Regulární jazyky, konečné automaty a konečné pologrupy

Martin Masař

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin Masař**
Osobní číslo:	**A20313**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Regulární jazyky, konečné automaty a konečné pologrupy**
Téma práce anglicky:	**Regular Languages, Finite Automata, and Finite Semigroups**

Zásady pro vypracování

1. Nastudujte a popište různé způsoby zadání regulárního jazyka – pomocí regulárního výrazu, konečného automatu a konečné pologrupy – a související teorii.
2. Vysvětlete souvislosti mezi jednotlivými způsoby zadání regulárního jazyka.
3. Vybrané souvislosti popište podrobněji a ilustrujte na příkladech.
4. Popište, jak se v jednotlivých přístupech realizují vybrané základní operace s regulárními jazyky.
5. Seznamte se s dostupnými softwarovými systémy vhodnými pro ilustraci tohoto tématu a popište, jak je lze konkrétně využít. Demonstrujte na příkladech.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ČERNÁ, Ivana, Mojmír KŘETÍNSKÝ a Antonín KUČERA. Formální jazyky a automaty I [online]. 1. vyd. Brno: Masarykova univerzita, 2006. Elportál. Dostupné z: http://is.muni.cz/elportal/?id=703389. ISSN 1802-128X.
2. JANČAR, Petr, Martin KOT a Zdeněk SAWA. Úvod do teoretické informatiky – učební text [online]. 1. vyd. Ostrava: Ediční středisko VŠB-TUO, 2007. Dostupné z: http://www.cs.vsb.cz/kot/down/uti2007/UTI-text.pdf.
3. VAVREČKOVÁ, Šárka. Teorie jazyků a automatů II, Základy teoretické informatiky II [online]. Opava: Slezská univerzita v Opavě, 2015. Dostupné z: http://vavreckova.zam.slu.cz/obsahy/tja2/skripta/tja2.pdf.
4. PIN, Jean-Éric. Mathematical Foundations of Automata Theory [online]. Version of February 18, 2022. Institut de Recherche en Informatique Fondamentale, 2022. Dostupné z: https://www.irif.fr/ jep/PDF/MPRI/MPRI.pdf.
5. ROZENBERG, Grzegorz a Arto SALOMAA, ed. Handbook of formal languages: Volume 1 Word, Language, Grammar. Berlin: Springer, 1997. ISBN 978-3-642-63863-3.
6. VÁVROVÁ, Pavlína. Operace s regulárními jazyky pomocí rozpoznávání monoidy. Brno, 2012. Dostupné také z: https://is.muni.cz/th/psp6g/. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Michal Kunc.
7. VANÍČEK, Jiří et al. 2007. Teoretické základy informatiky. Praha: Kernberg. Informatika studium. ISBN 978-80-903962-4-1.

Vedoucí bakalářské práce: **Mgr. Jana Volaříková**
Ústav matematiky

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

L.S.

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 25.5.2023 …….

podpis studenta

ABSTRAKT

Tato bakalářská práce se zabývá studiem a popisem různých způsobů zadání regulárního jazyka a související teorie. Teoretická část obsahuje popis regulárních jazyků, následně jsou popsány jednotlivé způsoby zadání regulárních jazyků, pomocí regulárního výrazu, konečného automatu a monoidu a jejich souvislosti a operace, které mohou provádět. Vybrané způsoby a operace jsou ilustrovány na příkladech, které jsou v praktické části práce implementovány pomocí jazyka Wolfram Mathematica a Pythonu.

Klíčová slova: regulární jazyk, regulární výraz, konečný automat, monoid, Wolfram Mathematica, Python

ABSTRACT

This bachelor's thesis deals with the study and description of various methods for specifying a regular language and related theory. The theoretical part includes a description of regular languages, followed by an explanation of the individual ways to specify regular languages and their connections, as well as operations that can be performed using them. Selected methods and operations are illustrated on examples, which are implemented in the practical part of the thesis using Wolfram Mathematica and Python.

Keywords: regular language, regular expression, finite automaton, monoid, Wolfram Mathematica, Python

Tímto bych chtěl srdečně poděkovat mé vedoucí bakalářské práce paní Mgr. Janě Volaříkové za cenné rady a nesmírnou ochotu při konzultacích. Další poděkování patří mé rodině a blízkým přátelům, kteří mě po celou dobu studia neustále podporovali.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Regulární jazyky mají v dnešní době široké uplatnění v různých oblastech, kde je potřeba pracovat s textovými daty a provádět analýzy na základě určitých vzorců. Například regulární výrazy jsou neodmyslitelnou součástí programování a textové manipulace. Jsou využívány při vyhledávání, zpracování a manipulaci s textovými řetězci v různých programovacích jazycích jako JavaScript, Python nebo Java. Například při vývoji webových aplikací lze pomocí regulárních výrazů ověřovat formáty vstupních údajů v HTML formulářích, jako jsou e-mailové adresy nebo telefonní čísla. Dále se používají při routování a analýze URL adres a při zpracování textových dat v rámci serverové strany aplikace.

Regulární jazyky ovšem nejsou omezeny jen na práci s textovými daty a provádění analýzy na základě určitých vzorců. Využívají se také v oblasti modelování a simulace různých systémů. Konečné automaty, které jsou spojeny s regulárními jazyky, mohou být použity k popisu a analýze chování systémů v různých aplikacích, včetně softwarového inženýrství, telekomunikací nebo návrhu hardwaru.

Tato bakalářská práce se v teoretické části zaměřuje na popis různých způsobů zadání regulárního jazyka, konkrétně pomocí regulárního výrazu, konečného automatu a monoidu a jejich souvislostí. Dále je v teoretické části popsána implementace vybraných základních operací s regulárními jazyky v jednotlivých přístupech. Vybrané souvislosti jsou podrobněji ilustrovány na příkladech, které jsou následně v praktické části zpracovány a popsány v jazyce Wolfram Mathematica a Python.

I. TEORETICKÁ ČÁST

1 FORMÁLNÍ JAZYKY

Pro popis formálních jazyků je nutné definovat následující pojmy.

Definice:

Abeceda Σ je libovolná konečná neprázdná množina symbolů. [1]

Příkladem abecedy může být třeba množina {0,1}, která se používá pro tvorbu binárních řetězců, nebo množina znaků latinské abecedy. Z jednotlivých symbolů abecedy se skládají řetězce. [2]

Definice:

Řetězec nad danou abecedou Σ je libovolná konečná posloupnost symbolů abecedy. Prázdný řetězec ε je posloupnost, která neobsahuje žádný symbol. [2]

Definice:

Jazykem L nad abecedou Σ nazýváme libovolnou podmnožinu množiny Σ*

Formální jazyk L je: [3]

- prázdný, jestliže L = ∅,
- konečný, pokud obsahuje konečný počet slov,
- nekonečný, jestliže obsahuje nekonečný počet slov

1.1 Gramatika

Je způsob popisu formálního jazyka pomocí pravidel, která určují, jaké symboly se mohou vyskytovat v řetězci a jak se mohou tyto symboly kombinovat. [4]

Gramatika je uspořádaná čtveřice G = (N, Σ, P, S), kde [4]

- N je abeceda neterminálních symbolů
- Σ je konečná množina terminálních symbolů, splňující podmínku N ∩ Σ = ∅
- P je konečná množina přepisovacích pravidel, kde jsou pravidla(α, β) obvykle zapisována ve tvaru α → β, α musí obsahovat alespoň jeden neterminální symbol, β je libovolný řetězec terminálních a neterminálních symbolů a nejsou na něj kladena žádná pravidla, tudíž může být reprezentován i prázdným slovem.
- S ∈ N je počáteční neterminální symbol

Jazyk L lze z gramatiky G získat pomocí přepisovacích pravidel, kde se začíná počátečním symbolem gramatiky a opakovaně používají přepisovací pravidla. [4]

Například jazyk L(G) generovaný gramatikou G: N = {S, A, B} Σ = {a, b} P = {S → AB | ASB, A → a, B → b} bude vypadat následovně L(G) = {$a^n b^n$, n>=1}

Nejznámější klasifikací gramatik je Chomského hierarchie, která dělí gramatiky do 4 úrovní na základě tvaru jejich přepisovacích pravidel.

Typ 0 – Neomezená gramatika

Obsahuje pravidla v nejobecnějším tvaru shodným s definicí samotné gramatiky. [5, 6]

Typ 1 – Kontextová gramatika

má pravidla ve tvaru α A β → α γ β kde A je neterminální symbol a α, γ, β jsou řetězce terminálních a neterminálních symbolů, přičemž γ musí být neprázdný řetězec. [5, 6]

Typ 2 – Bezkontextová gramatika

má pravidla ve tvaru A → β kde A je neterminální symbol a β je řetězec terminálních a neterminálních symbolů. [5, 6]

Typ 3 – Regulární gramatika

Obsahuje gramatiky s nejpřísnějšími pravidly. Je zapsána ve tvaru A → a, nebo A → aB, kde A je neterminální symbol, a je terminální symbol, který může být následován B, které je neterminální symbol. [5, 6]

1.2 Regulární jazyky

Představují nejmenší třídu formálních jazyků podle Chomského hierarchie, která může reprezentovat i jazyky. Mohou být popsané například pomocí konečného automatu, regulárního výrazu nebo gramatiky typu 3. Používají se například při analýze a navrhování programovacích jazyků nebo rozpoznávání vzorů v datech. [7]

Regulární jazyk nad abecedou Σ je definován následovně: [8]

- Ø a {ε} je regulární jazyk.
- Když a ∈ Σ, pak {a} je regulární jazyk.
- Jsou-li A a B regulární jazyky, pak A* (iterace), A ∪ B (sjednocení), A • B (zřetězení) jsou také regulární jazyky.
- Žádné jiné jazyky nad Σ nejsou regulární.

Iterace

Značí se symbolem *, a používá se k vytvoření nového jazyka, který odpovídá libovolnému počtu opakování původního regulárního jazyka, včetně nulového počtu. Pokud je jazyk L regulární, tak i jazyk L* bude regulárním jazykem. Formální zápis iterace zní: $L^* = \{w_1w_2\ldots. w_n \mid n \geq 0$ a $w_i \in L$ pro každé $i = 1, 2,\ldots n\}$. [9]

Sjednocení

Je další z operací regulárních jazyků. Značí se symbolem ∪ a slouží k vytvoření nového jazyka, který obsahuje všechna slova z obou původních jazyků. Pokud jsou jazyky L_1 a L_2 regulární, tak i jazyk $L = L_1 \cup L_2$ bude regulárním jazykem. Operace sjednocení je asociativní a komutativní. Formální zápis sjednocení zní: $L_1 \cup L_2 = \{w \mid w \in L_1$ nebo $w \in L_2\}$. [10, 11]

Zřetězení

Operace zřetězení se značí symbolem • a slouží k vytvoření nového jazyka obsahujícího zřetězená slova z obou původních jazyků. Pokud jsou jazyky L_1 a L_2 regulární, tak i jazyk $L = L_1 \bullet L_2$ bude regulárním jazykem. Na rozdíl od operace sjednocení je operace zřetězení pouze asociativní, ale už není komutativní. Formální zápis sjednocení zní: $L_1 \bullet L_2 = \{xy \mid x \in L_1, y \in L_2\}$. [10, 11]

1.3 Regulární výrazy

Regulární výrazy jsou způsobem zápisu regulárních jazyků. Skládají se z jednoho nebo více znaků. Komplexní regulární výrazy se tvoří pomocí použití symbolů *, +, .. Toto umožňuje zápis jazyků, jako jsou například všechny řetězce obsahující určitý podřetězec, všechny řetězce začínající určitým prefixem a další. Jednotlivé symboly mají různé priority. Nejvyšší prioritu má iterace, následuje zřetězení a jako poslední je sjednocení. Regulární výraz můžeme použít k popisu regulárních jazyků. [1]

Regulární výraz nad abecedou Σ je definován následovně: [4, 12]

- Ø, {ε}, a, kde $a \in \Sigma$ jsou regulární výrazy nad abecedou Σ
- Jsou-li A a B regulární výrazy, pak A*, A + B, AB jsou také regulární výrazy.
- Žádné jiné výrazy nad Σ nejsou regulární.

Jestliže R je regulární výraz, tak L(R) označuje jazyk popsaný regulárním výrazem R. Jazyk L(R) je definován následujícími pravidly: [1]

1. Ø je regulární výraz, který označuje prázdnou množinu,
2. ε je regulární výraz, který označuje {ε}.
3. Pro každé a ∈ Σ je a regulární výraz, který označuje {a}.

Pokud R_1 a R_2 jsou regulární výrazy, pak můžeme použít pravidla této definice sloužící ke snižování L(R) na jednodušší komponenty: [1]

- $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
- $L(R_1R_2) = L(R_1) \cup L(R_2)$
- $L((R_1)) = L(R_1)$
- $L(R_1^*) = (L(R_1))^*$

V programování se často využívá rozšířená verze regulárních výrazů označovaná jako regex. Zde mohou regulární výrazy obsahovat také speciální znaky, které zastávají určité funkce, jako je například \d pro jakoukoliv číslici, \n nový řádek, \w pro jakýkoli znak slova a mnoho dalších. Tyto regulární výrazy mohou být použity v různých programovacích jazycích a nástrojích jako například grep a sed, dále mohou být použity k ověření vstupu uživatele, vyhledávání a nahrazování textu v souborech a extrakci dat z textu. Některé z těchto úprav pouze mění zápis regulárního výrazu a dají se přepsat v základním tvaru, například \d nebo \w je možné zapsat pomocí sady operátorů sjednocení. Aserce jako například ^, $ nebo (?=...) na regulární výraz převést nelze. [13, 14, 15]

1.4 Konečné automaty

Konečné automaty, také známé jako konečné stavové automaty, jsou matematické modely používané k popisu chování systémů, které se mohou nacházet v konečném počtu stavů. Hrají významnou roli v informatice, zejména v oblasti zpracování textu a návrhu hardwaru. [12, 16]

Konečný automat je matematický model reprezentující stroj, který může číst a manipulovat s řetězci symbolů. Automat zpracovává zadaný vstupní řetězec symbol po symbolu a přechází ze stavu do stavu podle své přechodové funkce. Pokud automat po zpracování celého vstupního řetězce dosáhne koncového stavu, pak je řetězec automatem přijat, jinak je řetězec odmítnut. Jazyk L můžeme reprezentovat konečným automatem tak, že množina akceptovaných slov automatu je rovna jazyku L. Takovýto jazyk je regulární jazyk. [1]

Konečné automaty a regulární výrazy jsou úzce propojeny a jejich propojení spočívá v tom že, regulární výrazy jsou řetězce, které popisují regulární jazyky (množiny řetězců) a konečné automaty jsou matematické modely přijímající řetězce definované regulárním jazykem. Tyto dva koncepty jsou navzájem ekvivalentní, což znamená, že každý regulární výraz může být převeden na ekvivalentní konečný automat a každý konečný automat může být převeden na ekvivalentní regulární výraz. Můžeme tedy říci, že konečný automat lze použít k rozpoznávání, zda daný řetězec patří do daného jazyka, který je definován regulárním výrazem. [1]

Konečný automat je uspořádaná pětice A = (Q, Σ, δ, q_, F), kde [17]

Q . . . neprázdná konečná množina stavů

Σ . . . neprázdná konečná abeceda (množina signálů)

δ . . . parciální přechodová funkce, definovaná níže

q_ . . . počáteční stav, q_ ∈ Q

F . . . množina koncových stavů, F ⊆ Q, F ≠ ∅

Parciální přechodová funkce δ konečného automatu A = (Q, Σ, δ, q_, F) je definována takto: [1]

δ: Q × Σ → Q pro deterministický konečný automat

δ: Q × Σ → 2Q pro nedeterministický konečný automat

Parciální přechodová funkce δ‘Q × Σ* → Q konečného automatu A = (Q, Σ, δ, q_, F) je definována induktivně vzhledem k délce slova Σ*: [1]

δ‘(q,ε) = q pro každý stav q∈Q

δ‘(q, ωa) =\{ δ(δ‘(q, ω), a) je-li δ‘(q, ω) a δ(δ‘(q, ω), a), jinak funkce není definována\}

Existují dva hlavní typy konečných automatů.

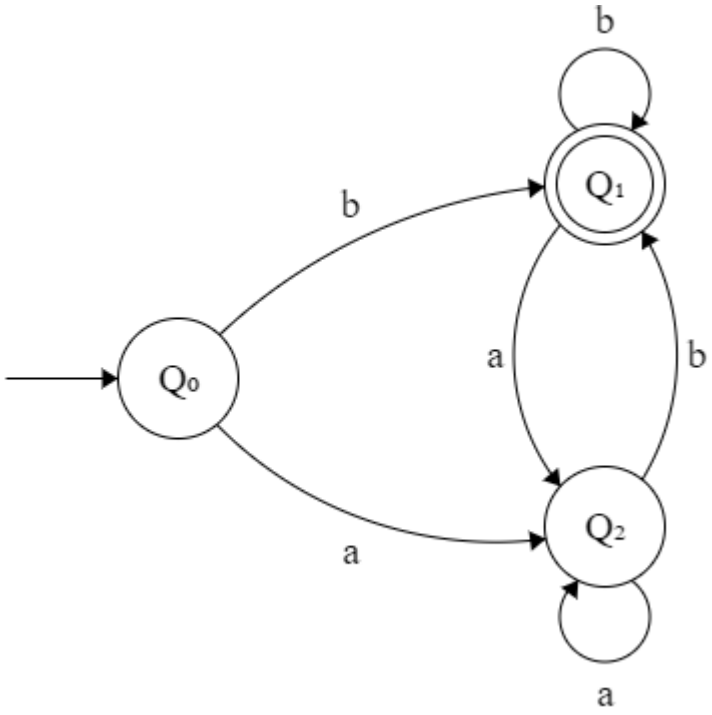
1.4.1 Deterministické konečné automaty

V deterministickém konečném automatu (dále DKA) má každý stav jedinečný přechod pro každý vstupní symbol a tímto je zajištěno jeho deterministické chování. [12]

Přijímání řetězců v DKA probíhá tak, že automat začíná v počátečním stavu a na základě čtení vstupních symbolů postupuje pomocí jednotlivých přechodů. Pokud automat dočte

vstupní řetězec a skončí v koncovém stavu, znamená to, že přijal vstupní řetězec jako součást jazyka, v opačném případě, pokud automat skončí ve stavu, který není koncovým stavem, nebo není schopen dočíst vstupní řetězec (například kvůli neexistujícímu přechodu) znamená to, že řetězec není součástí jazyka. [1]

DKA může být reprezentován pomocí stavového diagramu, který ukazuje všechny stavy, vstupní symboly a přechodové funkce. Stavový diagram umožňuje snadnou vizualizaci funkce automatu a pomáhá při analýze vlastností jazyka, který je reprezentován automatem. V zobrazeném diagramu níže je počáteční stav Q_0 označen šipkou a konečný stav Q_1 je značen dvojitým kruhem. [1]



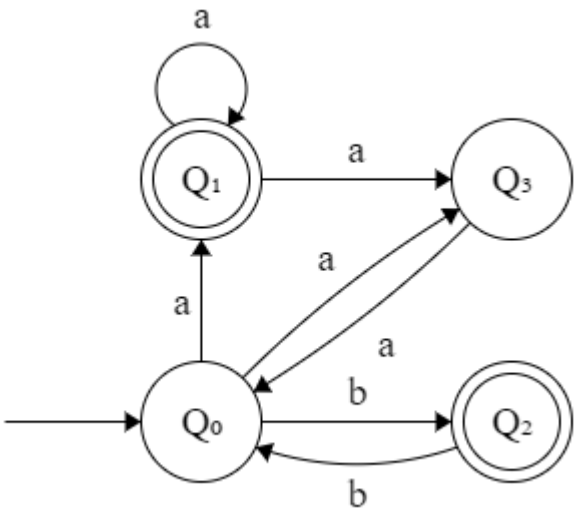
Obrázek 1. DKA A_1

1.4.2 Nedeterministické konečné automaty

V nedeterministickém konečném automatu (dále NKA) může mít stav více přechodů pro stejný vstupní symbol, což vede k více možným cestám a jeho nedeterministickému chování. [12]

