.1 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak* ***jazyk mezi dvěma stavy*** p *a* $q \in Q$ *je definován:* $L(p, q) = \{w_b | \ w_b \in \Sigma^*, pw_b \vdash^* q\}$

Stávající jazykové definice jsou obohaceny o jazyk mezi stavy p a q, který představuje množinu řetězců, pomocí kterých se dá s využitím příslušných hran přechodů v automatu přejít ze stavu p do stavu q.

Pro popis případů, ve kterých se dá minimalizace provést odstraněním, je nutné ustanovit pomocný pojem. Je jím ryzí jazyk, a to jak levý, pravý, tak i mezi dvěma stavy. Ryzost jazyka rozšiřuje jeho vlastnosti o podmínku vynechání určitého stavu automatu při tvoření jeho řetězců. Například levý jazyk stavu q, který nevyužívá stav r, je taková množina řetězců, pomocí kterých je možné přejít ze stavu počátečního do stavu q, a to bez použití stavu r.

Definice 4.2 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak* ***ryze levý jazyk*** *stavu* $q \in Q$ *značený* $\overleftarrow{L}(q, \overline{p})$ *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,* $r_0 r_1 r_2 \ldots r_n \in Q \backslash \{p\}$*, pro kterou platí následující tři podmínky:*

1. $r_0 = q_0$
2. $r_{i+1} = \delta(r_i, y_{i+1})$ *pro* $i = 0, \ldots, n$-1
3. $r_n = q$

Definice 4.3 *Nechť je dán NKA* $M = (Q, \Sigma, \delta, q_0, F)$*, pak* ***ryze pravý jazyk*** *stavu* $q \in Q$ *značený* $\overrightarrow{L}(q, \overline{p})$ *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, $r_0 r_1 r_2 \ldots r_n \in Q \backslash \{p\}$, pro kterou platí následující tři podmínky:

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

Definice 4.4 *Nechť je dán NKA $M = (Q, \Sigma, \delta, q_0, F)$, pak* ***ryzí jazyk mezi stavy*** *p a q bez použití s, kde $p, q, s \in Q$ 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. $r_n = q$

Definice 4.5 *Nechť je dán NKA $M = (Q, \Sigma, \delta, q_0, F)$ a při výpočtu minimalizace je rozhodnuto, že minimalizace stavů $p, q \in Q$ bude provedena smazáním stavu p, pak výsledný* ***automat se smazaným stavem*** *je definován jako $M' = (Q', \Sigma, \delta', q_0, F')$, kde*:

1. $Q' = Q \backslash \{p\}$,
2. Σ *se nemění,*
3. $\delta'(a, q) = \delta(a, q) \cap Q'$, *pro* $q \in Q'$, $a \in \Sigma_\epsilon$,
4. $q_0 \in Q'$ *zůstává počátečním stavem,*
5. $F' = F \cap Q'$.

Automat s odstraněným stavem ve srovnání s původním automatem postrádá smazaný stav a také veškeré přechody s tímto stavem spojené. Tento způsob minimalizace nabízí možnost vzniku nedosažitelných nebo neukončujících stavů, které se dají snadno detekovat a odstranit, a tím přispět k větší minimalizaci. Vzhledem k tomu, že u automatu M vytvářela přechodová funkce δ množinu pravidel přechodů R, pak i nově vzniklá přechodová funkce δ' utváří množinu pravidel, nyní už ovšem značenou R'.

4.2 Odstranění v závislosti na cestě

V případě minimalizace dvou stavů p a q nedeterministického konečného automatu M mohou nastat celkem tři případy: mezi minimalizovanými stavy se nenachází žádná cesta, všechny cesty vedou pouze jedním směrem (například z p do q), nebo cesty vedou oběma směry (z p do q a opačně).

4.2.1 Žádná cesta

Teorém 4.2.1. *Nechť je dán NKA $M = (Q, \Sigma, \delta, q_0, F)$ a stavy $p, q \in Q$. Pokud platí $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$ a zároveň $\nexists\, w \in \Sigma^*$, pro které by platilo $qw\vdash^+p \vee pw\vdash^+q$, pak lze provést odstranění stavu p. Toto odstranění zachovává jazyk L(q) a L(M).*

Důkaz. Nechť je dán NKA $M = (Q, \Sigma, \delta, q_0, F)$ a stavy $p, q \in Q$, pro které platí $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$, a zároveň $\nexists\, w$ ve tvaru $w = y_1y_2\ldots y_n$, kde $y_i \in \Sigma$ pro $i = 1\ldots n$ a neexistuje taková sekvence stavů $r_0r_1\ldots r_n$, kde $r_i \in Q$, pro $i = 0\ldots n$, která by tvořila sekvenci přechodů $r_0y_1y_2\ldots y_n\vdash r_1y_2\ldots y_n\vdash\ldots\vdash r_{n-1}y_n\vdash r_n$, kde $r_0 = q \wedge r_n = p$, nebo $r_0 = p \wedge r_n = q$.

Při odstranění stavu p není změněn jazyk $L(q)$, protože stav p se nenachází mezi stavy tento jazyk definující. Pokud by tomu tak mělo být, pak by musela v sekvenci stavů $r_0r_1\ldots r_n$ definujících jazyk $L(q)$ existovat taková podsekvence stavů $r_jr_{j+1}\ldots r_k$ a existovat takový řetězec w_e ve tvaru $w_e = y_{j+1}\ldots y_k$, kde $y_m \in \Sigma$ pro $m = j+1\ldots$k, který by spolu se stavy tvořil sekvenci přechodů $r_jy_{j+1}y_{j+2}\ldots y_k\vdash r_{j+1}y_{j+2}\ldots y_k\vdash\ldots\vdash r_{j-1}y_k\vdash r_k$, kde $r_j = q \wedge r_k = p$, nebo $r_j = p \wedge r_k = q$. Existence této sekvence přechodů by byla v přímém rozporu se vstupní podmínkou neexistence cesty mezi stavy q a p. Odstranění stavu p nemá na $L(q)$ žádný vliv, protože stav p není součástí stavů tento jazyk definujících.