Proces přijímání řetězců v NKA probíhá tak, že automat začíná v počátečním stavu a na základě čtení vstupních symbolů postupuje pomocí jednotlivých přechodů. Automat může v každém kroku zvolit více možností přechodu, což může vést k několika možným výsledkům. Pokud automat skončí v koncovém stavu v jedné ze svých možných větví, znamená to, že

přijal vstupní řetězec jako součást jazyka, v opačném případě, pokud automat nekončí v žádném koncovém stavu v žádné ze svých možných větví, znamená to, že řetězec není součástí jazyka. Stejně jako u diagramu DKA i zde je opět počáteční stav Q_0 označen šipkou a konečné stavy Q_1, Q_2 jsou značeny dvojitým kruhem. [1]



Obrázek 2. NKA A_2

Hlavní rozdíl mezi NKA a DKA spočívá v tom, že NKA může mít více možností přechodu ze stavu do stavu pro jeden vstupní symbol. To znamená, že v každém kroku může automat zvolit různé větve, aby prošel vstupní řetězec. Dalším rozdílem je zpětné ověření, což je možnost zpětně trasovat průchod slova automatem, které je na rozdíl od NKA u DKA možné díky jeho deterministickému chování. I přes tyto rozdíly NKA i DKA rozpoznávají stejnou množinu jazyků, kterou je právě množina regulárních jazyků. [18]

1.4.3 Úpravy konečných automatů

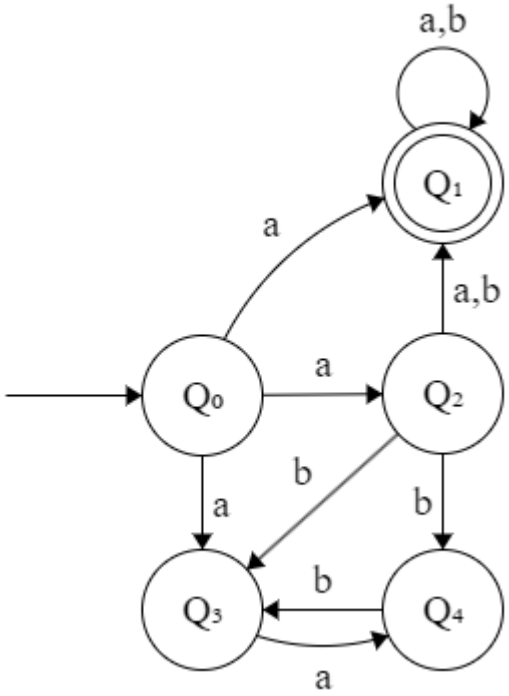
1.4.3.1 Převod NKA na DKA

Jedna z možností, jak převést NKA na DKA, je pomocí tabulky přechodů NKA, kde řádky znázorňují stavy, sloupce reprezentují vstupní symboly a buňky reprezentují následující stavy. Následně určíme počáteční stav DKA, který odpovídá množině počátečních stavů NKA. Po určení počátečního stavu vytvoříme tabulku přechodů pro DKA, kde oproti tabulce NKA řádky reprezentují množiny stavů. Pro každý vstupní symbol obsahuje odpovídající buňka v přechodové tabulce množinu stavů získaných dodržováním přechodových pravidel v přechodové tabulce NKA. Veškeré stavy v DKA, které obsahují alespoň jeden konečný

stav z NKA, jsou konečné stavy. Nakonec pomocí sestrojené tabulky přechodů DKA sestrojíme ze stavů a přechodů výsledný DKA. [19]

Příklad 1.1:

Máme NKA N = (Q, Σ, δ, q_, F), kde Q = {Q_0, Q_1, Q_2, Q_3, Q_4}, Σ = {a, b}, q_ = Q_0, F = {Q_1}, přechody jsou znázorněny na obrázku.



Obrázek 3. NKA N

Vytvoříme tabulku přechodů NKA

NKA	a	b
→Q_0	{Q_1, Q_2, Q_3}	{}
←Q_1	Q_1	Q_1
Q_2	Q_1	{Q_1, Q_3, Q_4}
Q_3	Q_4	{}
Q_4	{}	Q_3

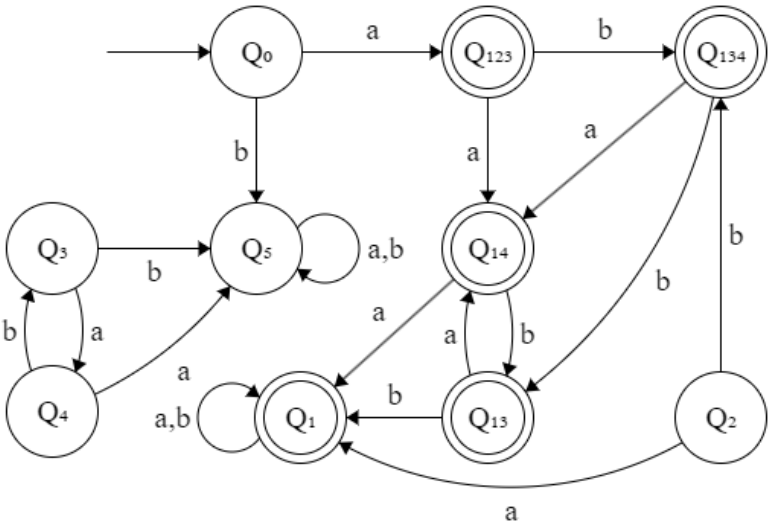
Tabulka 1. Tabulka přechodů NKA N

Dále vytvoříme tabulku pro DKA

DKA	a	b
→Q_0	Q_{123}	Q_5
←Q_{123}	Q_{14}	Q_{134}
←Q_{14}	Q_1	Q_{13}
←Q_{134}	Q_{14}	Q_{13}
←Q_{13}	Q_{14}	Q_1
←Q_1	Q_1	Q_1
Q_2	Q_1	Q_{134}
Q_3	Q_4	Q_5
Q_4	Q_5	Q_3
Q_5	Q_5	Q_5

Tabulka 2. Tabulka přechodů DKA D

Následně určíme počáteční a koncové stavy DKA podle toho, jestli obsahují počáteční nebo koncové stavy DKA. Nakonec pomocí této tabulky sestrojíme výsledný DKA, který vypadá následovně: D = (Q, Σ, δ, q_, F), kde Q = {Q_0, Q_1, Q_2, Q_3, Q_4, Q_5, Q_{13}, Q_{14}, Q_{123}, Q_{134}}, Σ = {a, b}, q_ = Q_0, F = {Q_1, Q_{13}, Q_{14}, Q_{123}, Q_{134}}, přechody jsou znázorněny na obrázku.



Obrázek 4. DKA D

1.4.3.2 Minimalizace konečného automatu

Minimalizace konečných automatů slouží k získání minimálního automatu, aniž by se změnil jazyk rozpoznávaný tímto automatem. To znamená, že minimalizovaný automat bude ekvivalentní s původním automatem, ale bude obsahovat méně stavů. Pro minimalizaci DKA existuje algoritmus, který je členěn na 2 fáze. První je vyhledání a odstranění nedosažitelných a mrtvých stavů. Do nedosažitelného stavu se konečný automat nikdy nedostane při přechodu z jednoho stavu do druhého a mrtvý stav je nepřijímající stav, který má přechody pouze sám do sebe. [20]

Další částí je algoritmus, který spočívá v rozdělení stavů do tříd ekvivalence. Nejdříve rozdělíme stavy na 2 skupiny. Skupinu koncových a skupinu nekoncových stavů. Následně kontrolujeme tyto skupiny a dělíme je dále podle vstupních symbolů a zařazení do skupiny. Tento proces se opakuje pro každou skupinu tak dlouho**,** dokud je skupiny možné dále dělit. [20]

Příklad 1.2 :

V předešlém příkladu jsme vytvořili DKA D, který zde minimalizujeme.

První odstraníme nedosažitelné stavy, kterým je v tomto případě stav Q_2.

Následně si opět vytvoříme tabulku přechodů DKA. Zde můžeme použít již vytvořenou tabulku 2.

Po vytvoření tabulky přechodů začneme aplikovat algoritmus ekvivalence

0. ekvivalence $\{Q_0, Q_3, Q_4, Q_5\}$, $\{Q_1, Q_{13}, Q_{14}, Q_{123}, Q_{134}\}$ rozdělíme na koncové a nekoncové stavy

1. ekvivalence $\{Q_0\}$, $\{Q_3, Q_4, Q_5\}$, $\{Q_1, Q_{13}, Q_{14}, Q_{123}, Q_{134}\}$

2. ekvivalence $\{Q_0\}$, $\{Q_3, Q_4, Q_5\}$, $\{Q_1, Q_{13}, Q_{14}, Q_{123}, Q_{134}\}$

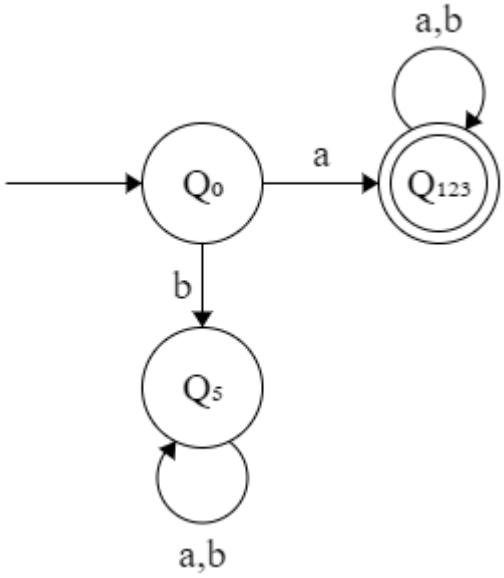
Zde můžeme vidět že se 1. a 2. ekvivalence nezměnily, a proto dále dělit není možné.

Poté opět vytvoříme tabulku přechodů tentokrát pro minimalizovaný DKA.

DKA	a	b
$\rightarrow Q_0$	Q_{123}	Q_5
$\{Q_3, Q_4, Q_5\}$	$\{Q_3, Q_4, Q_5\}$	$\{Q_3, Q_4, Q_5\}$
$\leftarrow\{Q_1, Q_{13}, Q_{14}, Q_{123}, Q_{134}\}$	$\{Q_1, Q_{13}, Q_{14}, Q_{123}, Q_{134}\}$	$\{Q_1, Q_{13}, Q_{14}, Q_{123}, Q_{134}\}$

Tabulka 3. Tabulka přechodů minimalizovaného DKA D

Nakonec podle tabulky přechodů vytvoříme minimalizovaný DKA. Zde je také možnost odstranění mrtvých stavů, v tomto případě $\{Q_3, Q_4, Q_5\}$. Tento krok zde ale není nutný a nově vytvořený automat je v tomto případě dostatečně přehledný i bez této úpravy. Navíc po této úpravě dostaneme pouze částečný DKA, jelikož stavu Q_0 bude chybět přechod pomocí znaku b.



Obrázek 5. Minimalizovaný DKA D

1.5 Konečné pologrupy

Pologrupy jsou důležitým nástrojem v matematice, zejména v teorii čísel a v teorii automatů. Vyjadřují množiny uzavřené na danou binární operaci a musí zde platit asociativní zákon, tedy pro libovolné prvky x, y, z platí (x * y) * z = x * (y * z). Konečné pologrupy jsou pologrupy, které mají konečný počet prvků. V praxi mají konečné pologrupy využití například v kryptografii, kde se používají jako součást algoritmů pro šifrování a dešifrování zpráv. [21]

1.5.1 Monoidy

Speciálním druhem pologrupy je monoid. Všechny monoidy jsou pologrupy, ale ne všechny pologrupy jsou monoidy. Je to algebraická struktura, která má oproti pologrupě navíc jednotkový prvek. Tento prvek může být také označován jako neutrální prvek a značí se obvykle jako e. Jednotkový prvek má vlastnost, že x * e = e * x = x pro všechny prvky x v pologrupě. [21]

1.5.2 Homomorfismus monoidů

Homomorfismus monoidů je zobrazení mezi dvěma monoidy, které zachovává operace mezi prvky těchto struktur. Konkrétněji řečeno, nechť (A, *) a (B, ∘) jsou dva monoidy, pak homomorfismus ϕ je zobrazení A → B, pokud pro každé dva prvky x a y z A platí: ϕ (x * y) = ϕ (x) ∘ ϕ (y). [21, 22]

1.5.3 Zadání regulárního jazyka pomocí homomorfismu monoidů

Regulární jazyk L můžeme vyjádřit pomocí homomorfismu monoidu reprezentovaného čtveřicí (M, Σ, ϕ, F), která rozpoznává jazyk L. M je monoid. Σ reprezentuje abecedu. ϕ: Σ* → M je homomorfismus monoidů. Σ* je monoid s binární operací zřetězení řetězců. Podmnožina F z M je množina akceptačních podmínek. [22]

ϕ: Σ* → M je homomorfismus moniodů rozpoznávající jazyk L, pokud existuje množina F, kde L je rovno ϕ^{-1}(F). [22]

2 OPERACE S REGULÁRNÍMI JAZYKY

2.1 Iterace

2.1.1 Iterace pomocí regulárních výrazů

Iterace je jednou ze základních operací aplikovaných na regulární výrazy. Umožňuje nám vytváření regulárních výrazů zahrnujících opakující se sekvence znaků. Například regulární výraz (a)* odpovídá řetězcům s libovolným počtem a včetně nulového počtu. [1]

Formálně lze iteraci regulárního výrazu R značenou R* definovat pomocí následujících pravidel: [1, 10]

- Prázdný řetězec ε patří do jazyka reprezentovaného R*
- Pokud x patří do jazyka reprezentovaného R, pak x také patří do jazyka reprezentovaného R*
- Pokud x a y patří do jazyka reprezentovaného R*, pak xy také patří do jazyka reprezentovaného R*
- Pokud x patří do R*, pak libovolný počet opakování x také patří do R*

2.1.2 Iterace pomocí konečných automatů

V teorii automatů se iterace používá k vytvoření nového automatu, který rozpoznává jazyk obsahující libovolný počet opakování jazyka rozpoznávaného původním automatem. Při použití operace iterace na automat M nám v množině stavů automatu M* přibyde nový počáteční stav q*, který je navíc dalším koncovým stavem. Ze stavu q* se vytvoří epsilon přechod, což je přechod, který umožňuje automatu změnit stav bez přijetí vstupního symbolu, do původního počátečního stavu, a dále pro každý konečný stav v M existuje epsilon přechod z tohoto stavu do počátečního stavu automatu M*, což nám umožňuje rozpoznat libovolné opakování jazyka rozpoznávaného M, včetně nulových opakování. [4, 8]

Formálně lze operaci iterace konečného automatu definovat následujícími pravidly:

Stavy nového automatu jsou stejné jako stavy původního automatu, ale je k nim přidán stav q*.

Vstupní abeceda nového automatu je stejná jako u původního automatu.

Formálně lze přechodovou funkci δ‘ definovat následovně:[23]

$\delta'(q,x) = \{\delta(q,x)\}$ pokud $q \in Q \wedge \delta(q,x) \notin F$ (počítáme uvnitř A)

$= \{\delta(q,x), q\_\}$ pokud $q \in Q \wedge \delta(q,x) \in F$ (možný restart)

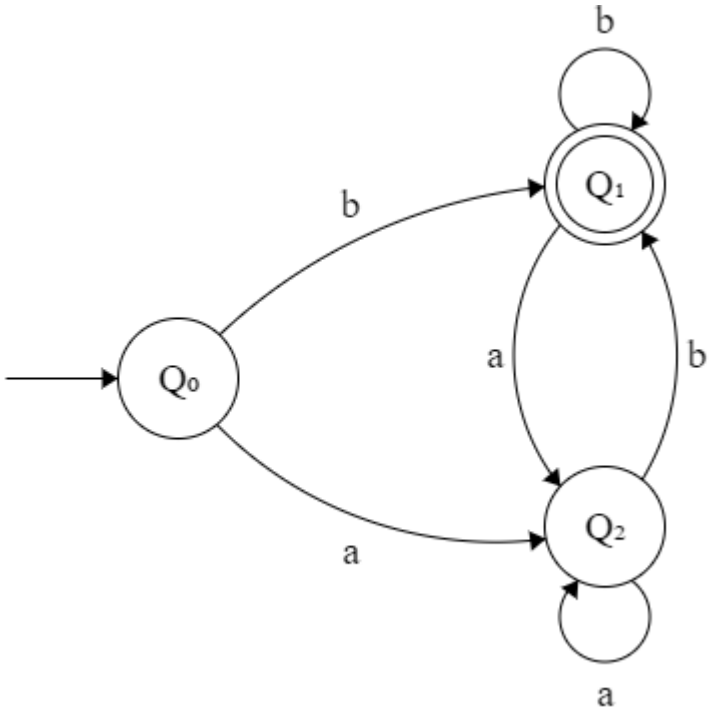
$\delta'(s,x) = \{\}$ žádné přechody z nového stavu

Počáteční stav nového automatu je počáteční stav q*.

Koncové stavy nového automatu jsou stavy, které jsou přijímající stavy původního konečného automatu, a stav q*. [8, 23]

Příklad iterace pomocí DKA 2.1:

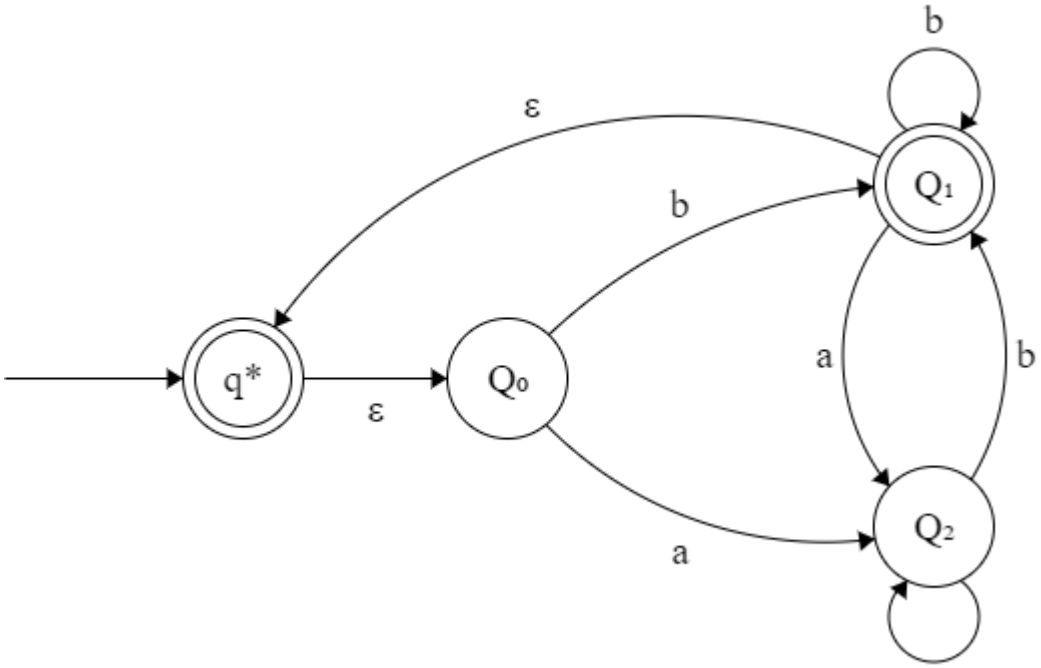
Iteraci konečného automatu $A_1 = (Q_1, \Sigma_1, \delta_1, q\_1, F_1)$, kde $Q_1 = \{Q_0, Q_1, Q_2\}$, $\Sigma_1 = \{a, b\}$, $q\_1 = Q_0$, $F_1 = \{Q_1\}$, přechody jsou znázorněny na obrázku, můžeme provést následujícím způsobem.