Jazyk přijímaný automatem M je všeobecně definován jako množina řetězců w ve tvaru $w = y_1y_2\ldots y_n$, kde $y_i \in \Sigma$ pro i = 1…n a existuje taková sekvence stavů $r_0r_1\ldots r_n$, kde $r_i \in Q$, pro $i = 0\ldots n$, pro které platí následující trojice podmínek:

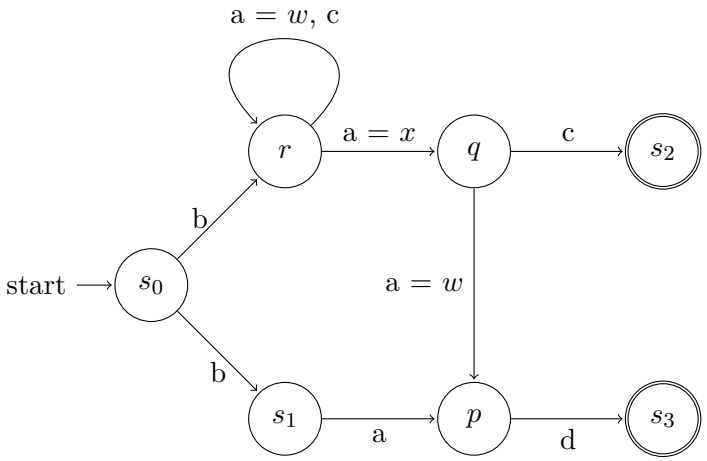
1. $r_0 = q_0$
2. $r_{i+1} = \delta(r_i, y_{i+1})$ *pro* $i = 0 \ldots n$-1
3. $r_n \in F$

Odstraněním stavu p jsou z množiny sekvencí stavů definujících jazyk automatu odebrány takové sekvence $r_0r_1\ldots r_n$, kde $p = r_k$ pro $k \in [0; n]$. Protože odstranění stavu p nemá vliv na jazyk stavu q a z podmínky $L(p)$ plyne, že $\forall\, w_p \in L(p)$ $\exists$ sekvence stavů $r_0r_1\ldots r_n$, kde $q = r_k$ pro $k \in [0; n]$, která přejímá řetězec w_p, pak jsou všechny řetězce, které byly přijímané přes odstraněný stav p také přijímané přes neovlivněný stav q a jazyk automatu M není změněn. □

4.2.2 Jednosměrná cesta

Lemma 4.2.2 (O zpětné smyčce)**.** *Nechť je dán NKA $M = (Q, \Sigma, \delta, s_0, F)$. Dále dvojice stavů $p, q \in Q$, pro které platí $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$, pak $\forall\, w \in \Sigma^*$ takové, že platí $qw\vdash^+p$ a zároveň neplatí $pw\vdash^+q$, $\exists\, r \in Q$ a $x \in \{w\}^* \wedge x \in L(r, q)^*$, pro které platí $rw\vdash^+r \wedge rx\vdash^+q$, kde $w \in L(r, r)$.*

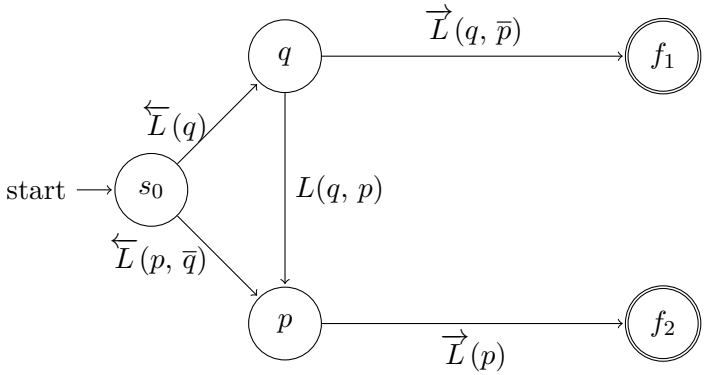
Lemma o zpětné smyčce říká, že pokud existuje cesta ze stavu q do p, pro které platí $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$ a již neexistuje stejná cesta zpět, pak v levém jazyce stavu q existuje taková smyčka, která umožňuje příjem řetězce totožného s řetězcem reprezentovaným cestou mezi stavy. Existenci smyčky demonstruje obrázek 4.1.



Obrázek 4.1: existence smyčky v levém jazyku

Obdobným případem je Lemma 4.2, kdy vede cesta ze slabšího stavu p do silnějšího stavu q.

Důkaz. Nechť je dán NKA $M = (Q, \Sigma, \delta, s_0, F)$ a dvojice stavů $p, q \in Q$, pro které platí $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$ a zároveň $\exists\, w \in \Sigma^*$ takové, že platí $qw\vdash^+p$ a zároveň neplatí $pw\vdash^+q$, tedy je možné s použitím řetězce w přejít ze stavu q do stavu p, ale již ne stejným řetězcem zpět. Situaci modeluje obrázek 4.3 zaměřený na část jazyků automatu obsahujících stavy q a p.



Obrázek 4.2: rozložení jazyků

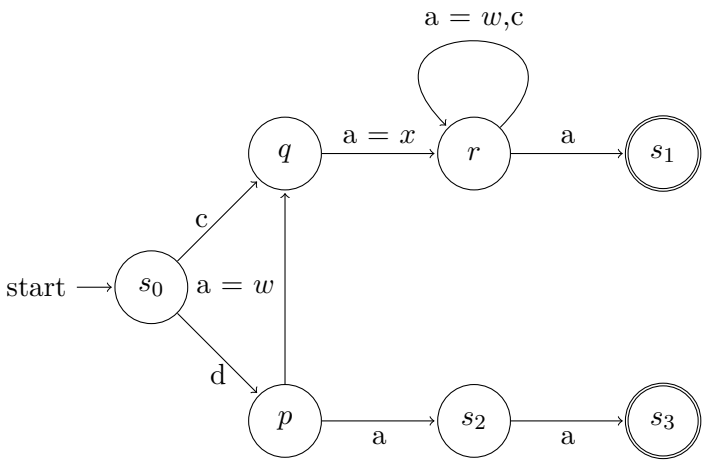
Protože $\overleftarrow{L}(p)$ se skládá ze dvou jazyků, jazyka procházejícího stavem q a jazyka $\overleftarrow{L}(p, \overline{q})$, který podle definice stavem q neprochází, pak platí: $\overleftarrow{L}(p) = \overleftarrow{L}(p, \overline{q}) \cup \overleftarrow{L}(q){\cdot}L(q, p)$. Ze vstupní podmínky $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$ vyplývá, že $\overleftarrow{L}(p, \overline{q}) \cup \overleftarrow{L}(q){\cdot}L(q, p) \subseteq \overleftarrow{L}(q)$. Ryzí levý jazyk stavu p, $\overleftarrow{L}(p, \overline{q})$, není pro existenci smyčky v levém jazyce stavu q podstatný, protože podmínka ryzosti jazyka $\overleftarrow{L}(p, \overline{q})$ zakazuje výskyt stavu q mezi stavy definujícími tento jazyk. Podstatnou částí zůstává $\overleftarrow{L}(q){\cdot}L(q, p) \subseteq \overleftarrow{L}(q)$. Pro každý řetězec $w_q \in \overleftarrow{L}(q)$ délky n, který je ve tvaru $y_1y_2 \ldots$n, kde $y_i \in \Sigma$, pro $i = 1 \ldots n$ a pro který existuje sekvence stavů $r_0r_1 \ldots r_n$, která tvoří sekvenci přechodů $r_0y_1y_2 \ldots y_n\vdash r_1y_2 \ldots y_n\vdash \ldots \vdash r_{n-1}y_n\vdash r_n$, kde $r_0 = s_0 \wedge r_n = q$, musí automat projít $n + 1$ stavy. Protože platí $\overleftarrow{L}(q){\cdot}L(q, p) \subseteq \overleftarrow{L}(q)$, pak stejná skupina stavů počtu n+1, která dokázala přijmout řetězec w_q délky n, musí být schopna na

nezměněném počtu stavů přijmout řetězec $w_q w_{qp}$ délky $n+m$, kde $w_{qp} \in \overleftarrow{L}(q) \cdot L(q, p)$ a m je délka řetězce w_{qp}. Přes některé stavy, které definují konec $\overleftarrow{L}(q)$, se musí projít vícekrát. Jediným možným způsobem, jak může stejná sekvence stavů přijímat řetězec w_q a také $w_q w_{qp}$, je takové uspořádání stavů, které přijímají regulární výraz $w_{pref} w_{qp}^*$, kde w_{pref} je prefixem řetězce w_q. To implikuje existenci smyčky ke konci levého jazyka stavu q. Pro jeho levý jazyk platí $\overleftarrow{L}(q) = \overleftarrow{L}(q) \cdot L(q, p)^*$. □

Obdobný stranově převrácený důkaz by bylo možné vést pro opačnou cestu mezi stavy, tentokrát ze stavu p do stavu q.