Obrázek 1. DKA A_1

Přidáme nový stav q*, který se stane současně počátečním i koncovým stavem. Z q* povede epsilon přechod do stavu Q_0 a ze stavu Q_1 povede epsilon přechod do q*.

Tímto jsme získali konečný automat A = (Q, Σ, δ, q_, F), kde Q = {q*, Q_0, Q_1, Q_2}, Σ = {a, b}, q_ = q*, F = {q*, Q_1}, který je iterací automatu A_1.



Obrázek 6. Iterace pomocí automatu DKA A_1

Iterace pomocí NKA je ilustrována na příkladu 1.1 v příloze 1.

2.2 Sjednocení

2.2.1 Sjednocení pomocí regulárních výrazů

Sjednocení je další základní operací regulárních výrazů, značí se + a umožňuje vytvářet komplexní vzory regulárních výrazů. Pomocí kombinace dvou nebo více regulárních výrazů můžeme vytvořit nový regulární výraz, který odpovídá jakémukoliv řetězci původních výrazů. Například regulární výraz a+b*a odpovídá jakémukoliv řetězci, který začíná na a nebo libovolný počet b a končí na a. [1]

Formálně lze sjednocení dvou regulárních výrazů definovat následovně:

Pokud jsou R_1 a R_2 regulární výrazy, které definují jazyky L_1 a L_2, pak $R_1 + R_2$ je regulární výraz, který definuje jazyk $L_1 \cup L_2$. [24]

2.2.2 Sjednocení pomocí konečných automatů

Je to způsob, jak kombinovat dva nebo více konečných automatů do nového automatu, který rozpoznává sjednocení jazyků rozpoznávaných těmito automaty. Umožňuje vytvářet automaty, které mohou rozpoznávat složitější jazyky. [8]

Formálně lze sjednocení dvou konečných automatů A_1 a A_2 definovat následujícími pravidly: [8, 10]

Stavy nového automatu jsou sjednocením stavů A_1 a A_2 a je přidán jeden nový stav S_0 u kterého musí platit $S_0 \notin Q_1$, $S_0 \notin Q_2$

Vstupní abeceda nového automatu je sjednocením abecedy automatů A_1 a A_2.

Přechodová funkce nového automatu je definována následovně:

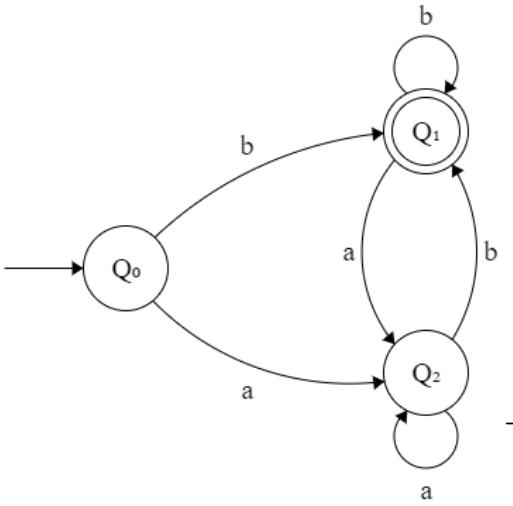
Pro každý stav q v novém automatu a každý vstupní symbol a v abecedě vstupů je další stav sjednocením stavů v A_1 a A_2 pro daný stav a vstupní symbol, $\delta(q, a) = \delta_1(q, a) \cup \delta_2(q, a)$, $\delta(S_0, a) = \delta_1(q\_, a) \cup \delta_2(q\_, a)$.

Počáteční stav nového automatu je nový stav S_0.

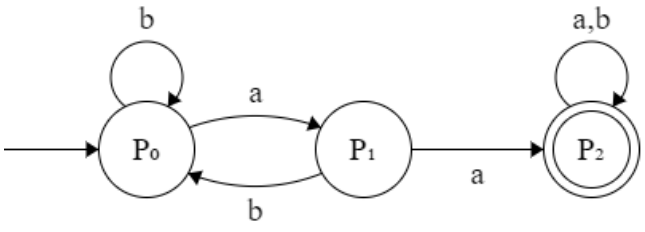
Koncové stavy nového automatu jsou stavy, které jsou koncové buď v A_1, nebo v A_2.

Příklad sjednocení pomocí dvou DKA 2.2:

Konečné automaty A_1 definovaný v příkladu iterace DKA, a automat $A_2 = (P, \Sigma_2, \delta_2, q\_2, F_2)$, kde $P = \{P_0, P_1, P_2\}$, $\Sigma_2 = \{a, b\}$, $q\_2 = P_0$, $F_2 = \{P_2\}$, přechody jsou znázorněny na obrázku níže.



Obrázek 1. DKA A_1



Obrázek 7. DKA A_2

Sjednocení těchto automatů můžeme provést následujícím způsobem. Zkontrolujeme, zdali platí podmínka $Q \cap P = \emptyset$. Pokud podmínka neplatí, je nutné stavy jednoho z automatů přejmenovat, aby podmínka platila.

Následně si sestavíme tabulky přechodů pro automaty A_1 a A_2, kde v prvním sloupci jsou stavy konečného automatu. V dalších sloupcích jsou stavy, do kterých vede přechod ze stavu v prvním sloupci. Po sestavení tabulek určíme počáteční stav pomocí → a koncový stav pomocí ←.

A_1	a	b
$\rightarrow Q_0$	Q_2	Q_1
$\leftarrow Q_1$	Q_2	Q_1
Q_2	Q_2	Q_1

Tabulka 4. Přechody automatu A_1

A_2	a	b
$\rightarrow P_0$	P_1	P_0
P_1	P_2	P_0
$\leftarrow P_2$	P_2	P_2

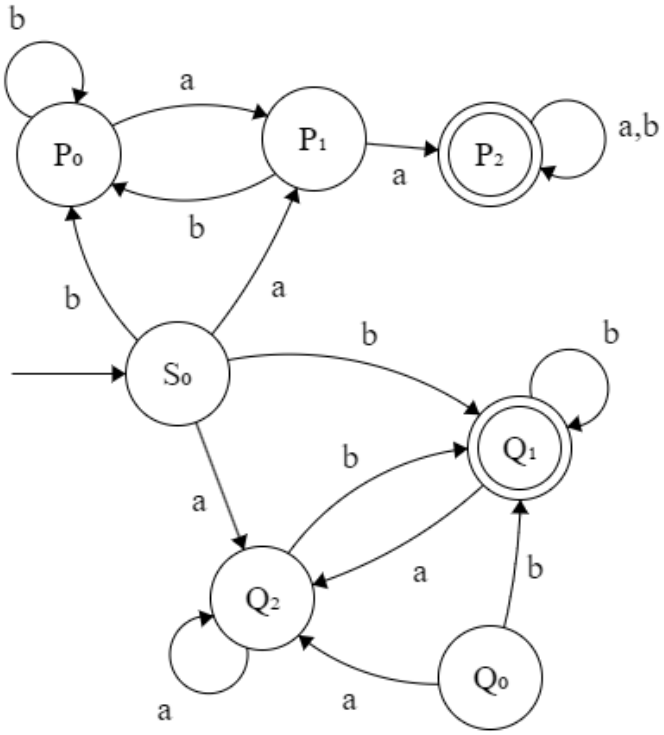
Tabulka 5. Přechody automatu A_2

Z výše uvedených tabulek přechodů můžeme sestavit tabulku přechodů pro náš nový automat.

A	a	b
$\rightarrow S_0$	$\{Q_2, P_1\}$	$\{Q_1, P_0\}$
Q_0	Q_2	Q_1
$\leftarrow Q_1$	Q_2	Q_1
Q_2	Q_2	Q_1
P_0	P_1	P_0
P_1	P_2	P_0
$\leftarrow P_2$	P_2	P_2

Tabulka 6. Přechody sjednocení automatů A_1 a A_2

Tímto jsme získali konečný automat A = (R, Σ, δ, q_, F), kde R = {S_0, Q_0, Q_1, Q_2, P_0, P_1, P_2}, Σ = {a, b}, q_ = S_0, F = {Q_1, P_2}, který je sjednocením automatu A_1 a A_2. Přechody automatu A jsou znázorněny na obrázku níže.



Obrázek 8. Sjednocení automatů A_1 a A_2

Sjednocení pomocí NKA je ilustrováno na příkladu 1.2 v příloze 1.

2.2.3 Sjednocení pomocí monoidů

Při sjednocení jazyků L_1 a L_2 pomocí monoidů reprezentovaných čtveřicí (M, Σ, ϕ, F), která rozpoznává jazyk L_1 a (N, Σ, ψ, G), která rozpoznává jazyk L_2, hledáme čtveřici, která rozpoznává sjednocení L_1 a L_2. [22]

Prvky nového monoidu získáme pomocí kartézského součinu prvků monoidu M a N. Následně dosadíme abecedu Σ. Homomorfismus (ϕ, ψ) nového monoidu vyjádříme pomocí obou homomorfismů. Pro všechna x∈Σ* platí že (ϕ, ψ) (x)=(ϕ(x), ψ(x)). Množina akceptovaných prvků bude reprezentována kartézským součinem množiny F a N sjednocenou s kartézským součinem množin G a M. Výsledný monoid bude vypadat následovně: (M × N, Σ, (ϕ, ψ), F × N ∪ G × M). [22]

2.3 Průnik

Značí se symbolem ∩ a používá se k vytvoření nového jazyka, který obsahuje pouze slova, která jsou zároveň v obou původních jazycích. Pokud jsou jazyky L_1 a L_2 regulární, pak jazyk $L = L_1 \cap L_2$ bude také regulární jazyk. Formálně lze průnik zapsat jako: $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ a } w \in L_2\}$. Stejně jako operace sjednocení tak i operace průniku je asociativní a komutativní. V praxi se průnik regulárních jazyků používá například při analýze a validaci vstupních dat. [10, 25]

2.3.1 Průnik pomocí regulárních výrazů

Ve standardních regulárních výrazech operace průniku není zahrnuta. Operaci průniku u regulárních výrazů je možné provést pomocí konečných automatů, které jsou s regulárními výrazy úzce spojeny. [1]

Nejjednodušším způsobem, jak provést operaci průniku dvou regulárních výrazů, je nejprve dané regulární výrazy převést na, ekvivalentní konečné automaty. Poté provedeme operaci průniku na nově vytvořených automatech, která je popsána v následující podkapitole. Po úpravě převedeme výsledný automat zpět na regulární výraz pomocí jedné z dostupných metod a získáme regulární výraz reprezentující průnik dvou regulárních výrazů. [1]

2.3.2 Průnik pomocí konečných automatů

Je základním konceptem v teorii automatů umožňující vytvoření nového automatu, který rozpoznává průnik jazyků rozpoznávaných dvěma nebo více konečnými automaty. Používá se k vytváření složitějších automatů, které mohou rozpoznávat určité podmnožiny jazyků. [8]

Formálně lze průnik dvou konečných automatů A_1 a A_2 definovat následujícími pravidly: [8, 10]

Stavy nového automatu jsou kartézským součinem stavů A_1 a A_2.

Vstupní abeceda nového automatu je průnikem abecedy automatů A_1 a A_2.

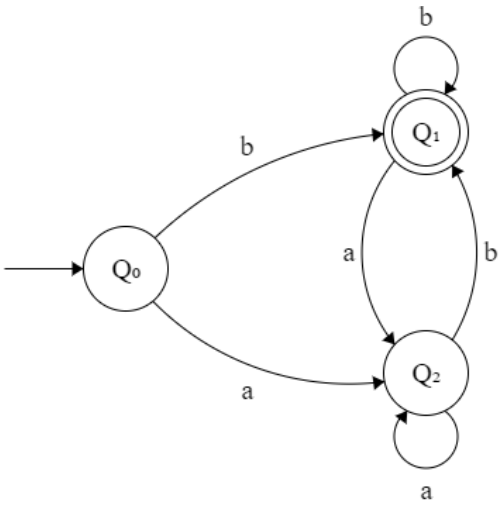
Přechodová funkce nového automatu je definována následovně: Pro každý stav (q_1, q_2) v novém automatu a každý vstupní symbol a v abecedě, je dalším stavem kartézský součin následujících stavů v A_1 a A_2 pro příslušné stavy a vstupní symbol, $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$.

Počáteční stav nového automatu je kartézským součinem počátečních stavů A_1 a A_2.

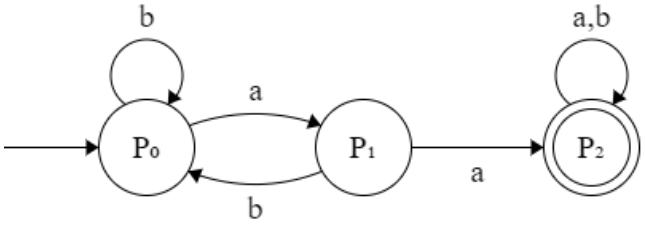
Konečné stavy nového automatu jsou stavy, které jsou konečné v obou A_1 a A_2.

Příklad průniku pomocí dvou DKA 2.3:

Konečné automaty A_1 definovaný v příkladu iterace DKA, a A_2 definovaný v příkladu sjednocení DKA.



Obrázek 1. DKA A_1



Obrázek 7. DKA A_2

Jejich průnik můžeme provést následujícím způsobem. Stejně jako u sjednocení i zde zkontrolujeme, zdali platí podmínka $Q \cap P = \emptyset$. Pokud podmínka neplatí, je nutné stavy jednoho z automatů přejmenovat, aby podmínka platila.

Opět si sestavíme tabulky přechodů pro automaty A_1 a A_2 stejně jako v předešlém příkladu.

A_1	a	b
$\rightarrow Q_0$	Q_2	Q_1
$\leftarrow Q_1$	Q_2	Q_1
Q_2	Q_2	Q_1

Tabulka 4. Přechody automatu A_1

A_2	a	b
$\rightarrow P_0$	P_1	P_0
P_1	P_2	P_0
$\leftarrow P_2$	P_2	P_2

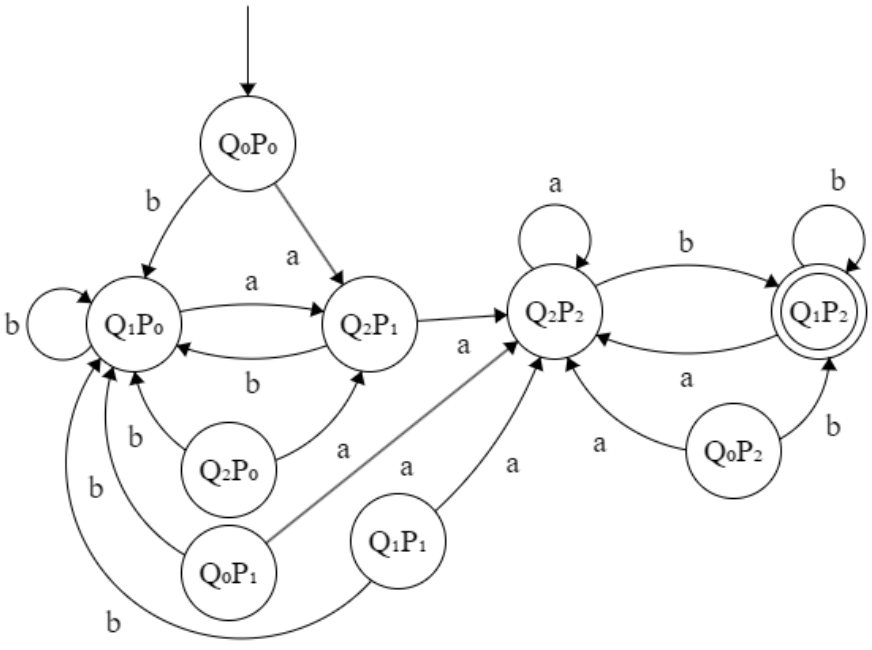
Tabulka 5. Přechody automatu A_2

Z tabulek přechodů A_1 a A_2 sestavíme tabulku přechodů pro náš nový automat.

A	a	b
→Q_0P_0	Q_2 P_1	Q_1 P_0
Q_0P_1	Q_2 P_2	Q_1 P_0
Q_0P_2	Q_2 P_2	Q_1 P_2
Q_1P_0	Q_2P_1	Q_1P_0
Q_1P_1	Q_2P_2	Q_1P_0
←Q_1P_2	Q_2P_2	Q_1P_2
Q_2P_0	Q_2 P_1	Q_1 P_0
Q_2P_1	Q_2 P_2	Q_1 P_0
Q_2P_2	Q_2 P_2	Q_1 P_2

Tabulka 7. Tabulka přechodů průniku automatů A_1 a A_2

Tímto jsme získaly konečný automat A = (R, Σ, δ, q, F), kde R = {Q_0P_0, Q_0P_1, Q_0P_2, Q_1P_0, Q_1P_1, Q_1P_2, Q_2P_0, Q_2P_1, Q_2P_2}, Σ = {a, b}, q = Q_0P_0, F = {Q_1P_2}, který je průnikem automatů A_1 a A_2.



Obrázek 9. Průnik automatů A_1 a A_2

Průnik pomocí NKA je ilustrován na příkladu 1.3 v příloze 1.

2.3.3 Průnik pomocí monoidů

Průniku monoidů reprezentovaných čtveřicí (M, Σ, ϕ, F), která rozpoznává jazyk L_1 a (N, Σ, ψ, G), která rozpoznává jazyk L_2, umožňuje získat monoid schopný rozpoznat průnik jazyků L_1 a L_2. [22]

Při průniku monoidů M a N získáme prvky nového monoidu pomocí kartézského součinu stejně jako u sjednocení. Následně dosadíme abecedu Σ. Homomorfismus (ϕ, ψ) nového monoidu reprezentují oba homomorfismy. Pro všechna $x \in \Sigma^*$ platí že $(\phi, \psi)(x) = (\phi(x), \psi(x))$. Množinou akceptovaných prvků bude kartézský součin množiny F a G. Výsledný monoid bude vypadat následovně: $(M \times N, \Sigma, (\phi, \psi), F \times G)$. [22]

2.4 Zřetězení

2.4.1 Zřetězení pomocí regulárního výrazu

Poslední základní operací regulárních výrazů je zřetězení. Je značena . (tečkou) nebo ničím, to znamená, že zřetězení regulárních výrazů R_1 a R_2 může být zapsána jako $R_1.R_2$ nebo R_1R_2. Kombinuje dva regulární výrazy do nového regulárního výrazu, který popisuje jazyk složený z všech možných zřetězení řetězců z jazyků rozpoznávaných původními regulárními výrazy. Například regulární výraz (a+b)b* odpovídá jakémukoliv řetězci, který začíná a nebo b a končí libovolným počtem b nebo je řetězec pouze a. [1]

2.4.2 Zřetězení pomocí konečného automatu

Dalším konceptem v teorii automatů je zřetězení umožňující nám vytvoření nového automatu, který rozpoznává zřetězení jazyků rozpoznávaných dvěma nebo více konečnými automaty. [8]

Formálně lze zřetězení dvou konečných automatů definovat následujícími pravidly: [8, 10]

Stavy nového automatu jsou sjednocením stavů A_1 a A_2.

Vstupní abeceda nového automatu je sjednocením abecedy automatů A_1 a A_2.

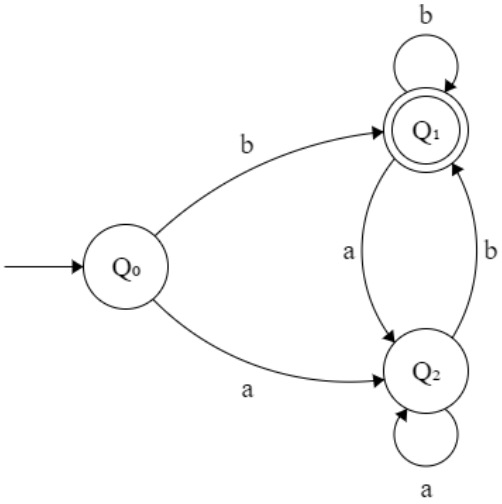
Přechody nového automatu jsou stejné jako přechody automatu A_1 a A_2 a z A_1 jsou do počátečního stavu A_2 přidány všechny přechody, které vedou do koncových stavů v A_1.