Lemma 4.2.3 (O dopředné smyčce)**.** *Nechť je dán NKA $M = (Q, \Sigma, \delta, s_0, F)$. Dále dvojice stavů $p, q \in Q$, pro které platí $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$, pak $\forall\ w \in \Sigma^*$ takové, že platí $pw \vdash^+ q$ a zároveň neplatí $qw \vdash^+ p$, $\exists\ r \in Q$ a $x \in \{w\}^* \wedge x \in L(q, r)^*$, pro které platí $rw \vdash^+ r \wedge qx \vdash^+ r$, kde $w \in L(r, r)$.*

Lemma o dopředné smyčce říká, že pokud existuje cesta ze stavu p do q, pro které platí $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$ a již neexistuje stejná cesta zpět, pak v pravém jazyce stavu q existuje taková smyčka, které umožňuje přijímat řetězec totožný s řetězcem reprezentovaným cestou mezi stavy. Existenci smyčky demonstruje obrázek 4.2 zaměřený na část jazyků automatu obsahujících stavy q a p.

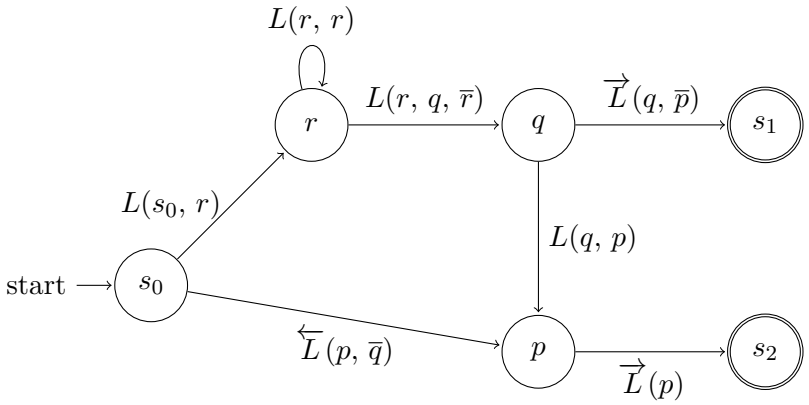


Obrázek 4.3: existence smyčky v pravém jazyce

V důsledku Lemmat o smyčce není potřeba žádná kontrola při odstraňování slabého stavu na základě jazykové inkluze, kde mezi minimalizovanými stavy existuje pouze jednosměrná cesta uskutečnitelná řetězcem w.

Teorém 4.2.4. *Nechť je dán NKA $M = (Q, \Sigma, \delta, q_0, F)$ a stavy $p, q \in Q$. Pokud platí $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$ a pro všechny $w \in \Sigma^*$ platí současně pouze $qw \vdash^+ p$, nebo pouze $pw \vdash^+ q$, tedy všechny cesty mezi q a p vedou stejným směrem, pak lze provést odstranění stavu p. Toto odstranění zachovává jazyk $L(q)$ a $L(M)$, ale může ovlivnit jeden z jazyků $\overleftarrow{L}(q)$, nebo $\overrightarrow{L}(q)$.*

Důkaz. Nechť je dán NKA $M = (Q, \Sigma, \delta, q_0, F)$, pro který platí, $\overleftarrow{L}(q) \subseteq \overleftarrow{L}(p) \wedge \overrightarrow{L}(q) \subseteq \overrightarrow{L}(p)$ a existují takové řetězce $w \in \Sigma^*$, pro které platí $qw\vdash^+p$ a zároveň neexistují řetězce w_e, pro které by platilo $pw_e\vdash^+q$, tedy existuje cesta ze stavu q do stavu p pomocí řetězce w, ale již neexistují cesty z p do q.



Obrázek 4.4: rozložení jazyků při jednosměrné cestě

Pro pravý jazyk stavu p platí: $\overrightarrow{L}(p) \subseteq L(q, p)^N \cdot \overrightarrow{L_{postf}}(p)$. Pro pravý jazyk stavu q pak platí: $\overrightarrow{L}(q) \subseteq L(q, p)^{N+1} \cdot \overrightarrow{L_{postf}}(p) \cup \overrightarrow{L}(q, \overline{p})$. Aby platilo $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$, musí $\overrightarrow{L}(q)$ přijímat také $\overrightarrow{L_{postf}}(p)$, z čehož plyne $\overrightarrow{L}(q, \overline{p}) \supseteq \overrightarrow{L_{postf}}(p)$.

Podle Lemmatu o zpětné smyčce existuje takový stav $r \in Q$ a $x \in \{w\}^* \wedge x \in L(r, q)^*$, pro který platí $rw\vdash^+r \wedge rx\vdash^+q$.

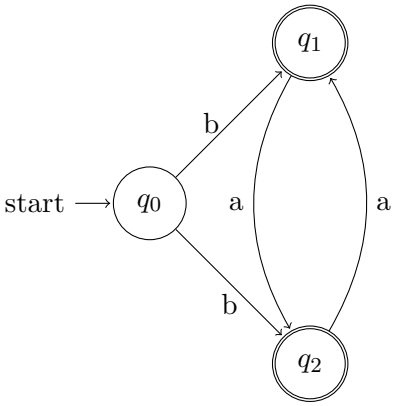
Pro jazyk stavu q tedy platí: $L(q) = L_{pref}(q) \cdot L(r, r) \cdot L(r, q, \overline{r}) \cdot \overrightarrow{L}(q)$. Podle Lemmatu o zpětné smyčce platí: $L(r, r) = L(q, p)^*$, $L(r, q, \overline{r}) = L(q, p)^M$, pak lze $L(q)$ zapsat ve tvaru: $L(q) = L_{pref}(q) \cdot L(q, p)^* \cdot L(q, p)^M \cdot (L(q, p)^{N+1} \cdot \overrightarrow{L_{postf}}(p) \cup \overrightarrow{L}(q, \overline{p}))$. Mezi stavy definujícími $\overleftarrow{L}(q)$ se nenachází odstraňovaný stav p. Tento fakt plyne z podmínky neexistence řetězce w_e a konfigurace, pro kterou by platilo $pw_e\vdash^+q$. Nechť je jazyk po odstranění stavu p označen $L_{-p}(q)$. Pro tento jazyk platí: $L_{-p}(q) = L_{pref}(q) \cdot L(q, p)^* \cdot L(q, p)^M \cdot \overrightarrow{L}(q, \overline{p})$. Protože $\overrightarrow{L}(q, \overline{p}) \supseteq \overrightarrow{L_{postf}}(p)$, $L(q, p)^* = L(r, r)$ a $L(q, p)^M = L(r, q)$, může být prohlášeno, že $L_{-p}(q) = L(q)$.