Počáteční stav nového automatu je počáteční stav A_1.

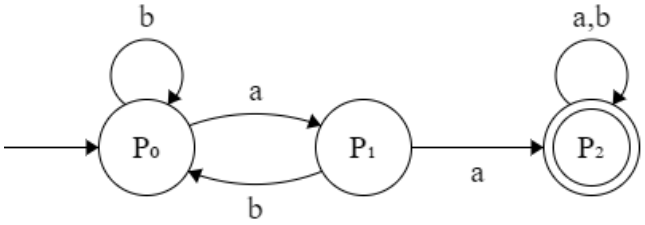
Koncové stavy nového automatu jsou stavy, které jsou koncové v A_2.

Příklad zřetězení pomocí dvou DKA 2.4:

Konečné automaty A_1 definovaný v příkladu iterace DKA, a A_2 definovaný v příkladu sjednocení DKA.



Obrázek 1. DKA A_1



Obrázek 7. DKA A_2

Při provádění zřetězení jako první opět zkontrolujeme, zdali platí podmínka $Q \cap P = \emptyset$. Pokud podmínka neplatí, je nutné stavy jednoho z automatů přejmenovat, aby podmínka platila. Následně si sestavíme tabulky přechodů pro automaty A1 a A2 stejně jako v příkladu sjednocení DKA.

A_1	a	b
$\rightarrow Q_0$	Q_2	Q_1
$\leftarrow Q_1$	Q_2	Q_1
Q_2	Q_2	Q_1

Tabulka 4. Přechody automatu A_1

A_2	a	b
$\rightarrow P_0$	P_1	P_0
P_1	P_2	P_0
$\leftarrow P_2$	P_2	P_2

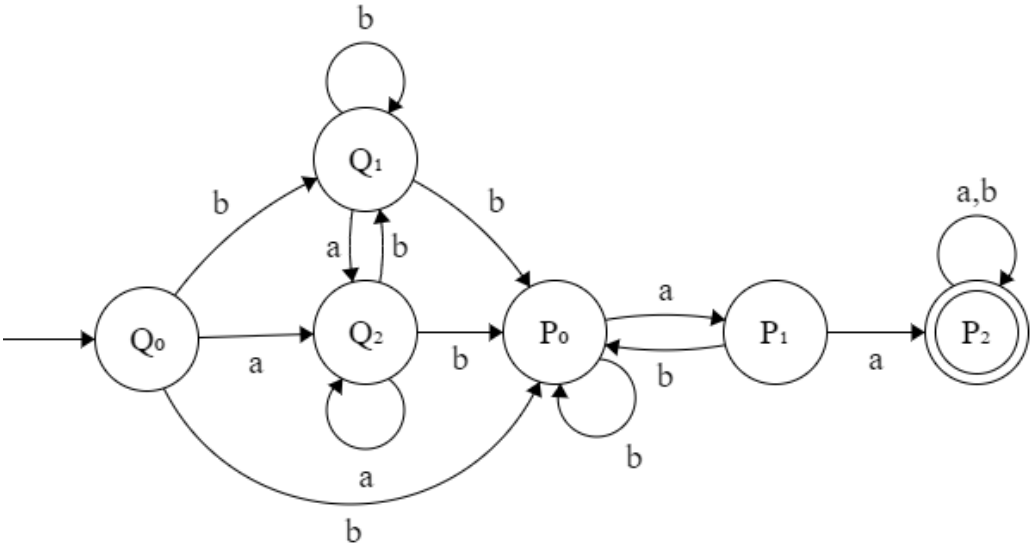
Tabulka 5. Přechody automatu A_2

Z výše uvedených tabulek přechodů můžeme sestavit tabulku přechodů pro náš nový automat.

A	a	b
→Q_0	Q_2	{Q_1, P_0}
Q_1	Q_2	{Q_1, P_0}
Q_2	Q_2	{Q_1, P_0}
P_0	P_1	P_0
P_1	P_2	P_0
←P_2	P_2	P_2

Tabulka 8. Tabulka přechodů zřetězení automatů A_1 a A_2

Tímto jsme získali konečný automat A = (R, Σ, δ, q_, F), kde R = {Q_0, Q_1, Q_2, P_0, P_1, P_2}, Σ = {a, b}, q_ = Q_0, F = {P_2}, který je zřetězením automatu A_1 a A_2.



Obrázek 10. Zřetězení automatů A_1 a A_2

Zřetězení pomocí NKA je ilustrováno na příkladu 1.4 v příloze 1.

2.5 Doplněk

Tato operace se značí pruhem nad původním regulárním jazykem. Doplněk regulárního jazyka je operace produkující nový jazyk, který obsahuje všechna slova, která neobsahuje

původní regulární jazyk. Formálně lze doplněk zapsat jako: $\overline{L} = \Sigma^* - L = \{w \in \Sigma^*; w \notin L\}$. [10]

2.5.1 Doplněk pomocí regulárního výrazu

Stejně jako operace průniku i operace doplňku není zahrnuta ve standardních regulárních výrazech. Provádět operaci doplňku na regulárních výrazech můžeme díky již zmíněnému úzkému spojení regulárních výrazů a konečných automatů. [1]

Nejjednodušším způsobem, jak provést operaci doplňku regulárního výrazu, je opět daný regulární výraz převést na ekvivalentní konečný automat a následně provést operaci doplňku na získaném automatu, která je popsána v následující podkapitole. Po úpravě převedeme výsledný automat zpět na regulární výraz. Tímto získáme regulární výraz $\overline{R}$, který označuje doplněk původního regulárního výrazu R. [1]

2.5.2 Doplněk pomocí konečného automatu

V teorii automatů je doplněk konečného automatu konečným automatem, který rozpoznává doplněk jazyka rozpoznávající původní automat. Doplněk jazyka L je definován jako množina všech řetězců v abecedě Σ^*, které nejsou v L. Pro získání doplňku konečného automatu nejprve převedeme automat na DKA. Poté vyměníme koncové a nekoncové stavy DKA, a to nám vytvoří nový DKA, který rozpoznává doplněk jazyka, který rozpoznává původní automat. [8]

Formálně lze operaci doplňku konečného automatu definovat následujícími pravidly: [8, 10]

Stavy nového automatu jsou stejné jako stavy původního automatu.

Vstupní abeceda nového automatu je stejná jako vstupní abeceda původního automatu.

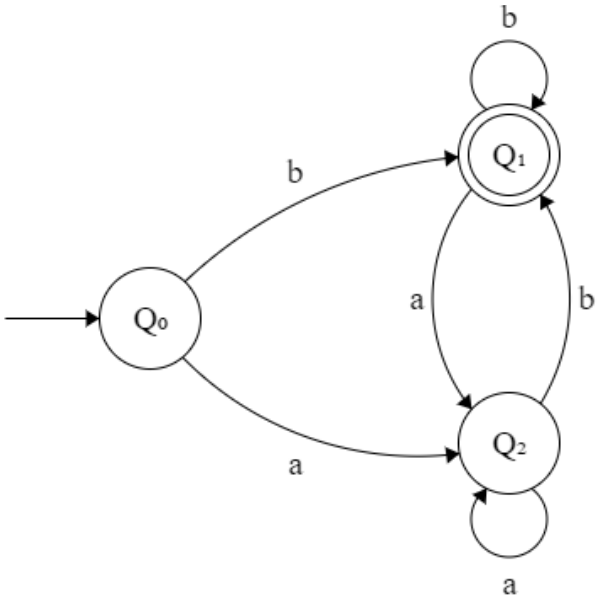
Přechodová funkce nového automatu je stejná jako přechodová funkce automatu původního.

Počáteční stav nového automatu je stejný jako počáteční stav původního automatu

Koncové stavy nového automatu jsou stavy, které nejsou koncové v původním automatu.

Příklad doplňku pomocí DKA 2.5:

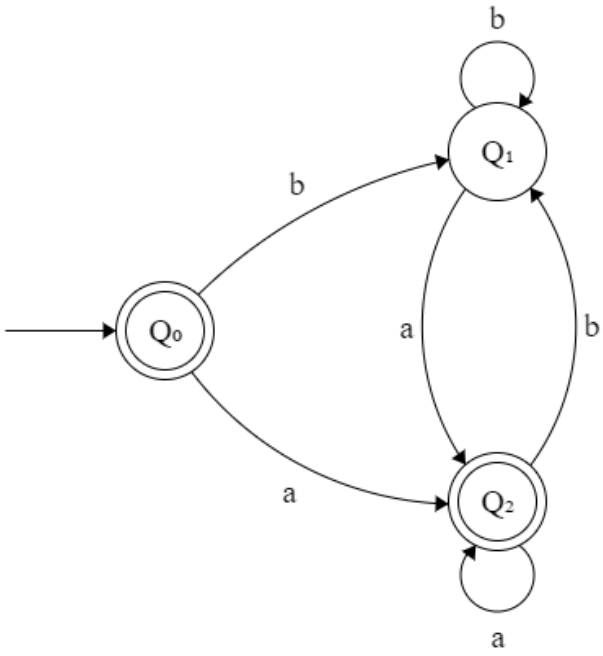
Konečné automaty A_1 definovaný v příkladu iterace DKA.



Obrázek 1. DKA A_1

Při provádění doplňku změníme koncové stavy na nekoncové a naopak.

Získáme konečný automat A = (R, Σ, δ, q_, F), kde R = {Q_0, Q_1, Q_2}, Σ = {a, b}, q_ = Q_0, F = {Q_0, Q_2}, který je doplňkem automatu A1.



Obrázek 11. Doplněk automatu A_1

Doplněk pomocí NKA je ilustrován na příkladu 1.5 v příloze 1.

2.5.3 Doplněk pomocí monoidu

Doplněk monoidu reprezentovaného čtveřicí (M, Σ, ϕ, F), která rozpoznává jazyk L_1, umožňuje získat monoid schopný rozpoznat doplněk jazyku L_1. [22]

Při doplňku monoidu M jsou prvky nového monoidu stejné jako u původního monoidu. Abecedu Σ a homomorfismus ϕ nového monoidu jsou taktéž stejné jako u původního monoidu. Množinou akceptovaných prvků bude rozdíl množiny M a podmnožiny F z množiny M. Výsledný monoid bude vypadat následovně: (M, Σ, ϕ, M\F). [22]

3 PŘEVODY

3.1 Převod regulárního výrazu na konečný automat

Převod regulárního výrazu na konečný automat je proces, kterým lze z daného regulárního výrazu vytvořit ekvivalentní konečný automat, který rozpoznává stejný jazyk jako původní regulární výraz popisuje. Existuje několik algoritmů pro převod regulárního výrazu na konečný automat, z nichž některé jsou popsány níže. [26, 27]

3.1.1 Thompsonův algoritmus

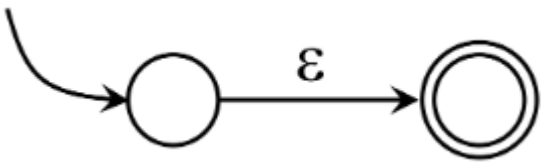
Prvním způsobem převodu regulárního výrazu je pomocí Thompsonova algoritmu, někdy také nazývaného McNaughton–Yamada–Thompsonův algoritmus. Tento algoritmus transformuje regulární výraz na NKA rozpoznávající jazyk regulárního výrazu. Algoritmus funguje na principu vytvoření jednotlivých automatů pro podvýrazy a následně je spojí pomocí epsilon přechodů podle operace, která kombinuje regulární výrazy. [27]

Formálně lze NKA vytvořený tímto algoritmem, který převádí regulární výraz R o délce n na NKA popsat následovně: [27]

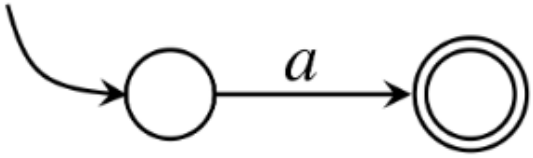
- Má vždy jeden počáteční a jeden konečný stav
- Nejvíce bude obsahovat 2n stavů
- Neexistují přechody do jeho počátečního stavu
- Neexistují přechody vycházející z jeho konečného stavu
- Každý stav má nejvýše 2 přechody

Při tvoření algoritmu se řídíme několika základními pravidly.

Pokud regulární výraz obsahuje pouze 1 symbol, je převeden na konečný automat následovně pomocí epsilon přechodu nebo přechodu s patřičným symbolem.

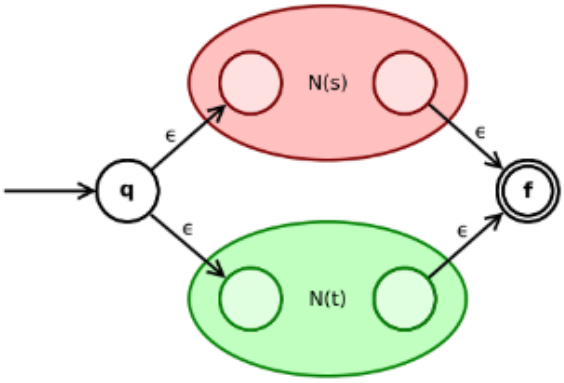


Obrázek 12. Thompsonův algoritmus - epsilon přechod [27]



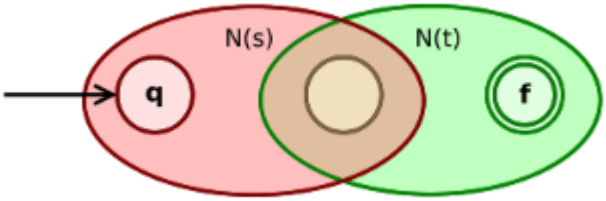
Obrázek 13. Thompsonův algoritmus - přechod se symbolem [27]

Sjednocení s a t je kombinací podvýrazů, kde sdílejí počáteční a konečný stav.



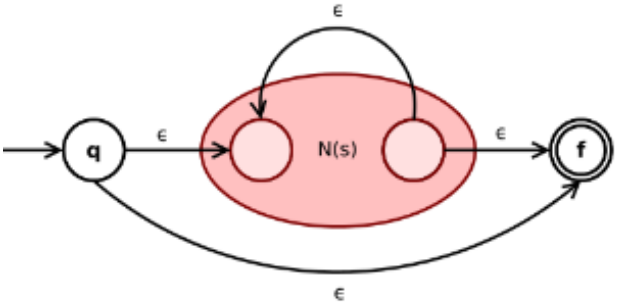
Obrázek 14. Thompsonův algoritmus - sjednocení [27]

Při zřetězení s a t jsou kombinovány tak, že konečný stav s je počáteční stav t.



Obrázek 15. Thompsonův algoritmus - zřetězení [27]

Při iteraci se vytvoří dva nové stavy, počáteční a koncový. Provede se epsilon přechod z počátečního do koncového stavu, což nám umožní přijímat prázdné řetězce a další z původního koncového stavu do původního počátečního stavu, který nám umožní opakování.

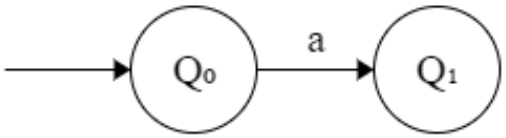


Obrázek 16. Thompsonův algoritmus - iterace [27]

Příklad 3.1:

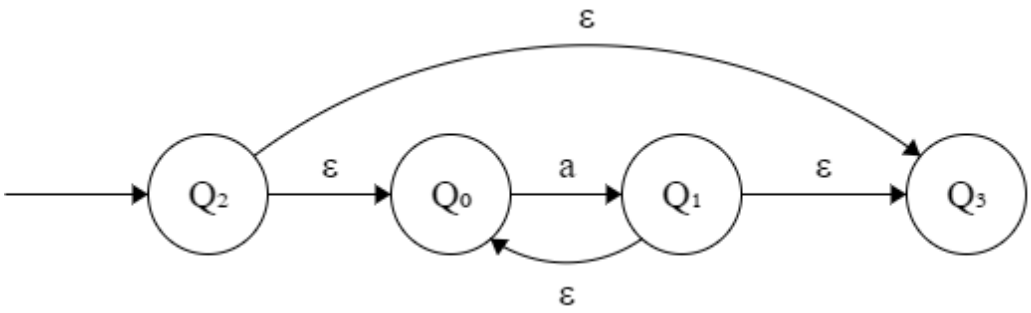
Máme regulární výraz a*b(b+aa*b)*

Nejprve vytvoříme automat pro výraz a podle vzoru, který je vyobrazený výše.



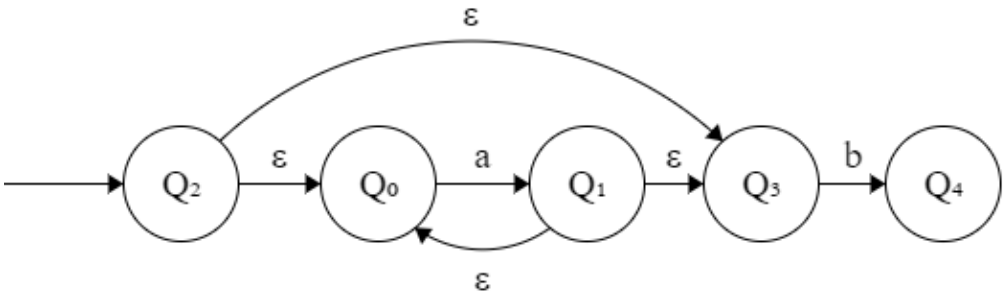
Obrázek 17. Příklad thompsonova algoritmu - automat a

Po vytvoření automatu na něj aplikujeme iteraci a získáme automat zobrazující a*.



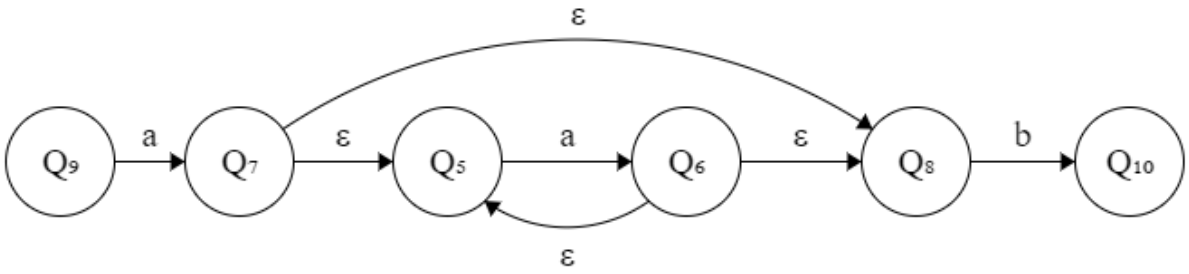
Obrázek 18. Příklad thompsonova algoritmu - automat a*

Následně můžeme již vytvořený výraz propojit s zřetězením b a tím získáme automat reprezentující výraz a*b.



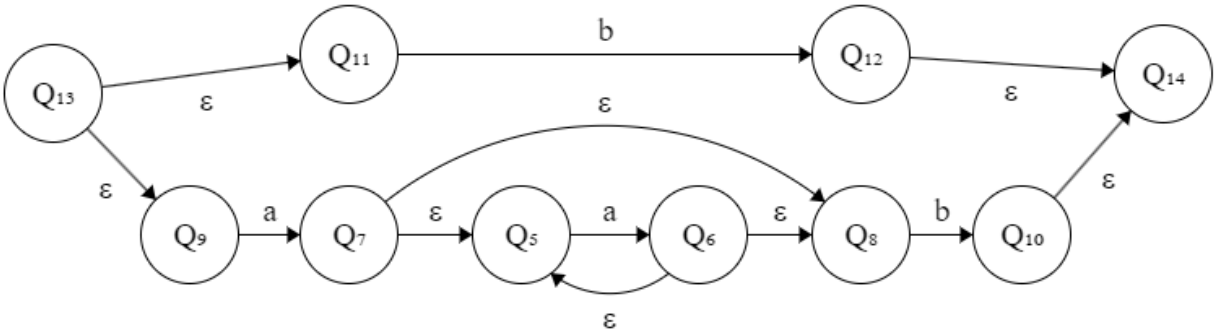
Obrázek 19. Příklad thompsonova algoritmu - automat a*b

V dalším kroku zpracuje zbývající část výrazu, která je (b+aa*b)*. Jednou z možností, jak tento výraz zpracovat, je nejprve převést jednotlivé části v závorkách. Nejprve převedeme na automat a* jako v předešlých krocích a následně k němu připojíme zřetězení a,b a získáme automat reprezentující výraz aa*b, což bude vypadat následovně.



Obrázek 20. Příklad thompsonova algoritmu - automat aa*b

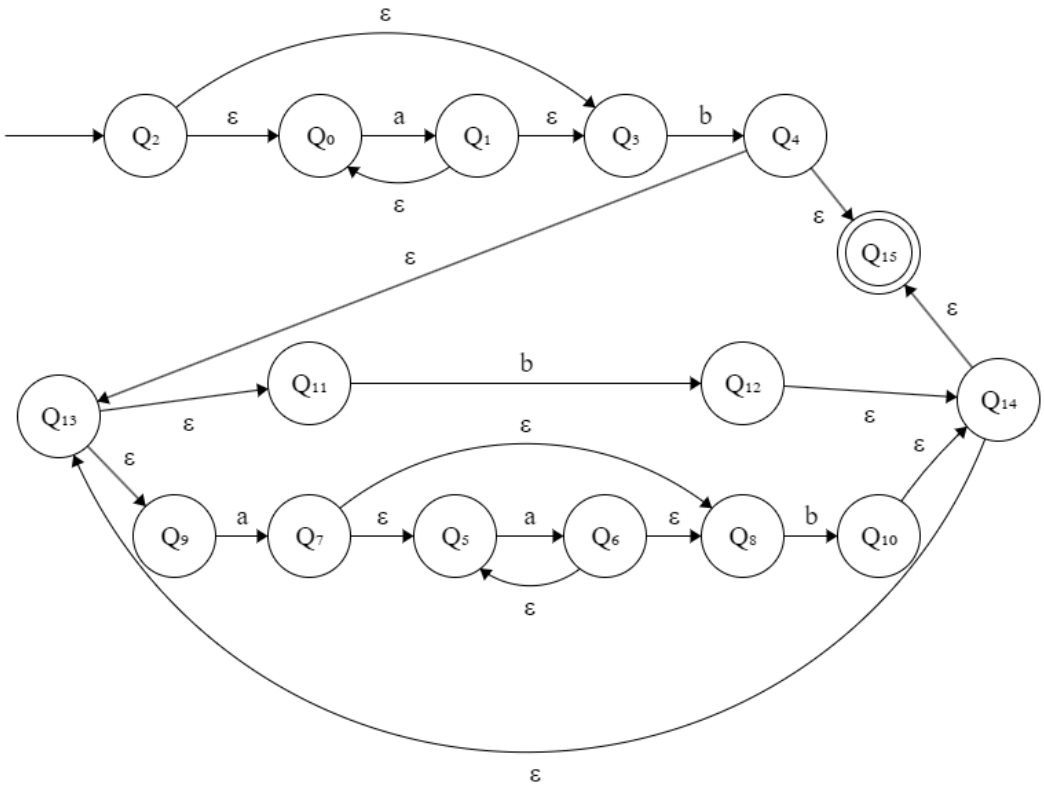
Po zpracování výrazu aa*b vytvoříme automat pro výraz b+aa*b. Ten následně můžeme pomocí závorek propojit a získáme automat (b+aa*b).



Obrázek 21. Příklad thompsonova algoritmu - automat (b+aa*b)

Po vytvoření automatu (b+aa*b) na něj aplikujeme iteraci a získáme automat (b+aa*b)*, který následně propojíme s již sestaveným automatem a*b pomocí epsilon přechodu a získáme výsledný automat ekvivalentní výrazu a*b(b+aa*b)*.

A = (Q, Σ, δ, q_, F), kde Q = {Q_0, Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8, Q_9, Q_{10}, Q_{11}, Q_{12}, Q_{13}, Q_{14}, Q_{15}}, Σ = {a, b}, q_ = Q_0, F = {Q_{15}}, přechody jsou zobrazeny na následujícím obrázku.



Obrázek 22. Příklad thompsonova algoritmu - automat a*b(b+aa*b)*

3.1.2 Glushkův algoritmus

Dalším možným algoritmem je Glushkův algoritmus.

Tento algoritmus vytvoří pro každý regulární výraz ekvivalentní NKA, kde počet stavů NKA je roven počtu symbolů regulárního výrazu. Princip algoritmu je vysvětlen na následujícím příkladu.

Příklad 3.2: [28, 29]

Máme regulární výraz a*b(b+aa*b)*

Nejprve výraz linearizujeme přidáním sekvenčního indexu jednotlivým symbolům

a_1*b_2(b_3+a_4a_5*b_6)*

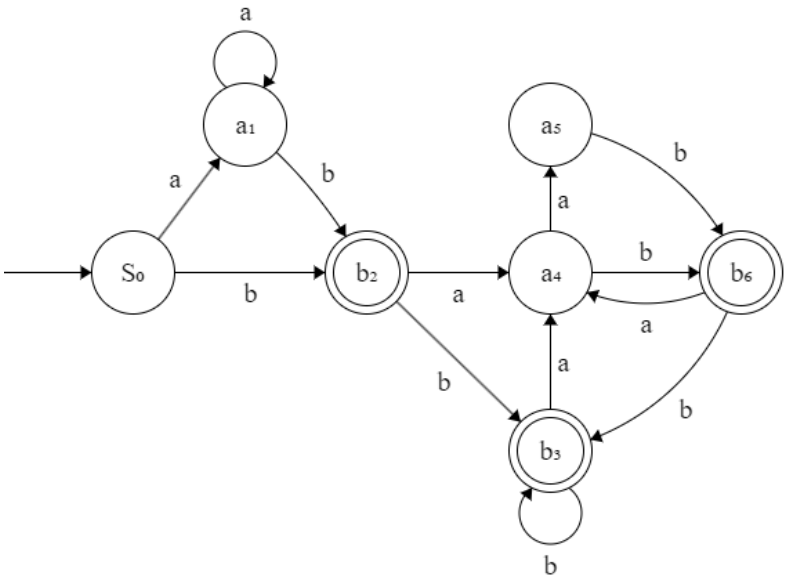
Po linearizaci regulárního výrazu vytvoříme počáteční stav S_0 a pro každý symbol vytvoříme nový stav také. Tímto krokem získáme seznam stavů Glushkova automatu, který v tomto případě vypadá následovně: Q = {S_0, a_1, b_2, b_3, a_4, a_5, b_6}.

Po vytvoření stavů určíme počáteční symboly, což jsou symboly, které se mohou vyskytovat u řetězců odpovídajících regulárnímu výrazu na prvním místě. U tohoto regulárního výrazu jsou počáteční symboly: P = {a_1, b_2}

V tomto kroku se určí konečné symboly stejným způsobem, jako v předešlém kroku byly určeny počáteční symboly. Konečné symboly jsou všechny symboly, kterými může končit řetězec odpovídající regulárnímu výrazu. U tohoto regulárního výrazu jsou konečné symboly: K = {b_2, b_3, b_6}

Po určení počátečních a konečných symbolů určíme páry symbolů, což jsou jakékoliv 2 symboly, které se mohou v řetězci vyskytnou za sebou. U tohoto příkladu jsou páry následující: Pa = {(a_1, a_1), (a_1, b_2), (b_2, b_3), (b_2, a_4), (b_3, b_3), (b_3, a_4), (a_4, a_5), (a_4, b_6), (a_5, a_5), (a_5, b_6), (b_6, b_3), (b_6, a_4)}

Nyní je možné sestrojit konečný automat. Z počátečního stavu S_0 povedou přechody do počátečních symbolů P. Znak přechodu je určen podle znaku stavu, do kterého jde bez přidaného indexu. Následně jsou vytvořeny jednotlivé přechody pomocí párů, a nakonec jsou určeny konečné stavy pomocí konečných symbolů K. Vytvořený konečný automat vypadá následovně: G = (Q, Σ, δ, q_, F), kde Q = {S_0, a_1, b_2, b_3, a_4, a_5, b_6}, Σ = {a, b}, q_ = S_0, F = {b_2, b_3, b_6}, přechody jsou znázorněny na obrázku.



Obrázek 23. Glushkův automat G

3.1.3 Metoda dekompozice

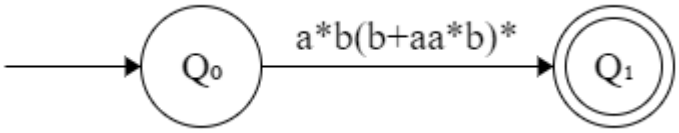
Tato metoda vytvoří konečný automat s jedním koncovým stavem. Při převodu regulárního výrazu R na konečný automat si jako první určíme počáteční a koncový stav a přechod mezi nimi označíme daným regulárním výrazem R. Pokud je R iterací (a)* vytvoříme pouze 1 stav, který bude zároveň počátečním a konečným stavem. Po určení počátečního a koncového stavu opakovaně aplikujeme následující pravidla, dokud výraz není kompletně rozložen. [26]

Sjednocení lze eliminovat pomocí zavedení paralelních přechodů mezi 2 stavy. Operátor zřetězení lze eliminovat pomocí vytvoření nového stavu mezi 2 stavy, kde se tento operátor nachází. Poslední operátor je iterace a eliminujeme ji přechodem stavu do sebe samého. [26]

Příklad 3.3:

Máme regulární výraz R = a*b(b+aa*b)*

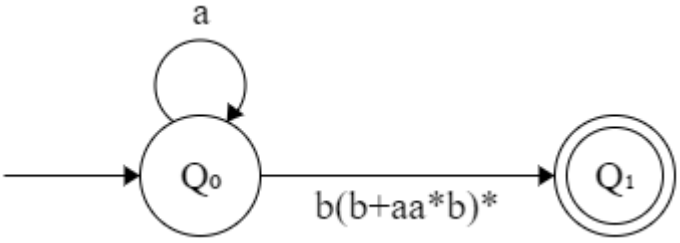
V prvním kroku vytvoříme počáteční a koncový stav automatu mezi kterými bude přechod označen regulárním výrazem R.



Obrázek 24. Počáteční automat dekompozice

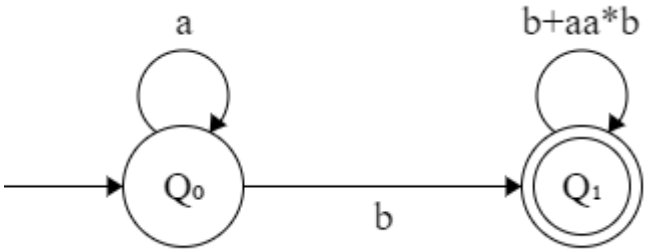
V dalších krocích rozkládáme jednotlivé části regulárního výrazu dle jejich operátoru.

Rozložíme a* pomocí přechodu a z Q_0 do Q_0.



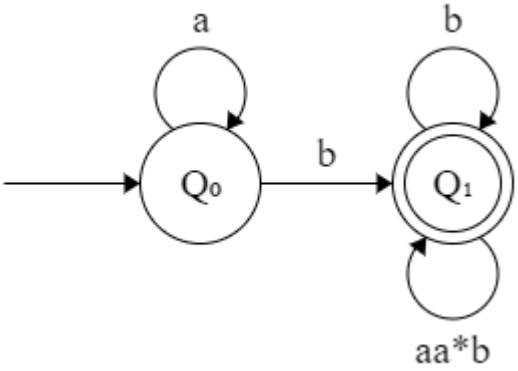
Obrázek 25. Automat dekompozice se zpracovaným a*

Dále rozložíme b(b+aa*b)* a uděláme z b+aa*b přechod z Q_1 do Q_1 a tím získáme samotný přechod b.



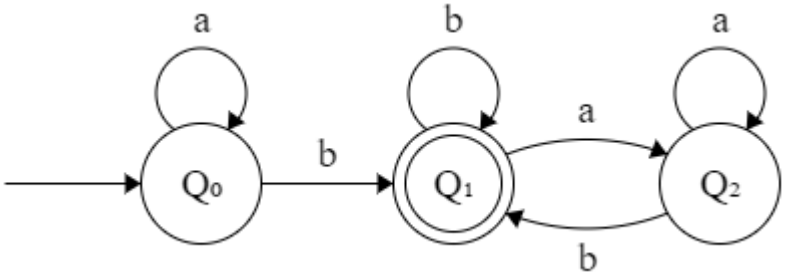
Obrázek 26. Automat dekompozice se zpracovaným a*b

Následně rozdělíme b a aa*b na samostatné přechody.



Obrázek 27. Automat dekompozice se zpracovaným a*b a rozděleným b a aa*b

Nakonec vytvoříme nový stav Q_2 a pomocí něj rozložíme aa*b. Tímto získáme finální konečný automat, který bude vypadat následovně. A = (Q, Σ, δ, q_, F), kde Q = {Q_0, Q_1, Q_2}, Σ = {a, b}, q_ = Q_0, F = {Q_1}, přechody jsou znázorněny na obrázku.



Obrázek 28. Výsledný automat dekompozice

3.2 Převod konečného automatu na regulární výraz

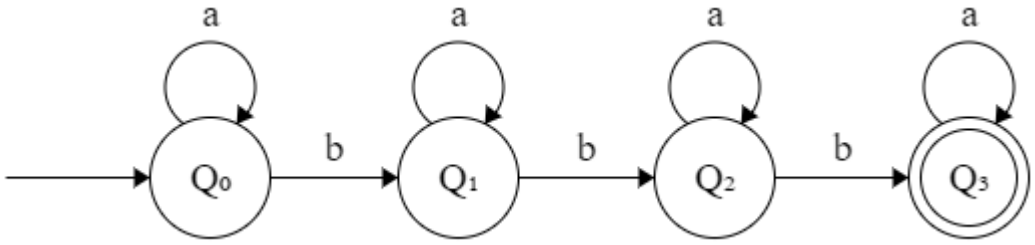
Každý konečný automat lze převést na ekvivalentní regulární výraz. Existuje několik způsobů, jak provést tuto konverzi. Jedním z nejznámějších a nejjednodušších způsobů je metoda odstraňování stavů.

3.2.1 Metoda odstraňování stavů

Tato metoda je velmi podobná metodě dekompozice u převodu regulárních výrazů na konečné automaty a spočívá v postupném odstraňování stavů z konečného automatu, dokud nezůstane pouze počáteční a konečný stav. Každý odstraněný stav je nahrazen novým regulárním výrazem, který popisuje všechny cesty mezi vstupním stavem a výstupním stavem, které procházejí tímto stavem. Výsledkem bude regulární výraz, který popisuje jazyk přijímaný původním automatem. Je také nutné podotknout, že výsledný regulární výraz nemusí být vždy optimální, a existují různé techniky pro minimalizaci regulárních výrazů, které mohou být použity pro optimalizaci výsledného výrazu. [30]

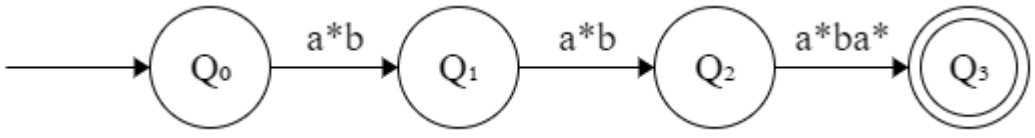
Příklad 3.4:

Máme konečný automat A = (Q, Σ, δ, q_, F), kde Q = {Q_0, Q_1, Q_2, Q_3}, Σ = {a, b}, q_ = Q_0, F = {Q_3}, přechody jsou znázorněny na obrázku.



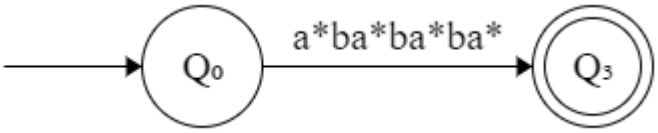
Obrázek 29. Automat A pro převod na regulární výraz

V prvním kroku odstraníme veškeré iterace a získáme automat tvořený pouze ze zřetězení.



Obrázek 30. Automat A s odstraněnými iteracemi

Zřetězení spojíme a získáme výsledný regulární výraz a*ba*ba*ba*.



Obrázek 31. Automat A s odstraněným zřetězením

3.2.2 Úpravy regulárních výrazů

Předtím než začneme převádět konečné automaty na regulární výrazy pomocí následující metody, je nutné znát určité operace a pravidla.

3.2.2.1 Identity regulárních jazyků

Identity regulárních jazyků jsou pravidla popisující ekvivalentní způsoby zápisu regulárních jazyků. Tyto identity se používají k usnadnění práce s regulárními výrazy a ke zjednodušení jejich zápisu.

Identity: [31]

- L + M = M + L
- (L + M) + N = L + (M + N)
- (L.M).N = L .(M.N)
- Φ + L = L + Φ = L
- ε.L = L.ε = L
- Φ.L = L.Φ = Φ
- L.(M + N) = L.M + L.N
- (M + N).L = M.L + N.L
- L + L = L
- (L*)* = L*
- Φ* = ε
- ε* = ε
- (L.M)*.L = L.(M.L)*

3.2.2.2 Ardenův teorém

Ardenův teorém se využívá při řešení systémů lineárních rovnic nad regulárními výrazy.

Formální zápis Ardenova teorému zní:

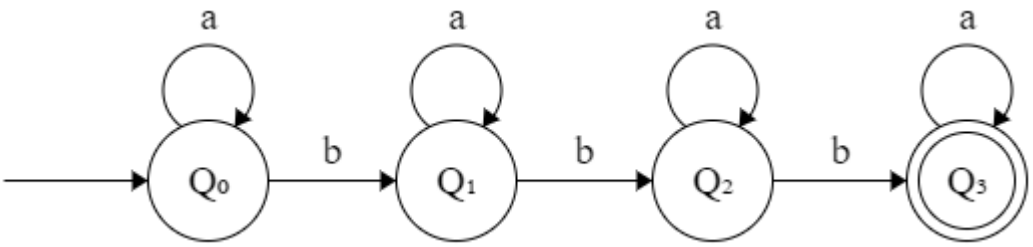
Pokud P a Q jsou dva regulární výrazy nad Σ a pokud P neobsahuje ε, pak následující rovnice R = Q + RP má jediné řešení R = QP*. [32]

3.2.3 Ardenova metoda

Další možnou metodou je Ardenova metoda, při které tvoříme z jednotlivých stavů automatu rovnice a ty následně pomocí identit a Ardenova teorému upravujeme pro dosažení výsledného regulárního výrazu. [32]

Příklad 3.5:

Máme konečný automat A = (Q, Σ, δ, q_, F), kde Q = {Q_0, Q_1, Q_2, Q_3}, Σ = {a, b}, q_ = Q_0, F = {Q_3}, přechody jsou znázorněny na obrázku, definovaný v předešlém příkladu.



Obrázek 29. Automat A pro převod na regulární výraz

Z jednotlivých stavů konečného automatu vytvoříme následující rovnice:

$Q_0 = \varepsilon + Q_0a$

$Q_1 = Q_0b + Q_1a$

$Q_2 = Q_1b + Q_2a$

$Q_3 = Q_2b + Q_3a$

Nejprve upravíme první rovnici.

$Q_0 = \varepsilon + Q_0a$

Na rovnici aplikujeme Ardenův teorém, kde R = Q_0, Q = ε, P = a. Vznikne nám nová rovnice:

$Q_0 = \varepsilon a^*$

Zde jde pomocí identity εR = R upravit rovnice, kde za R dosadíme a* a dostaneme:

$Q_0 = a^*$

Po úpravě první rovnice pokračujeme úpravou rovnice druhé

$Q_1 = Q_0b + Q_1a$

Jelikož z předešlé úpravy známe Q_0, tak ho můžeme dosadit:

$Q_1 = a^*b + Q_1a$

Opět aplikujeme Ardenův teorém stejným způsobem jako v předešlém případě.