Protože bylo dokázáno, že odstraněním stavu p není jazyk q ovlivněn a platí $\overleftarrow{L}(q) \subseteq \overleftarrow{L}(p) \wedge \overrightarrow{L}(q) \subseteq \overrightarrow{L}(p)$, pak pro každou sekvenci stavů $r_0r_1 \ldots r_n$ definující $L(M)$, která příjmá řetězec $w_p \in L(p)$, kde $r_k = p$ pro $k \in [0; n]$, existuje sekvence stavů $r_0r_1 \ldots r_n$ přijímající stejný řetězec $w_q \in L_{-p}(q)$, kde $w_p = w_q$. $L(M)$ není odstraněním stavu p při existenci jednosměrných cest změněn. □

4.2.3 Obousměrná cesta

Teorém 4.2.5. *Nechť je dán NKA $M = (Q, \Sigma, \delta, q_0, F)$ a stavy $p, q \in Q$. Pokud platí $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$ a $\exists\ w \in \Sigma^*$, pro které platí $qw\vdash^+p \wedge pw\vdash^+q$. Pak nelze provést odstranění stavu p, protože by nebyl zachován $L(M)$.*

Pokud mezi stavy p a q existuje stejné cesta tam i zpět, pak nelze s jistotou provést odstranění slabšího stavu, protože by automat mohl přijít o tuto velkou smyčku reprezentovanou obousměrnou cestou mezi minimalizovanými stavy, viz automat M_2. V případě stejné obousměrné cesty se dá minimalizace provést pouze sloučením stavů. V praxi je výskyt stejné obousměrné cesty vzácný.

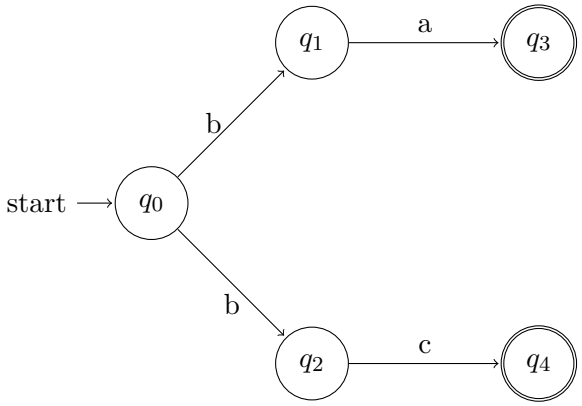


Obrázek 4.5: automat M_2, obousměrná cesta

Odstraněním slabšího stavu při existenci stejné obousměrné cesty by automat přišel o část svého jazyka. Tento problém obousměrných cest se vyskytuje jak při ekvivalenci jazyků, tak i při jazykové inkluzi. Výskyt stejné obousměrné cesty se musí kontrolovat při každé minimalizaci metodou odstranění, a pokud je nalezen řetězec w, pro který platí $qw\vdash^+ p \wedge pw\vdash^+ q$, dá se tedy dostat z jednoho stavu do druhého a zpět, pak se musí minimalizace provést sloučením při dodržení požadovaných podmínek zvolené metody. Pokud by se odstranil v automatu M_2 stav q_2, nemohl by již automat přijímat řetězce $ba*$.

4.3 Jednostranná ekvivalence

V případě znalosti pouze jednostranné jazykové ekvivalence, tedy $\overrightarrow{L}(q) \subseteq \overrightarrow{L}(p) \wedge \overrightarrow{L}(q) \supseteq \overrightarrow{L}(p)$, ale již není známo $\overleftarrow{L}(q) \subseteq \overleftarrow{L}(p)$, ani $\overleftarrow{L}(q) \supseteq \overleftarrow{L}(p)$, nebo obráceně, není možné minimalizaci odstraněním provést. Odstranění stavu by vedlo ke ztrátě jazyka smazaného stavu, který ale není jinde v automatu obsažen, viz. automat M_4



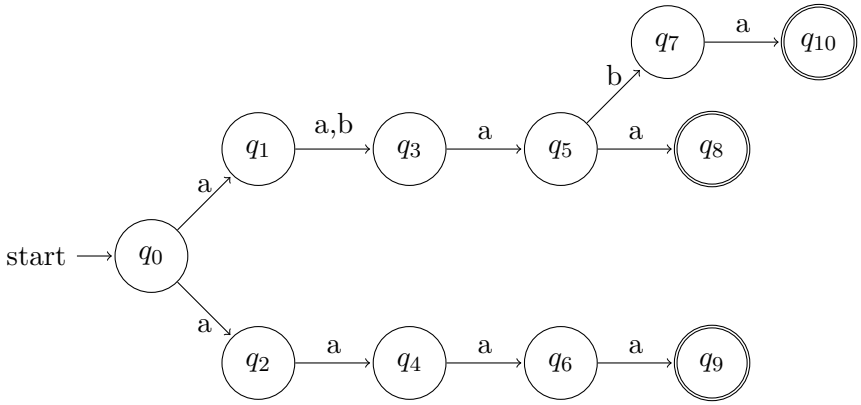
Obrázek 4.6: automat M_4, jednostranná jazyková ekvivalence

Z automatu M_4 je zřejmé, že odstranění jednoho ze stavů q_2, nebo q_3 není možné z důvodu nezachování jazyka automatu. Pravé jazyky stavů p a q jsou různé. Odstraněním jednoho stavu z této dvojice by výsledný jazyk přišel o část jazyka definovanou smazaným stavem.

4.4 Oboustranná inkluze

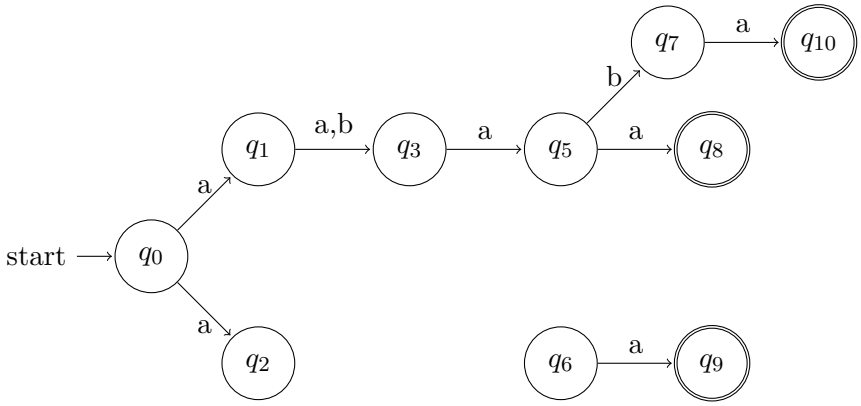
Pokud je v případě minimalizace stavů q a p stav p slabší, tedy $\overrightarrow{L}(q) \subseteq \overrightarrow{L}(p) \wedge \overleftarrow{L}(q) \supseteq \overleftarrow{L}(p)$, pak je možné s dodržením kontroly pro obousměrnou cestu mezi stavy odstranit slabší stav. Je odstraněn stav p a vzniká automat s odstraněným stavem. K minimalizaci automatu je použit algoritmus čistého odstranění.

Příklad 3.1 [8] Nechť je dán NKA $M_3 = (Q, \Sigma, \delta, q_0, F)$, viz. obrázek 4.7.



Obrázek 4.7: automat M_3

U automatu M_3 již bylo vyhodnoceno, že na stavy q_3 a q_4 se dá aplikovat odstranění. Je zrušen stav q_4. Výsledný automat s odstraněným stavem $M_3' = (Q', \Sigma, \delta', q_0, F')$ je na obrázku 4.7.

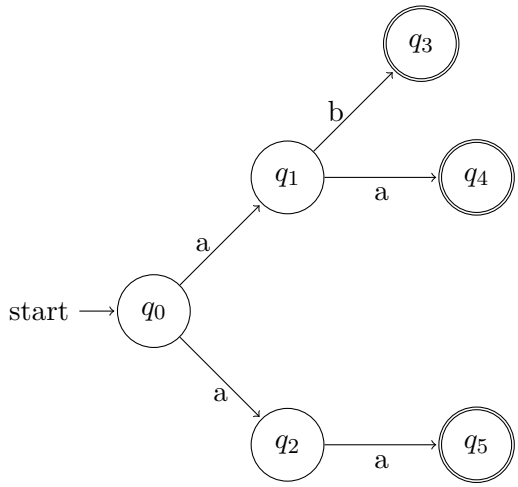


Obrázek 4.8: automat s odstraněním M_3'

Z automatu M_3' je zřejmé, že odstranění stavu nabízí potenciální vznik nedosažitelných a neukončujících stavů, které je již jednoduché odstranit. Této vlastnosti využívá pro urychlení minimalizace algoritmus proudového odstranění.

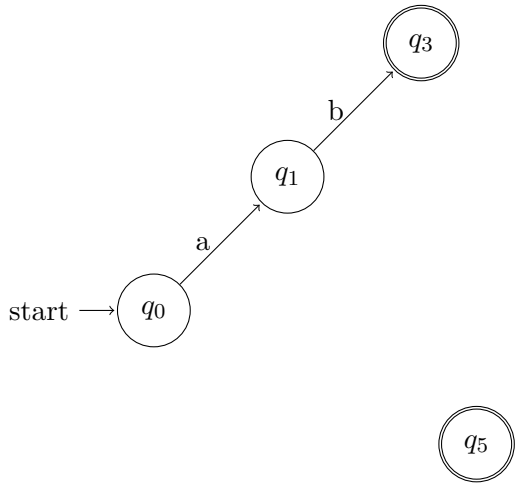
4.5 Oboustranná ekvivalence

V případě, že ani jeden stav není vysloveně slabší, mezi levými a pravými jazyky stavů q a p je oboustranná ekvivalence, $\overrightarrow{L}(q) \subseteq \overrightarrow{L}(p) \wedge \overrightarrow{L}(q) \supseteq \overrightarrow{L}(p) \wedge \overleftarrow{L}(q) \subseteq \overleftarrow{L}(p) \wedge \overleftarrow{L}(q) \supseteq \overleftarrow{L}(p)$, pak není možné provést bezpečně minimalizaci odstraněním za použití algoritmu čistého odstranění. Metoda neví, jaká větev bude ve výsledku použita, viz. automaty M_4 a M_4'. Tento nedostatek je odstraněn metodou proudového odstranění, která je informovaná o využívání jednotlivých větví automatů. Díky této metodě je možné odstranění aplikovat i na stavy v oboustranné jazykové ekvivalenci.



Obrázek 4.9: automat M_4, budou odstraněny stavy q_4 a q_2

Na základě znalosti oboustranné jazykové ekvivalence stavů q_5 a q_4 byl k odstranění vybrán stav q_4. Dále podle jazykové inkluze mezi stavy q_1 a q_2 byl k odstranění vybrán stav q_2 (slabší). Vznikl automat M_4' s chybně odstraněným stavem.



Obrázek 4.10: automat M_4', automat o odstraněnými stavu q_4 a q_2

Z automatu M_4' lze vidět, že odstranění stavu q_4 a ponechání q_5 na základě informace o oboustranné jazykové ekvivalenci sice minimalizovalo automat správně, ale jinak bezpečné odstranění slabého stavu q_2 vedlo ke ztrátě větve ponechané z prvního odstranění. Měl být ponechán stav q_4. Ale jak vědět, který stav ponechat a který odstranit? Řešení této otázky přináší informovaná metoda proudového odstraňování.

4.6 Postup minimalizace čistým odstraněním

Algorithm 2: Minimalizace čistým odstraněním

Inicializujte množinu $CLOSED$
$SIM_L \leftarrow$ simulace levých jazyků stavů automatu
$SIM_R \leftarrow$ simulace pravých jazyků stavů automatu
for all $q \succeq p \in SIM_R$ **do**
  if $q \succeq p \in SIM_L \wedge q \notin CLOSED \wedge p \notin CLOSED$ **then**
    if $p \succeq q \notin SIM_R \vee p \succeq q \notin SIM_L$ **then**
      if $\nexists\, w \in \Sigma_\epsilon^* \implies \neg(qw\vdash^* p \wedge pw\vdash^* q)$ **then**
        $automat$.del(p)
        $CLOSED$.add(p)
      end if
    end if
  end if
end for

Nejdříve jsou za pomocí simulace vypočteny inkluze levých a pravých jazyků všech stavů automatu. Do seznamu $CLOSED$ se budou vkládat již odstraněné stavy. Protože metoda čistého odstranění minimalizuje pouze stavy, jejichž jazyky jsou ve stavu oboustranné inkluze, jsou dvojice potenciálně minimalizovatelných stavů vybírány ze SIM_R (může být i SIM_L), který obsahuje dvojice q a p, pro jejichž pravý jazyk platí $q \succeq p$. Stav p je odstraněn, pokud se dvojice $q \succeq p$ nachází i v SIM_L (oboustranná jazyková inkluze). Jazyky stavů nejsou vzájemně ekvivalentní, neexistuje takový řetězec w, kterým by se dalo přejít

z jednoho stavu do druhého a zpět a zároveň se stav p ani q nenachází v seznamu $CLOSED$. V případě odstranění stavu se p vloží do seznamu $CLOSED$.

4.7 Metoda proudového odstranění

Jedná se o informovanou metodu, která zabraňuje vzniku chyby při minimalizaci odstraněním oboustranně jazykově ekvivalentních stavů.