$Q_1 = a^*ba^*$

Dále pokračujeme v úpravě jednotlivých rovnic stejně jako doposud.

$Q_2 = Q_1b + Q_2a$

$Q_2 = a^*ba^*b + Q_2a$

$Q_2 = a^*ba^*ba^*$

$Q_3 = Q_2b + Q_3a$

$Q_3 = a^*ba^*ba^*b + Q_3a$

$Q_3 = a^*ba^*ba^*ba^*$

Jelikož je Q_3 konečným stavem a obsahuje pouze znaky vstupní abecedy, dokončili jsme převod a získali regulární výraz odpovídající konečnému automatu. Výsledný regulární výraz vypadá následovně: a*ba*ba*ba*

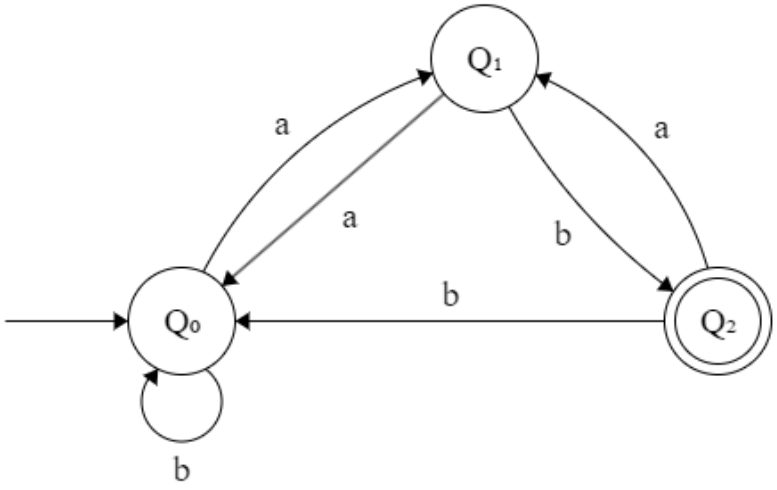
3.3 Převod DKA na monoid

Převod konečného automatu na monoid je proces transformace struktury konečného automatu do struktury monoidu. Tento převod umožňuje vyjádřit vlastnosti konečného automatu pomocí algebraických konceptů monoidu.

Máme DKA A = (Q, Σ, δ, q_, F). Každé písmeno a ∈ Σ definuje částečnou transformaci q → q·a na množině stavů Q. Tato transformace přiřazuje každému stavu q unikátní stav q·a, takový, že (q, a, q·a) ∈ δ. Pokud takový stav neexistuje, q·a je nedefinován.

Příklad 3.6:

DKA A = (Q, Σ, δ, q_, F), kde Q = {Q_0, Q_1, Q_2}, Σ = {a, b}, q_ = Q_0, F = {Q_2}, přechody jsou znázorněny na obrázku níže.



Obrázek 32. Automat A pro převod na monoid

Nejprve sestavíme tabulku znázorňující, kde konečný automat skončí v případě, že je zadáno slovo o délce 1, které je zobrazeno v prvním sloupci a začíná ve stavu zapsaném v prvním řádku.

B	Q_0	Q_1	Q_2
a	Q_1	Q_0	Q_1
b	Q_0	Q_2	Q_0

Tabulka 9. Tabulka znázorňující posun automatu A u slov o délce 1

Následně k tabulce, přidáme slova o délce 2 a kontrolujeme, zdali není kombinace stavů, ve kterých automat skončil již zobrazena v naší tabulce. V tomto případě se žádná kombinace stavů neopakovala.

B	Q_0	Q_1	Q_2
a	Q_1	Q_0	Q_1
b	Q_0	Q_2	Q_0
aa	Q_0	Q_1	Q_0
ab	Q_2	Q_0	Q_2
ba	Q_1	Q_1	Q_1
bb	Q_0	Q_0	Q_0

Tabulka 10. Tabulka znázorňující posun automatu A u slov o délce 2

K tabulce se slovy o délce 1 a 2 přidáme slova o délce 3. A opět kontrolujeme kombinace stavů. Tentokrát se nám některé kombinace opakují, a tak z nich můžeme vytvořit přepisovací pravidla. Jelikož aaa skončí ve stejný stavech jako a vytvoříme přepisovací pravidlo aaa → a. V dalšího slova aab skončíme ve stejných stavech jako b a tak můžeme vytvořit přepisovací pravidlo aab → b. Takto pokračujeme dál dokud neprojdeme všechna slova o délce 3.

Přepisovací pravidla:

aaa → a

aab → b

aba → ba bba → ba

abb → bb baa → bb bbb → bb

B	Q_0	Q_1	Q_2
a	Q_1	Q_0	Q_1
b	Q_0	Q_2	Q_0
aa	Q_0	Q_1	Q_0
ab	Q_2	Q_0	Q_2
ba	Q_1	Q_1	Q_1
bb	Q_0	Q_0	Q_0
bab	Q_2	Q_2	Q_2

Tabulka 11. Tabulka znázorňující posun automatu A u slov o délce 3

Jelikož stále nemám žádná přepisovací pravidla spojena s aa, ab a v minulé tabulce nám přibylo bab přidáme ke stávající tabulce slova o délce 4. Proces probíhá stejně jako u minulé tabulky, ale navíc zde kontrolujeme, zdali je možné slova zkrátit pomocí přepisovacích pravidel, která již máme. Například aaba je možné zkrátit na aba pomocí aba → ba a tak máme toto slovo již pokryto. Dalším příkladem může být abbb kde můžeme pomocí abb → bb zkrátit slovo na bbb nebo je také možné slovo zkrátit pomocí bbb → bb na abb.

Přepisovací pravidla:

aaa → a

aab → b

aba → ba bba → ba

abb → bb baa → bb bbb → bb

aaaa → aa

aaab → ab

abab → bab bbab → bab

Jelikož se tabulka nezměnila můžeme použít tabulku 11. Nyní když má každé zobrazené slovo v tabulce přiřazeno přepisovací pravidlo můžeme pomocí nich vytvořit monoid, který bude vypada následovně: ⟨ {a, b} | aaa = a, aab = b, aba = ba, bba = ba, abb = bb, baa = bb, bbb = bb, aaaa = a, aaab = ab, abab = bab, bbab = bab ⟩. Příslušný regulární jazky můžeme reprezentovat pomocí čtveřice (M, Σ, ϕ, F), kde Σ je abeceda automatu A. Homomorfismus ϕ přiřadí každému řetězci x ze Σ* prvek M reprezentující řetězec x: ϕ(x) = [x]. F reprezentuje řetězce přijímané automatem, F = {[ab], [bab]}.

II. PRAKTICKÁ ČÁST

4 WOLFRAM MATHEMATICA

V této kapitole si ukážeme, jak můžeme provést v programu Wolfram Mathematica některé operace nebo převody z předešlých kapitol. Použijeme k tomu balíček Automata od Klause Sutnera.

4.1 Operace

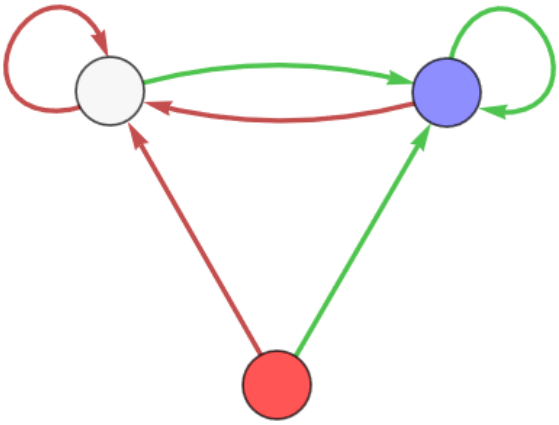
4.1.1 Iterace pomocí automatů

4.1.1.1 Iterace pomocí DKA

První vytvoříme DKA KA1, který reprezentuje automat A_1 z teoretické části, pomocí funkce FA, do které zadáváme jako vstupní parametr systém přechodů, který obsahuje počet stavů, počet znaků abecedy, přechody, samotnou abecedu a typ automatu. Druhý vstupní parametr určuje počáteční a koncové stavy. Po vytvoření automatu ho vykreslíme pomocí funkce PlotFA. Červené kolečko znázorňuje počáteční stav a modré znázorňuje stav koncový. Červené přechody jsou přechody a a zelené přechody jsou přechody b. Veškeré další automaty budou vykreslovány pomocí funkce PlotFA s identickými parametry jako má tato.

```
KA1 = FA[
  TSys[3, 2, {DirectedEdge[1,2,2], DirectedEdge[1,3,1], DirectedEdge[2,2,2],
DirectedEdge[2,3,1], DirectedEdge[3,2,2], DirectedEdge[3,3,1]},
   Alpha[2, <||>], <|"TransType" -> "DFA"|>], <|"AccCond" ->
    Existential[{1}, {2}]|>];
PlotFA[KA1, Type -> "MultiColor", EdgeStyle -> Thick,
 VertexSize -> Medium, "Trim" -> False]
```

Výstup:

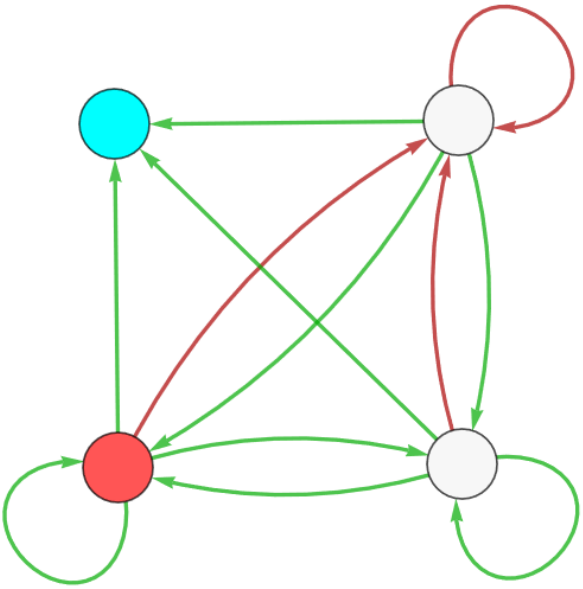


Obrázek 33. Wolfram automat KA1

Následně s automatem KA1 provedeme operaci iterace pomocí funkce KleeneStarFA a získáme automat KS1. Světle modrý kruh znázorňuje stav, který je zároveň koncový i počáteční.

```
KS1 = KleeneStarFA[KA1];
```

Výstup:

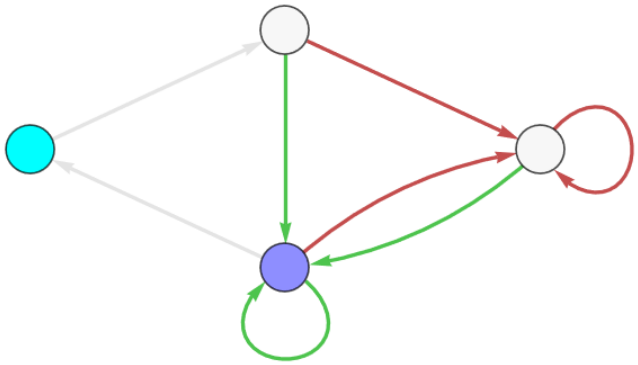


Obrázek 34. Wolfram automat KS1

K výslednému automatu KS1 můžeme zkonstruovat automat VA1, který odpovídá automatu A v příkladu číslo 2.1 a porovnat ekvivalence těchto dvou automatů pomocí funkce EquivalentQFA. Šedé přechody znázorňují epsilon přechody.

```
VA1 = FA[TSys[4, 2, {DirectedEdge[1,2,0], DirectedEdge[2,3,2], DirectedEdge[2,4,1], DirectedEdge[3,1,0], DirectedEdge[3,3,2], DirectedEdge[3,4,1], DirectedEdge[4,4,1], DirectedEdge[4,3,2]}, Alpha[2, <||>], <|"TransType" -> "FAE"|>], <|"AccCond" -> Existential[{1}, {1, 3}]|>];

EquivalentQFA[KS1, VA1]
```

Výstup:



Obrázek 35. Wolfram automat VA1

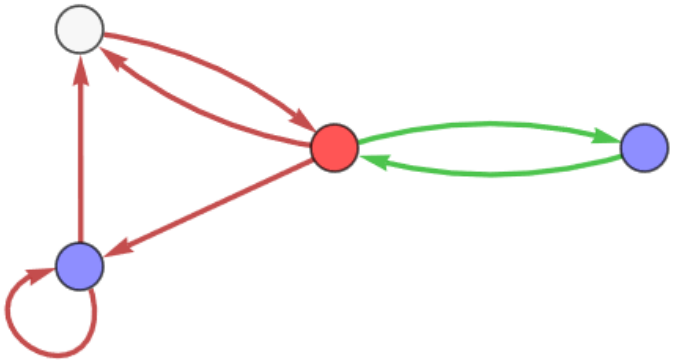
```
True
```

4.1.1.2 Iterace pomocí NKA

U tohoto příkladu vytvoříme NKA KA2 reprezentující NKA A_1 z přílohy 1, opět pomocí funkce FA.

```
KA2 = FA[TSys[4, 2, {DirectedEdge[1,2,1], DirectedEdge[1,3,2], DirectedEdge[1,4,1], DirectedEdge[2,2,1], DirectedEdge[2,4,1], DirectedEdge[3,1,2], DirectedEdge[4,1,1]}, Alpha[2, <||>], <|"TransType" -> "NFA"|>], <|"AccCond" -> Existential[{1}, {2, 3}]|>];
```

Výstup:

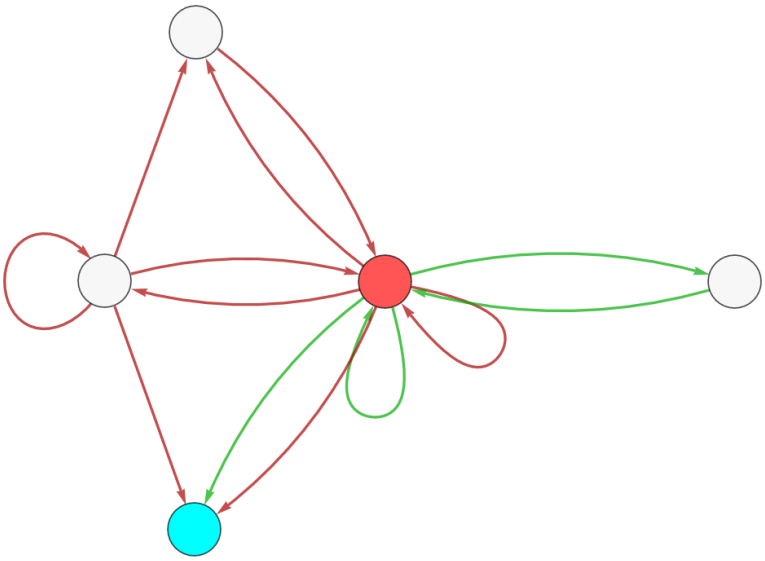


Obrázek 36. Wolfram automat KA2

Následně stejně jako u předešlého příkladu použijeme operaci iterace pomocí funkce KleeneStarFA a dostaneme automat KS2.

```
KS2 = KleeneStarFA[KA2];
```

Výstup:



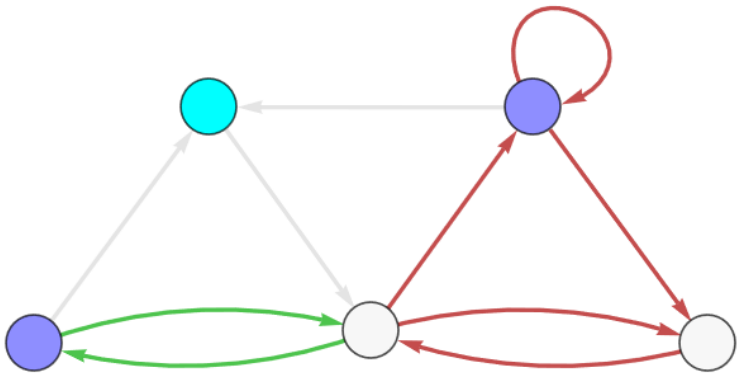
Obrázek 37. Wolfram automat KS2

Nakonec opět vytvoříme konečný automat, který jsme získali v příkladu 1.1 z přílohy 1 a porovnáme ekvivalenci s automatem, který jsme vytvořili pomocí funkce iterace.

```
VA2 = FA[TSys[5, 2, {DirectedEdge[1,2,0], DirectedEdge[2,3,1], DirectedE-
dge[ 2,4,2], DirectedEdge[2,5,1], DirectedEdge[3,3,1], DirectedEdge[3,5,1], Di-
rectedEdge[4,2,2], DirectedEdge[5,2,1], DirectedEdge[3,1,0],
```

```
DirectedEdge[4,1,0]}, Alpha[2, <||>], <|"TransType" -> "FAE"|>], <|"AccCond"
->      Existential[{1}, {1, 3, 4}]|>];

EquivalentQFA[KS2, VA2]
```

Výstup:



Obrázek 38. Wolfram automat VA2

```
True
```

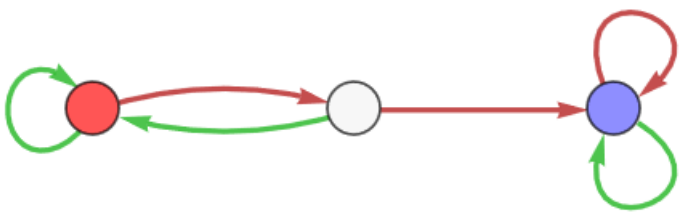
4.1.2 Sjednocení pomocí automatů

4.1.2.1 Sjednocení pomocí dvou DKA

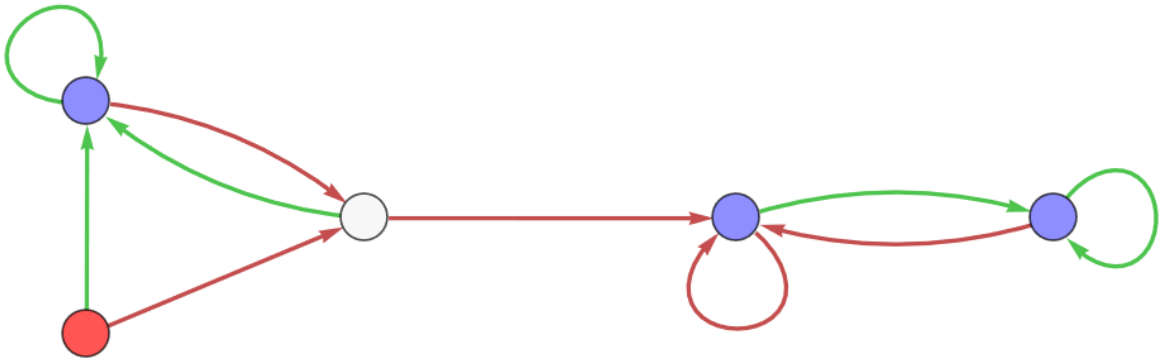
V tomto příkladu první vytvoříme DKA KA3, který reprezentuje automat A_2 z příkladu 2.2, který následně sjednotíme s DKA KA1 a získáme automat U1

```
KA3 = FA[TSys[3, 2, {DirectedEdge[1,2,1], DirectedEdge[1,1,2], DirectedE-
dge[2,3,1], DirectedEdge[2,1,2], DirectedEdge[3,3,1], DirectedEdge[3,3,2]},

 Alpha[2, <||>], <|"TransType" -> "DFA"|>], <|"AccCond" -> Existential[{1},
{3}]|>];

U1 = UnionFA[KA1, KA3];
```

Výstup:



Obrázek 39. Wolfram automat KA3

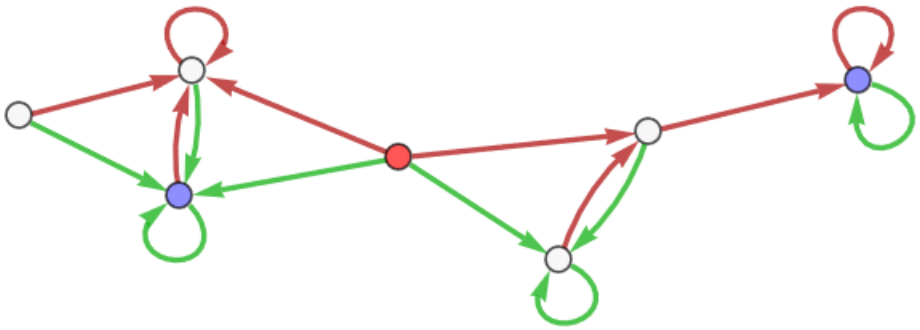


Obrázek 40. Wolfram automat U1

Po sjednocení těchto automatů opět vytvoříme automat A, který jsme získali v příkladu 2.2 a ověříme jejich ekvivalence.

```
VA3 = FA[TSys[7, 2, {DirectedEdge[1,2,2], DirectedEdge[1,3,1], DirectedEdge[2,3,1], DirectedEdge[2,2,2], DirectedEdge[3,4,1], DirectedEdge[3,2,2], DirectedEdge[4,4,1], DirectedEdge[4,4,2], DirectedEdge[1,5,2], DirectedEdge[1,6,1], DirectedEdge[5,5,2], DirectedEdge[5,6,1], DirectedEdge[6,5,2], DirectedEdge[6,6,1], DirectedEdge[7,5,2], DirectedEdge[7,6,1]},
  Alpha[2, <||>], <|"TransType" -> "NFA"|>], <|"AccCond" ->
   Existential[{1}, {4, 5}]|>];
EquivalentQFA[U1, VA3]
```

Výstup:



Obrázek 41. Wolfram automat VA3

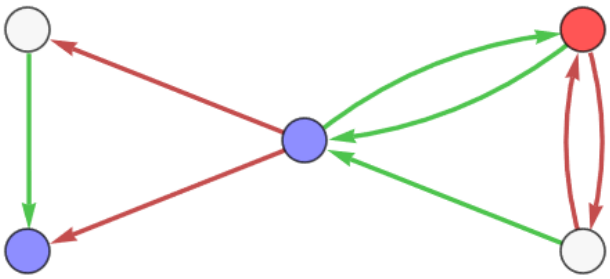
```
True
```

4.1.2.2 Sjednocení pomocí dvou NKA

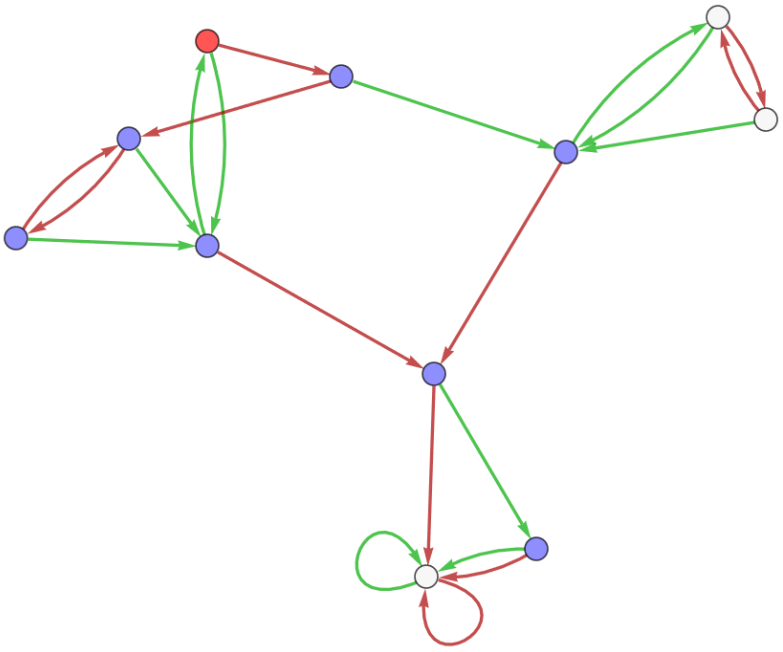
Zde vytvoříme NKA KA4, který odpovídá automatu A_2 z přílohy 1 a následně ho sjednotíme s automatem NKA KA2. Je nutné před použitím funkce UnionFA konečné automaty determinizovat.

```
KA4 = FA[TSys[5, 2, {DirectedEdge[1,2,1], DirectedEdge[1,3,2], DirectedEdge[2,1,1], DirectedEdge[2,3,2], DirectedEdge[3,1,2], DirectedEdge[3,4,1], DirectedEdge[3,5,1], DirectedEdge[4,5,2]}, Alpha[2, <||>], <|"TransType" -> "NFA"|>], <|"AccCond" -> Existential[{1}, {3, 5}]|>];

U2 = UnionFA[DeterminizeFA[KA2], DeterminizeFA[KA4]];
```

Výstup:



Obrázek 42. Wolfram automat KA4

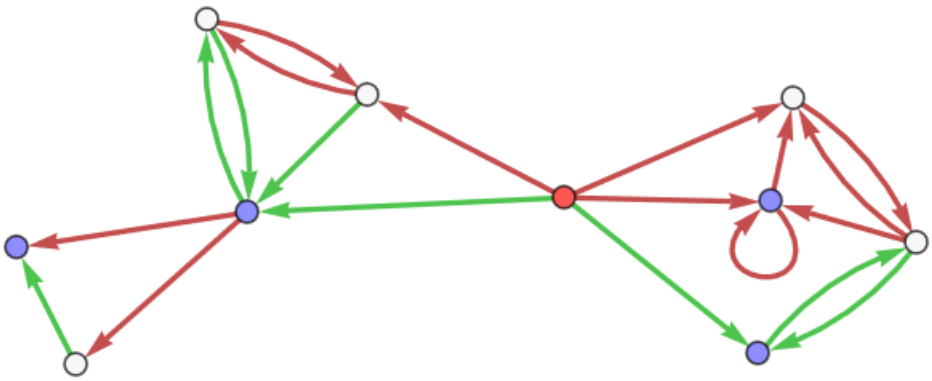


Obrázek 43. Wolfram automat U2

Opět provedeme kontrolu ekvivalence s automatem A z příkladu 1.2 z přílohy 1.

```
VA4 = FA[TSys[10, 2, {DirectedEdge[1,2,1], DirectedEdge[1,3,2], DirectedEdge[1,4,1], DirectedEdge[5,2,1], DirectedEdge[5,3,2], DirectedEdge[5,4,1], DirectedEdge[2,2,1], DirectedEdge[2,4,1], DirectedEdge[3,5,2], DirectedEdge[4,5,1], DirectedEdge[1,6,1], DirectedEdge[1,7,2], DirectedEdge[6,8,1], DirectedEdge[6,7,2], DirectedEdge[7,8,2], DirectedEdge[8,6,1], DirectedEdge[8,7,2], DirectedEdge[7,9,1], DirectedEdge[7,10,1], DirectedEdge[9,10,2]},
  Alpha[2, <||>], <|"TransType" -> "NFA"|>], <|"AccCond" -> Existential[{1}, {2, 3, 7, 10}]|>];
EquivalentQFA[U2, VA4]
```

Výstup:



Obrázek 44. Wolfram automat VA4

```
True
```

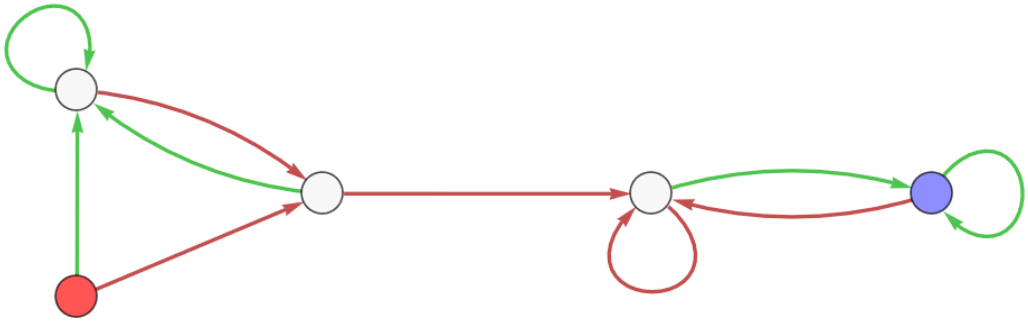
4.1.3 Průnik pomocí automatů

4.1.3.1 Průnik pomocí dvou DKA

K tomuto příkladu máme již vytvořeny KA1 z příkladu 2.1 i KA3 z příkladu 2.2 a tak můžeme rovnou provést operaci průniku obou automatů pomocí funkce IntersectinFA.

```
I1 = IntersectionFA[KA1, KA3];
```

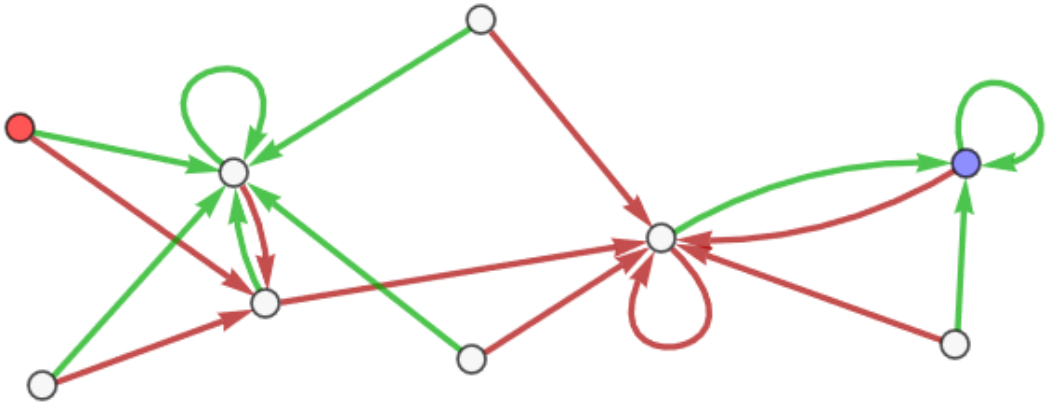
Výstup:



Obrázek 45. Wolfram automat I1

Stejně jako v předešlých příkladech i zde ověříme ekvivalenci s konečným automatem A z příkladu číslo 2.3

```
VA5 = FA[TSys[9, 2, {DirectedEdge[1,2,2], DirectedEdge[1,3,1], DirectedEdge[ 2,3,1], DirectedEdge[2,2,2], DirectedEdge[3,4,1], DirectedEdge[3,2,2], DirectedEdge[4,4,1], DirectedEdge[4,5,2], DirectedEdge[5,5,2], DirectedEdge[5,4,1], DirectedEdge[6,5,2], DirectedEdge[6,4,1], DirectedEdge[7,2,2], DirectedEdge[7,3,1], DirectedEdge[8,2,2], DirectedEdge[8,4,1], DirectedEdge[9,2,2], DirectedEdge[9,4,1]}, Alpha[2, <||>], <|"TransType" -> "NFA"|>], <|"AccCond" -> Existential[{1}, {5}]|>];

EquivalentQFA[I1, VA5]
```



Obrázek 46. Wolfram automat VA5

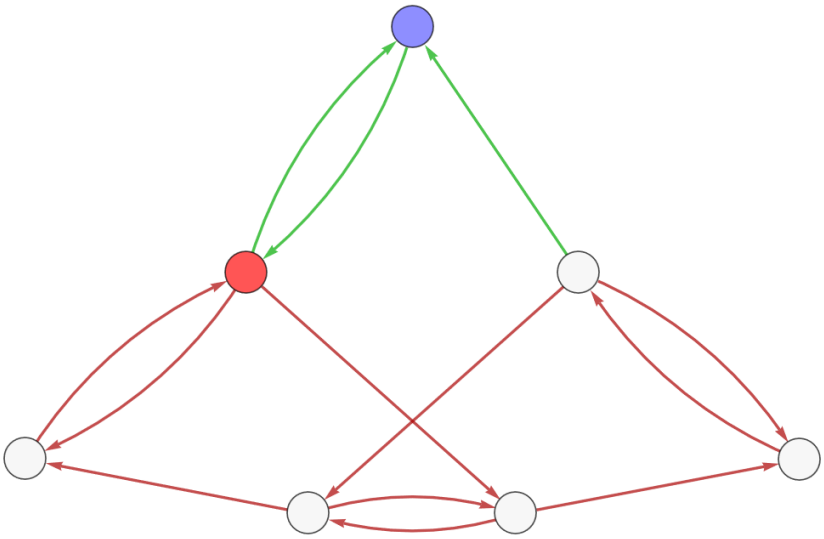
```
True
```

4.1.3.2 Průnik pomocí dvou NKA

Zde provedeme průnik dvou NKA. KA2 z příkladu číslo 1.1 z přílohy 1 a KA4 z příkladu 1.2 z téže přílohy. Tímto získáme automat I2

```
I2 = IntersectionFA[KA2, KA4];
```

Výstup:



Obrázek 47. Wolfram automat I2

Po vytvoření automatu I2 vytvoříme automat VA6, který jsme získali z příkladu 1.3 v příloze 1.

```
VA6 = FA[TSys[7, 2, {DirectedEdge[1,2,1], DirectedEdge[1,3,1], DirectedEdge[1,4,2], DirectedEdge[2,5,1], DirectedEdge[2,6,1], DirectedEdge[3,1,1], DirectedEdge[4,1,2], DirectedEdge[5,2,1], DirectedEdge[5,3,1], DirectedEdge[6,7,1], DirectedEdge[7,5,1], DirectedEdge[7,6,1], DirectedEdge[7,4,2]},
  Alpha[2, <||>], <|"TransType" -> "NFA"|>], <|"AccCond" -> Existential[{1}, {4}]|>];
EquivalentQFA[I2, VA6]
```

Výstup:

Výstupem je identický automat jako automat I2 a potvrzení ekvivalence daných automatů.

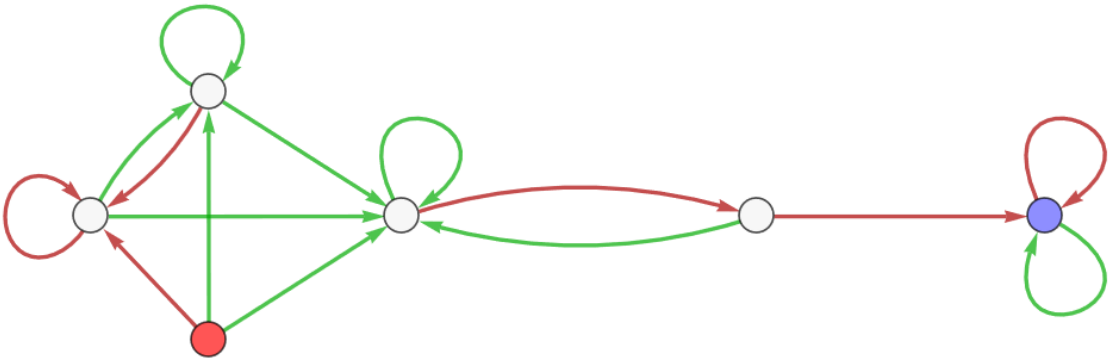
4.1.4 Zřetězení pomocí automatů

4.1.4.1 Zřetězení pomocí dvou DKA

Zřetězení DKA KA1 a KA2 můžeme provést pomocí funkce ConcatenateFA

```
C1 = ConcatenateFA[KA1, KA3];
```

Výstup:



Obrázek 48. Wolfram automat C1

Následně opět sestavíme konečný automat A z příkladu 2.4.

```
VA7 = FA[TSys[6, 2, {DirectedEdge[1,2,2], DirectedEdge[1,3,1], DirectedEdge[2,3,1], DirectedEdge[2,2,2], DirectedEdge[3,3,1], DirectedEdge[3,2,2], DirectedEdge[1,4,2], DirectedEdge[2,4,2], DirectedEdge[ 3,4,2], DirectedEdge[4,4,2], DirectedEdge[4,5,1], DirectedEdge[ 5,4,2], DirectedEdge[5,6,1], DirectedEdge[6,6,1], DirectedEdge[ 6,6,2]}, Alpha[2, <||>], <|"TransType" -> "NFA"|>], <|"AccCond" -> Existential[{1}, {6}]|>];

EquivalentQFA[C2, VA7]
```

Výstup:

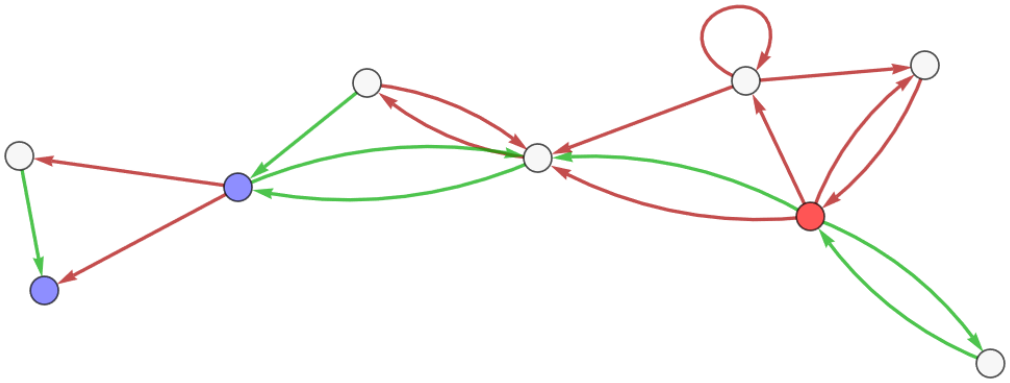
Opět je výstupem identický automat jako automat C1 a potvrzení ekvivalence daných automatů.

4.1.4.2 Zřetězení pomocí dvou DKA

Zřetězení NKA KA2 a KA4 opět provedeme pomocí funkce ConcatenateFA.

```
C2 = ConcatenateFA[KA2, KA4];
```

Výstup:



Obrázek 49. Wolfram automat C2

Znovu sestavíme konečný automat A z příkladu 1.4 z přílohy 1 a ověříme ekvivalenci s automatem C2.

```
VA8 = FA[TSys[9, 2, {DirectedEdge[1,2,1], DirectedEdge[1,3,2], DirectedEdge[1,4,1], DirectedEdge[2,2,1], DirectedEdge[2,4,1], DirectedEdge[3,1,2], DirectedEdge[4,1,1], DirectedEdge[1,5,1], DirectedEdge[2,5,1], DirectedEdge[1,5,2], DirectedEdge[5,6,1], DirectedEdge[5,7,2], DirectedEdge[6,5,1], DirectedEdge[6,7,2], DirectedEdge[7,5,2], DirectedEdge[7,8,1], DirectedEdge[7,9,1], DirectedEdge[ 8,9,2]}, Alpha[2, <||>], <|"TransType" -> "NFA"|>], <|"AccCond" -> Existential[{1}, {7, 9}]|>];

EquivalentQFA[C2, VA8]
```

Výstup:

Výstupem je opět identický automat jako je automat C2 a potvrzení ekvivalence automatů.