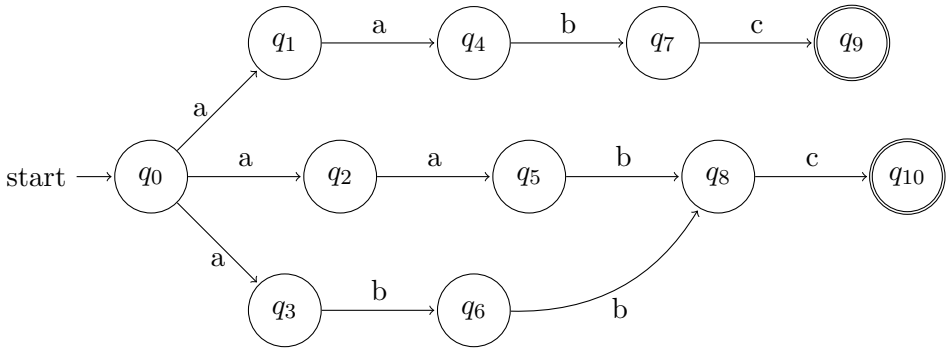
Algorithm 3: Proudové odstranění

Require: minimalizovaný automat A
Inicializujte množiny $DELETED_1$ a $DELETED_2$
$DELETED_1 \leftarrow$ jednosmerny_pruchod(A)
$DELETED_2 \leftarrow$ jednosmerny_pruchod(reversed(A))
for all $p \in DELETED_1 \cap DELETED_2$ **do**
  A.del(p)
end for

Algorithm 4: Jednosměrný průchod

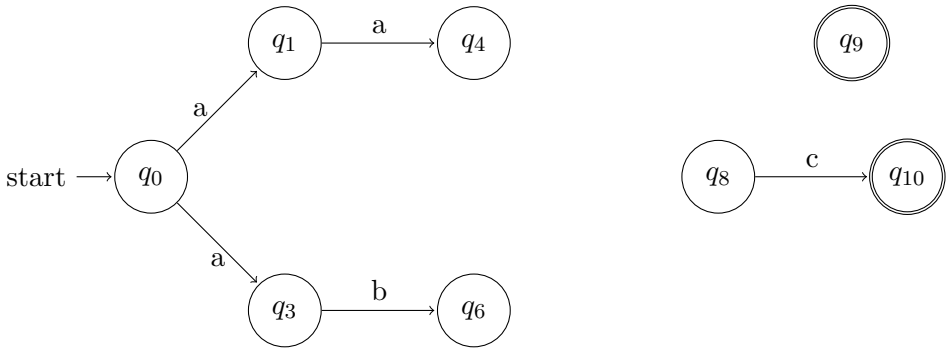
Inicializujte množiny $CLOSED$ a $DELETED$
Inicializujte frontu $OPEN$
Vytvořte kopii M minimalizovaného automatu
$SIM_L \leftarrow$ simulace levých jazyků stavů automatu
$SIM_R \leftarrow$ simulace pravých jazyků stavů automatu
$WEAK \leftarrow$ všechny stavy q, které jsou v oboustranné jazykové inkluzi s nějakým stavem p, ale nejsou se stavem p v oboustranné jazykové ekvivalenci
$OPEN \leftarrow$ počáteční stavy automatu M
while $OPEN \neq \emptyset$ **do**
  $Q \leftarrow OPEN$.pop()
  if $Q \in CLOSED \vee Q \in DELETED \vee Q \in WEAK$ **then**
    continue
  end if
  $CLOSED$.add(Q)
  for all $Q \succeq p \in SIM_R$ **do**
    if $Q \succeq p \in SIM_R$ **then**
      if $\nexists\, w \in \Sigma_\epsilon^* \implies \neg(Qw\vdash^* p \wedge pw\vdash^* Q)$ **then**
        M.del(p)
        $DELETED$.add(p)
        $OPEN$.add(rozgenerujte stav Q)
      end if
    end if
  end for
end while
return $DELETED$

Hlavní myšlenkou algoritmu je předem označit slabé stavy, přes které se při průchodu automatem nesmí překročit, protože existuje silnější stav, kterým s vysokou pravděpodobností bude slabý uzel později odstraněn. Dále se označují již prošlé stavy, není tedy možné, aby odstranění stavu mělo za následek znepřístupnění větve automatu, přes kterou již algoritmus jednou prošel, a tudíž se s ní ve výsledném minimalizovaném automatu počítalo. Pokud algoritmus narazí na slabý uzel, jde jinam, nebo se průchod touto větví ukončí. Pokud narazí na silný uzel, který je v oboustranné ekvivalenci, odstraní jeho protivníka a postupuje po aktivní zbylé větvi. Tím je zajištěno, že je postup informovaný a nemůže tak dojít k chybě jako v případě automatu M_4. Jeden průchod minimalizovaným automatem ale může mylně odstranit stavy, které jsou ve výsledném automatu potřebné. Jeden průchod není schopen se vypořádat se sbíhajícími se větvemi, viz. automat M_5. Tento problém je řešen korekcí, která spočívá v druhém, opačném průchodu po původním minimalizovaném automatu. Ve výsledku jsou odstraněny pouze ty stavy, které byly odstraněny v obou průchodech. Využití oboustranného průchodu při proudovém odstranění je nutností. Pokud by nebyla provedena korekce, mohlo by dojít k vyříznutí sbíhající se větve jako v případě automatu M_5.



Obrázek 4.11: automat M_5, vznik jednoprůchodové chyby

Při jednosměrném průchodu automatem M_5 je stav q_7 označen jako slabý. Postupně se procházejí všechny stavy. U stavu q_1 se rozhodne o odstranění stavu q_2, u stavu q_3 se rozhodne o odstranění stavu q_5, u stavu q_7 se větev zastaví, protože narazila na weak uzel. Ze stavu q_6 se zatím postoupí do stavu q_8 a odstraní se slabý stav q_7. Tím se teoreticky zničí cesta *aabc*, viz. automat M_5'. Z tohoto důvodu musí být provedena korekce zpětným průchodem. Při zpětném průchodu se postup horní větví zastaví hned u stavu q_7, a tudíž nebude horní větev procházena. Ve výsledném průniku stavů vhodných k odstranění zůstane pouze q_7. Horní větev bude později odstraněna z důvodu své neukončující povahy.



Obrázek 4.12: automat M'_5, automat se smazanými stavy bez korekce

4.8 Nebezpečí simulace

Jak již bylo dříve uvedeno, není možné bezpečně provádět odstranění stavu p z dvojice $q \succeq p$, kde jazyky těchto stavů jsou oboustranně ekvivalentní (ekvivalentní levé a ekvivalentní pravé jazyky). Pomocí vhodných podmínek je možné se tomuto nebezpečnému případu vyhnout. Naneštěstí v důsledku používání simulace k výpočtu jazykových inkluzí, která je aproximací a nemusí veškeré inkluze odhalit, není možné přesně určit, zda byly detekovány veškeré nebezpečné ekvivalentní stavy. S touto neznalostí neumí pracovat ani proudové odstranění a může dojít k chybě. Použití úplných algoritmů pro výpočet jazykových inkluzí by tento problém odstranilo. Nicméně i při použití simulace se zmíněný chybový stav vyskytuje pouze ve zlomku případů. Výsledný automat musí být sice při použití simulace pro jistotu kontrolován na nezměněný jazyk. Tuto nepříjemnou vlastnost ale daleko převyšují výsledky metody odstranění, která přináší až dvacetisedmiprocentní nárust efektivity minimalizace. Chyba neodhalení ekvivalence je vyobrazena automaty M_5 a M'_5.

Možností, jak minimalizovat chyby vnesené simulací do výsledného automatu, je použití postupu minimalizace čistým odstraněním, který je zpřísněn o podmínku zakazující jakoukoliv ekvivalenci mezi minimalizovanými stav. Pravděpodobnost výskytu chyby způsobené selháním simulace při kontrole levých a současně pravých jazyků minimalizované dvojice stavů je již zanedbatelná.

Dalším možným přístupem, který eliminuje anomálie vzniklé výpočtem simulace, je pro nalezení jazykových inkluzína místo této aproximační metody užití výpočetně úplných algoritmů, které jsou v mnohých případech dokonce rychlejší než simulace. Tím by se metoda proudového odstranění stala vzhledem k výslednému minimalizovanému automatu bezchybnou.

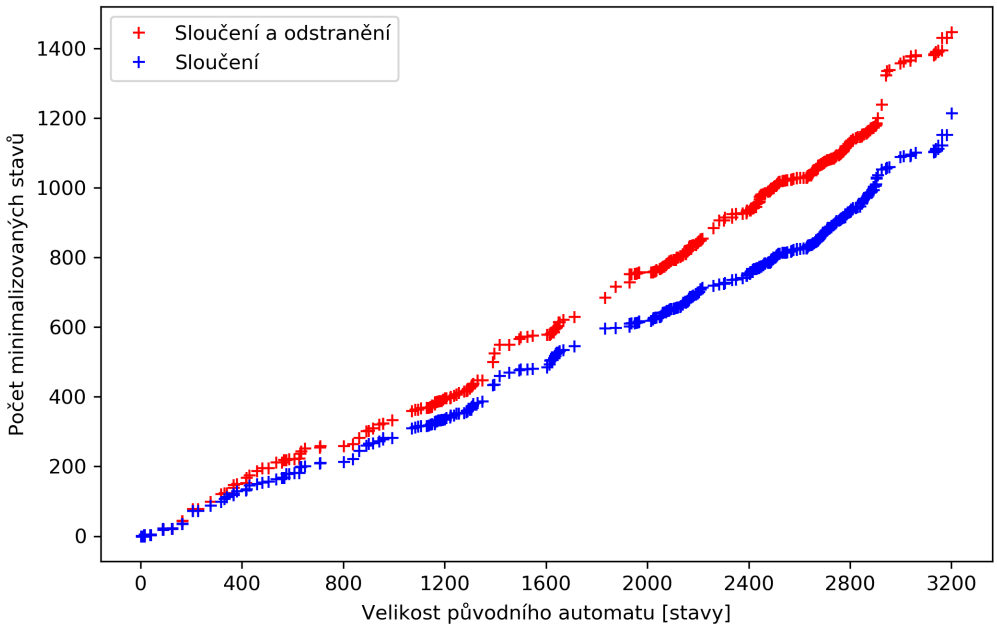
Kapitola 5

Experimentální měření

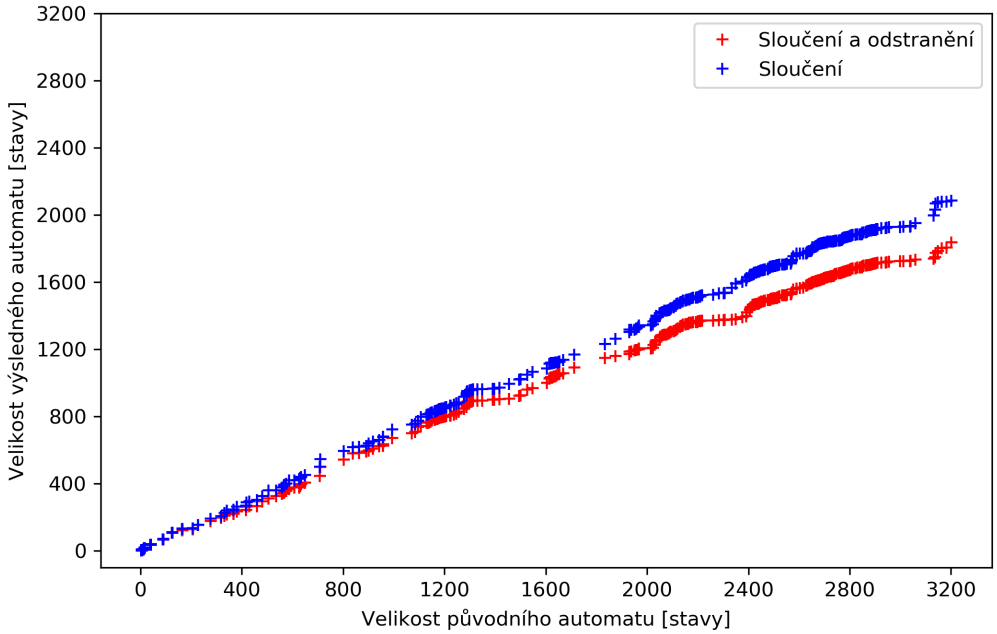
Algoritmy minimalizace byly implementovány v programovacím jazyce Python3 s použitím základních operací nad nedeterministickými automaty, které poskytla knihovna Symboliclib [1]. Testování probíhalo na 780 automatech s celkovým počtem 1 667 198 stavů a 7 621 222 hran přechodů. Hlavními kritérii hodnocení byla výsledná velikost minimalizovaného automatu a počet provedených minimalizací.

5.1 Vliv na počet stavů

Při použití metody minimalizující automat sloučením ekvivalentních stavů obsahoval výsledný automat 68% původního množství stavů. V kombinaci s metodou odstranění bylo ve výsledném automatu 61% původního množství stavů. Toto zlepšení představuje 21,8% nárůst efektivity minimalizace.



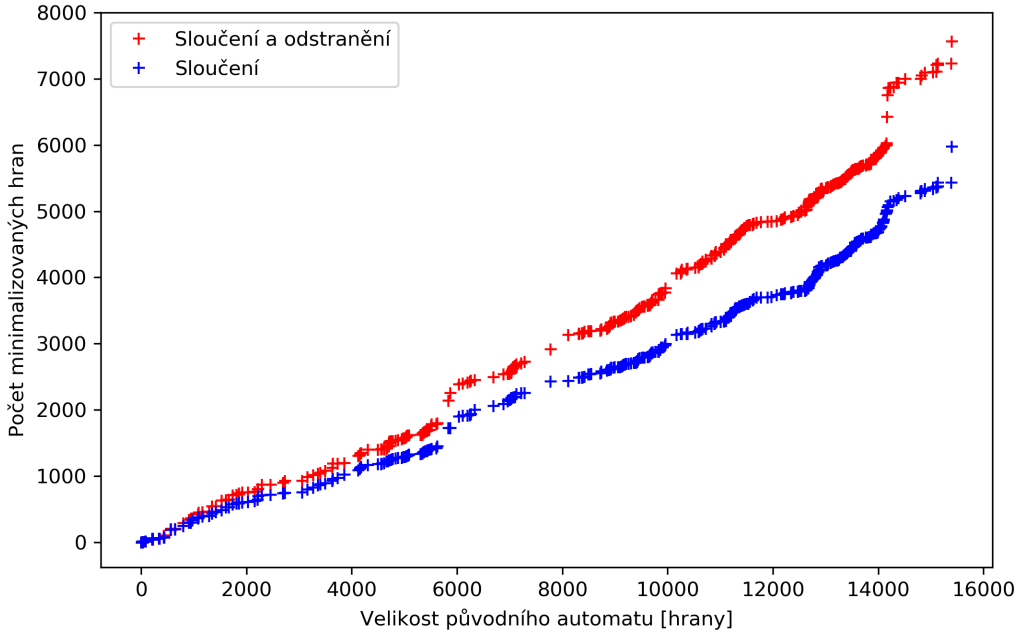
Obrázek 5.1: zvýšení efektivity (počet stavů) při použití metody odstranění



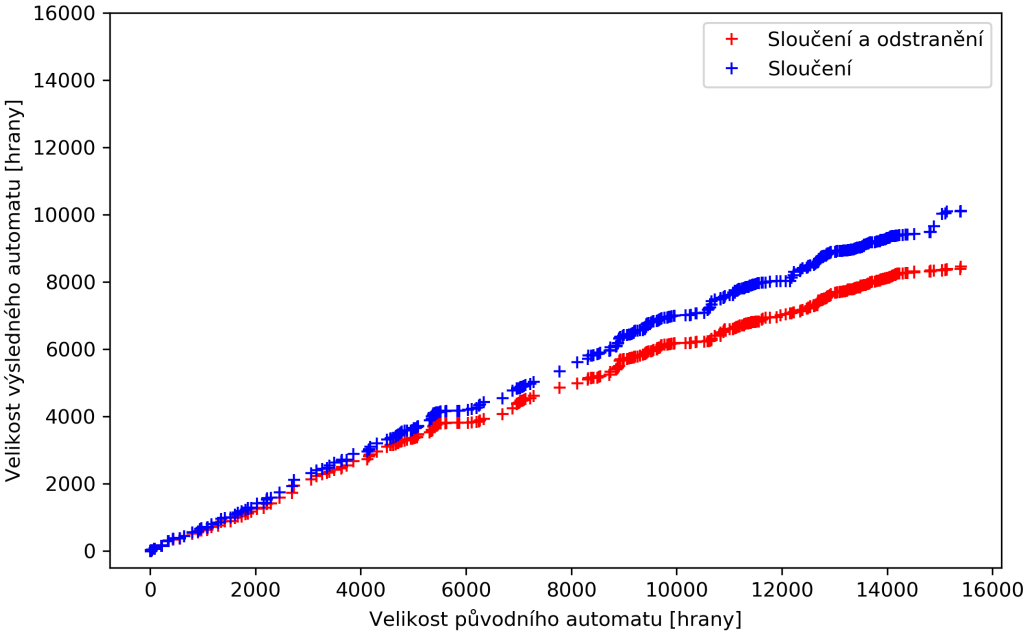
Obrázek 5.2: poměr stavů počátečního a výsledného automatu

5.2 Vliv na počet hran přechodů

Při použití metody minimalizující automat sloučením ekvivalentních stavů obsahoval výsledný automat 68,7% původního množství hran přechodů. V kombinaci s metodou odstranění bylo množství hran ve výsledném automatu sníženo až na 60% původního počtu. Tento rozdíl představuje co do velikosti až 27,5% nárůst efektivity při minimalizaci počtu hran.



Obrázek 5.3: zvýšení efektivity (počet hran) při použití metody odstranění



Obrázek 5.4: poměr hran přechodů počátečního a výsledného automatu

Kapitola 6

Závěr

Tato práce nalezla podmínky umožňující minimalizaci nedeterministického konečného automatu sloučením stavů, a to i v případě, že je dvojice minimalizovaných stavů pouze v oboustranné jazykové inkluzi.

Dalším hlavním přínosem práce bylo vytvoření algoritmů a specifikace podmínek, které umožňují minimalizaci NKA odstraněním slabšího stavu z minimalizované dvojice stavů. Byly také definovány podmínky pro odstranění stavů v oboustranné jazykové inkluzi i ekvivalenci. Práce dále určuje podmínky umožňující provedení minimalizace odstraněním při existenci cest mezi minimalizovanými stavy. V kapitole 4 byl uveden algoritmus metody odstranění, který pracuje na základě podmínek specifikovaných v předešlých kapitolách. Jeho užití při minimalizaci v kombinaci se známou metodou slučování stavů vedlo ke zvýšení efektivity, a to až o dvacet sedm procent.

Literatura

[1] Bielikova, M. *Symboliclib library for finite and symbolic automata and transucers.* 2017. Dostupné z: https://github.com/Miskaaa/symboliclib.

[2] Holík, L., Vojnar, T., Abdulla, P., Chen, Y.-F. a Mayr, R. When Simulation Meets Antichains (On Checking Language Inclusion of Nondeterministic Finite (Tree) Automata). In: *Tools and Algorithms for the Construction and Analysis of Systems* [print]. LNCS 6015. Springer Verlag: Springer Verlag, March 2010, kap. 34731, s. 158–174.

[3] Ilie, L., Navarro, G. a Yu, S. On NFA Reductions. *Theory Is Forever Lecture Notes in Computer Science.* Springer, Berlin, Heidelberg. 2004, s. 112–124.

[4] Meduna, A. a Lukáš, R. *Formální jazyky a překladače: Abecedy, řetězce a znaky.* Božetěchova 1/2, 612 00 Brno-Královo Pole: FIT VUT v Brně, 2017.

[5] Meduna, A. a Lukáš, R. *Formální jazyky a překladače: Modely pro regulární jazyky.* Božetěchova 1/2, 612 00 Brno-Královo Pole: FIT VUT v Brně, 2017.

[6] Meduna, A. a Lukáš, R. *Formální jazyky a překladače: Speciální typy konečných automatů.* Božetěchova 1/2, 612 00 Brno-Královo Pole: FIT VUT v Brně, 2017.

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

[8] Šedý, M. *Přežití silnějšího v minimalizaci nedeterministických automatů: Projektová praxe.* Brno: Vysoké učení technické v Brně, Fakulta informačních technolofgií, 2019. Vedoucí práce Mgr. Lukáš Holík Ph.D.

[9] Šedý, M. *Přežití silnějšího v minimalizaci nedeterministických automatů: Projektová praxe.* Brno: Vysoké učení technické v Brně, Fakulta informačních technolofgií, 2019, s. 3–5. Vedoucí práce Mgr. Lukáš Holík Ph.D.

[10] Šedý, M. *Přežití silnějšího v minimalizaci nedeterministických automatů: Projektová praxe.* Brno: Vysoké učení technické v Brně, Fakulta informačních technolofgií, 2019, s. 7–8. Vedoucí práce Mgr. Lukáš Holík Ph.D.