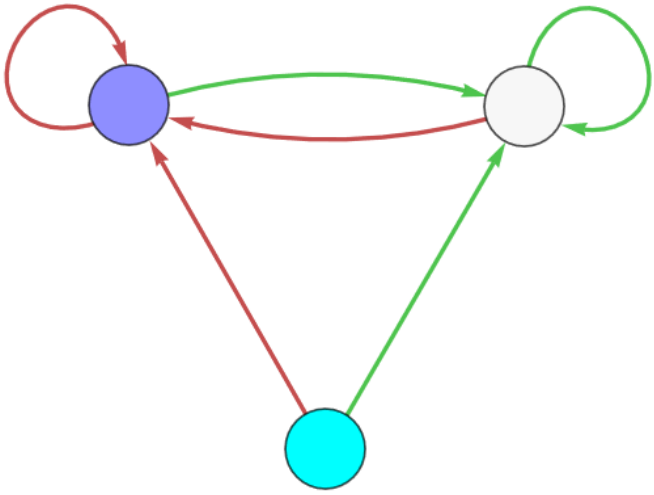
4.1.5 Doplněk pomocí automatů

4.1.5.1 Doplněk pomocí DKA

Doplněk DKA KA1 můžeme provést díky funkci ComplementFA

```
CO1 = ComplementFA[KA1];
```

Výstup:



Obrázek 50. Wolfram automat CO1

Po získání doplňku z funkce ComplementFA vytvoříme automat VA9, který jsme získali v příkladu 2.5.

```
VA9 = FA[TSys[3, 2, {DirectedEdge[1,2,2], DirectedEdge[1,3,1], DirectedE-
dge[2,2,2], DirectedEdge[2,3,1], DirectedEdge[3,2,2], DirectedEdge[
  3,3,1]}, Alpha[2, <||>], <|"TransType" -> "DFA"|>], <|"AccCond" ->
  Existential[{1}, {1, 3}]|>];
EquivalentQFA[CO1, VA9]
```

Výstup:

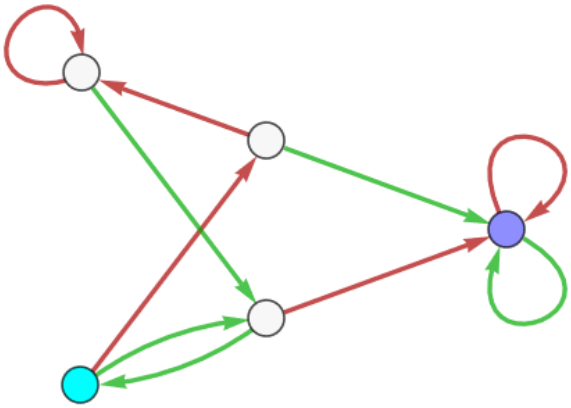
Výstupem je opět identický automat jako CO1 a potvrzení ekvivalence automatů.

4.1.5.2 Doplněk pomocí NKA

Opět použijeme funkci ComplementFA a oproti metodě z teorie zde můžeme do funkce vložit NKA a nemusíme ho determinizovat.

```
CO2 = ComplementFA[KA2];
```

Výstup:



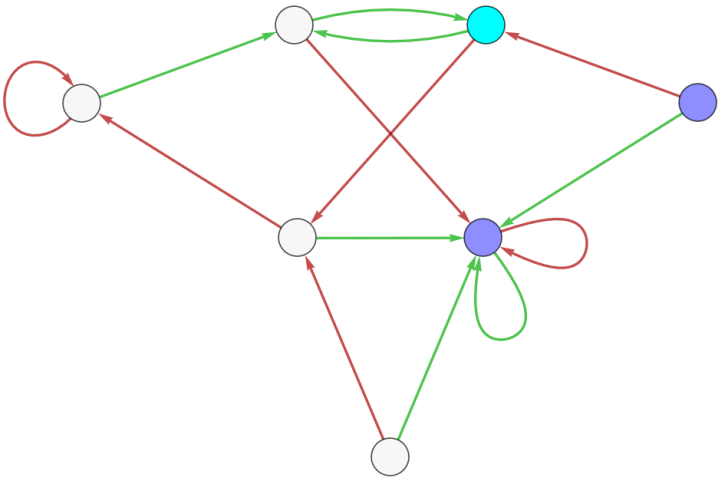
Obrázek 51. Wolfram automat CO2

```
True
```

Po získání automatu CO2 vytvoříme automat VA10, který je shodný s automatem A z příkladu 1.5 z přílohy1.

```
VA10 = FA[TSys[4, 2, {DirectedEdge[1,2,1], DirectedEdge[1,3,2], DirectedEdge[1,4,1], DirectedEdge[2,2,1], DirectedEdge[2,4,1], DirectedEdge[3,1,2], DirectedEdge[4,1,1]}, Alpha[2, <||>], <|"TransType" -> "NFA"|>], <|"AccCond" -> Existential[{1}, {1, 4}]|>];

EquivalentQFA[CO2, VA10]
```

Výstup:



Obrázek 52. Wolfram automat VA10

```
True
```

4.2 Převody

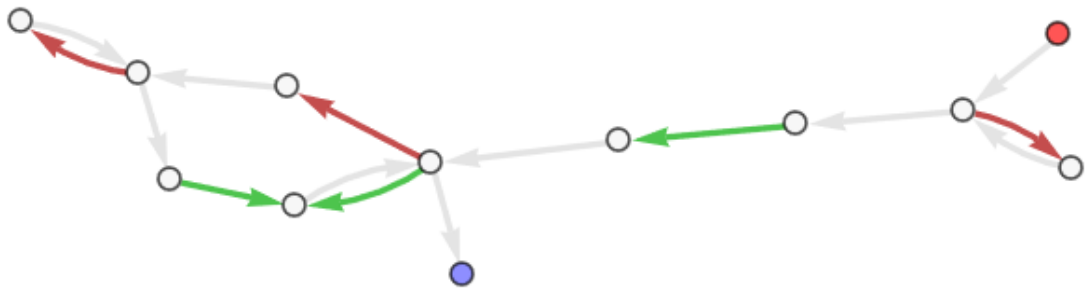
4.2.1 Převod regulárního výrazu na konečný automat

4.2.1.1 Thompsonův algoritmus

První vytvoříme regulární výraz RE1 z příkladu 3.1 pomocí funkcí RES, která zastává iteraci, RET, která zastává zřetězení a funkce REP, která představuje sjednocení. Následně převedeme regulární výraz na konečný automat pomocí funkce RegexToFA, která využívá Thompsonova algoritmu.

```
RE1 = RET[RES["a"], "b", RES[REP["b", RET["a", RES["a"], "b"]]]];

RKA1 = RegexToFA[RE1, Alpha[2]];
```

Výstup:

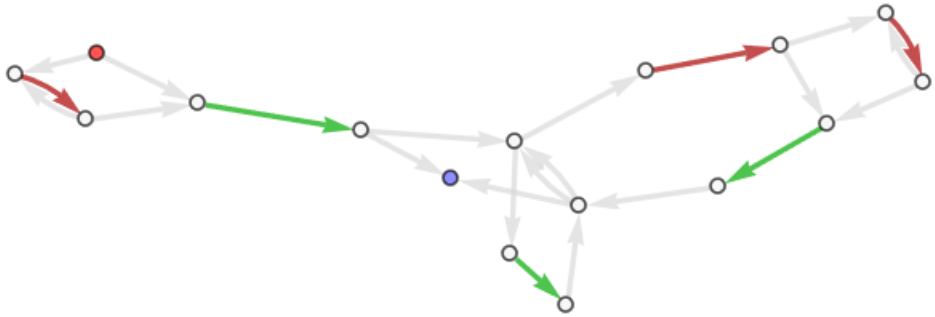


Obrázek 53. Wolfram automat RKA1

Následně vytvoříme výsledný konečný automat z příkladu 3.1 a porovnáme ho s automatem z funkce RegexToFA.

```
VA11 = FA[TSys[16, 2, {DirectedEdge[1,2,1], DirectedEdge[2,1,0], DirectedEdge[3,1,0], DirectedEdge[3,4,0], DirectedEdge[2,4,0], DirectedEdge[ 4,5,2], DirectedEdge[6,7,2], DirectedEdge[8,9,1], DirectedEdge[9,10,0], DirectedEdge[10,11,1], DirectedEdge[11,10,0], DirectedEdge[11,12,0], DirectedEdge[9,12,0], DirectedEdge[12,13,2], DirectedEdge[5,14,0], DirectedEdge[7,15,0], DirectedEdge[13,15,0], DirectedEdge[15,14,0], DirectedEdge[14,6,0], DirectedEdge[14,8,0], DirectedEdge[15,14,0], DirectedEdge[5,16,0], DirectedEdge[15,16,0]}, Alpha[2, <||>], <|"TransType" -> "FAE"|>], <|"AccCond" -> Existential[{3}, {16}]|>];

EquivalentQFA[RKA1, VA11]
```

Výstup:



Obrázek 54. Wolfram automat VA11

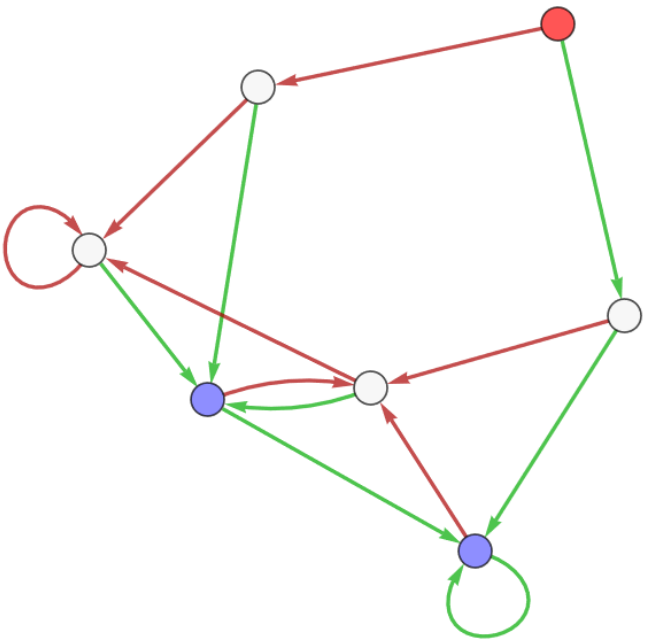
```
True
```

4.2.1.2 Glushkův algoritmus

Zde opět převedeme regulární výraz z příkladu 3.1 na konečný automat, tentokrát pomocí funkce RegexToGlushkov.

```
RKA2 = RegexToGlushkov[RE1, Alpha[2]];
```

Výstup:

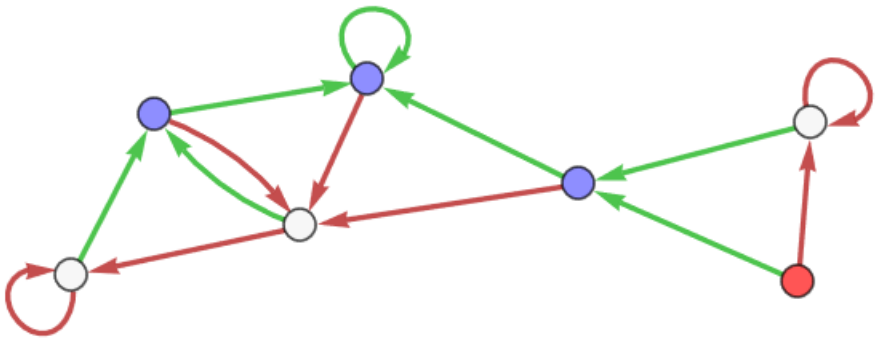


Obrázek 55. Wolfram automat RKA2

Následně zkontrolujeme výsledný konečný automat s automatem, který jsme získali v příkladu 3.2

```
VA12 = FA[TSys[7, 2, {DirectedEdge[1,2,1], DirectedEdge[1,3,2], DirectedEdge[2,2,1], DirectedEdge[2,3,2], DirectedEdge[3,4,2], DirectedEdge[3,5,1], DirectedEdge[4,5,1], DirectedEdge[4,4,2], DirectedEdge[5,6,1], DirectedEdge[5,7,2], DirectedEdge[6,6,1], DirectedEdge[6,7,2], DirectedEdge[7,5,1], DirectedEdge[7,4,2]}, Alpha[2, <||>], <|"TransType" -> "DFA"|>], <|"AccCond" -> Existential[{1}, {3, 4, 7}]|>];

EquivalentQFA[RKA2, VA12]
```

Výstup:



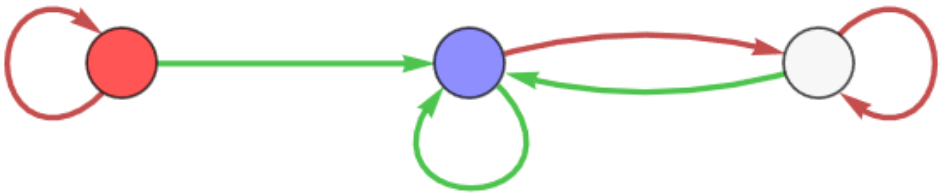
Obrázek 56. Wolfram automat VA12

```
False
```

Bohužel zde vidíme, že automaty nejsou ekvivalentní, a proto je porovnáme s ostatními vypracovanými automaty. Vytvoříme konečný automat VA13, který jsme získali v příkladu 3.3. Po zkontrolování výstupů můžeme vidět, že funkce RegexToGlushkov v tomto balíčku nepřevedla náš zadaný regulární výraz správně

```
VA13 = FA[TSys[3, 2, {DirectedEdge[1,1,1], DirectedEdge[1,2,2], DirectedEdge[2,3,1], DirectedEdge[2,2,2], DirectedEdge[3,3,1], DirectedEdge[3,2,2]}, Alpha[2, <||>], <|"TransType" -> "DFA"|>], <|"AccCond" -> Existential[{1}, {2}]|>];

EquivalentQFA[RKA1, VA12]

EquivalentQFA[RKA1, VA13]

EquivalentQFA[RKA1, RKA2]
```

Výstup:



Obrázek 57. Wolfram automat VA13

```
True, True, False
```

4.2.2 Převod DKA na Monoid

První vytvoříme DKA KA6 z příkladu 3.6.

```
KA6 = FA[TSys[3, 2, {DirectedEdge[1,2,1], DirectedEdge[1,1,2], DirectedE-
dge[2,1,1], DirectedEdge[2,3,2], DirectedEdge[3,2,1], DirectedEdge[
  3,1,2]}, Alpha[2, <||>], <|"TransType" -> "DFA"|>], <|"AccCond" ->
  Existential[{1}, {3}]|>];
```

Následně ho převedeme pomocí funkce SemigroupGenerate na pologrupu a pomocí funkce SemigroupToMonoid převedeme pologrupu na monoid. Po převedení vypíšeme tabulku 11 z příkladu 3.6 pomocí druhého příkazu a zjednodušená přepisovací pravidla pomocí třetího.

```
M1 = SemigroupToMonoid[SemigroupGenerate[KA6]];
Pairs[ SemigroupWitnesses[M1], SemigroupElements[M1]] // Grid
RewriteRulesSimplify@SemigroupEquations[M1] // Column
```

Výstup:

	SGT[{1, 2, 3}]	Přepisovací pravidla:
a	SGT[{2, 1, 2}]	aaa → a
b	SGT[{1, 3, 1}]	aab → b
aa	SGT[{1, 2, 1}]	aba → ba
ab	SGT[{3, 1, 3}]	abb → bb
ba	SGT[{2, 2, 2}]	baa → bb
bb	SGT[{1, 1, 1}]	bba → ba
bab	SGT[{3, 3, 3}]	bbb → bb

5 PYTHON

V této kapitole si ukážeme, jak můžeme provést v Pythonu některé operace nebo převody z předešlých kapitol. Použijeme k tomu balíček fado a pro vizualizaci konečných automatů je použit balíček graphviz.

5.1 Operace

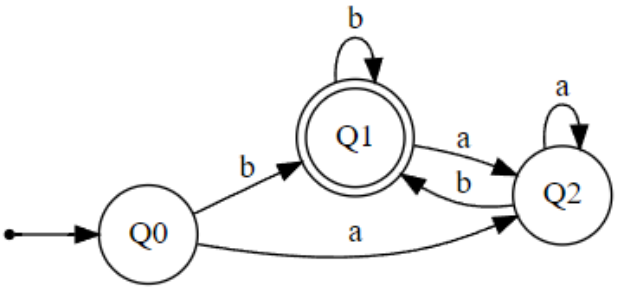
5.1.1 Iterace pomocí automatů

5.1.1.1 Iterace pomocí DKA

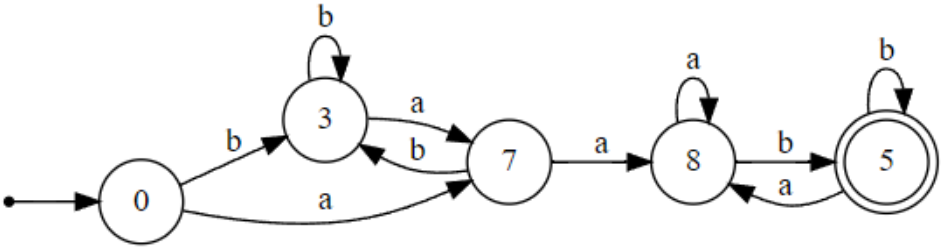
Nejprve vytvoříme DKA KA1, který odpovídá automatu A_1 z příkladu 2.1 v teoretické části a následně na něj aplikujeme operaci iterace pomocí funkce star.

```
KA1 = DFA()
KA1.setSigma(['a','b'])
KA1.addState('Q0')
KA1.addState('Q1')
KA1.addState('Q2')
KA1.setInitial(0)
KA1.addFinal(1)
KA1.addTransition(0, 'b', 1)
KA1.addTransition(0, 'a', 2)
KA1.addTransition(1, 'b', 1)
KA1.addTransition(1, 'a', 2)
KA1.addTransition(2, 'b', 1)
KA1.addTransition(2, 'a', 2)
KA1.display()
KS1 = KA1.star()
KS1.display()
```

Výstup:



Obrázek 58. Python automat KA1



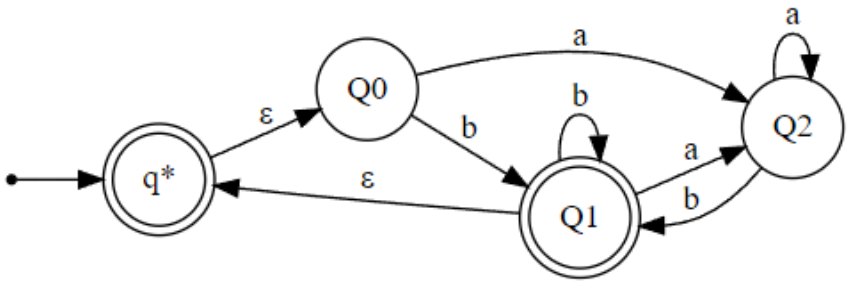
Obrázek 59. Python automat KS1

Následně vytvoříme konečný automat A z příkladu 2.1 a ověříme, zdali jsou automaty ekvivalentní pomocí funkce equivalentP.

```
VA1 = NFA()
VA1.setSigma(['a','b'])
VA1.addState('q*')
VA1.addState('Q0')
VA1.addState('Q1')
VA1.addState('Q2')
VA1.addInitial(0)
VA1.addFinal(0)
VA1.addFinal(2)
VA1.addTransition(0, '@epsilon', 1)
VA1.addTransition(1, 'b', 2)
VA1.addTransition(1, 'a', 3)
VA1.addTransition(2, '@epsilon', 0)
VA1.addTransition(2, 'b', 2)
VA1.addTransition(2, 'a', 3)
```

```
VA1.addTransition(3, 'a', 3)
VA1.addTransition(3, 'b', 2)
VA1.display()
KS1.equivalentP(VA1)
```

Výstup:



Obrázek 60. Python automat VA1

```
True
```

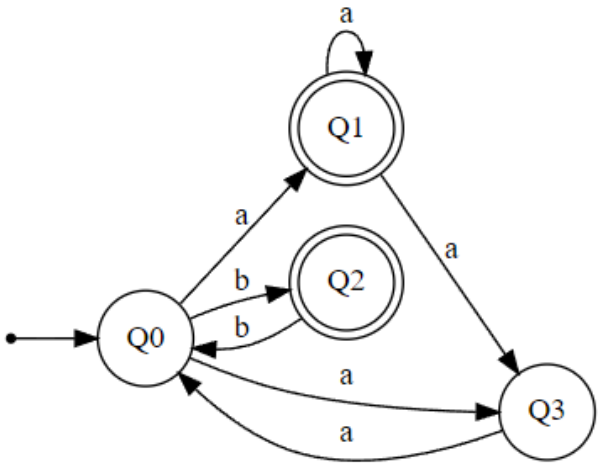
5.1.1.2 Iterace pomocí NKA

Vytvoříme NKA KA2, který je v příkladu 1.1 v příloze 1 automat A_1 a následně na něj aplikujeme operaci iterace pomocí funkce star stejně jako u DKA.

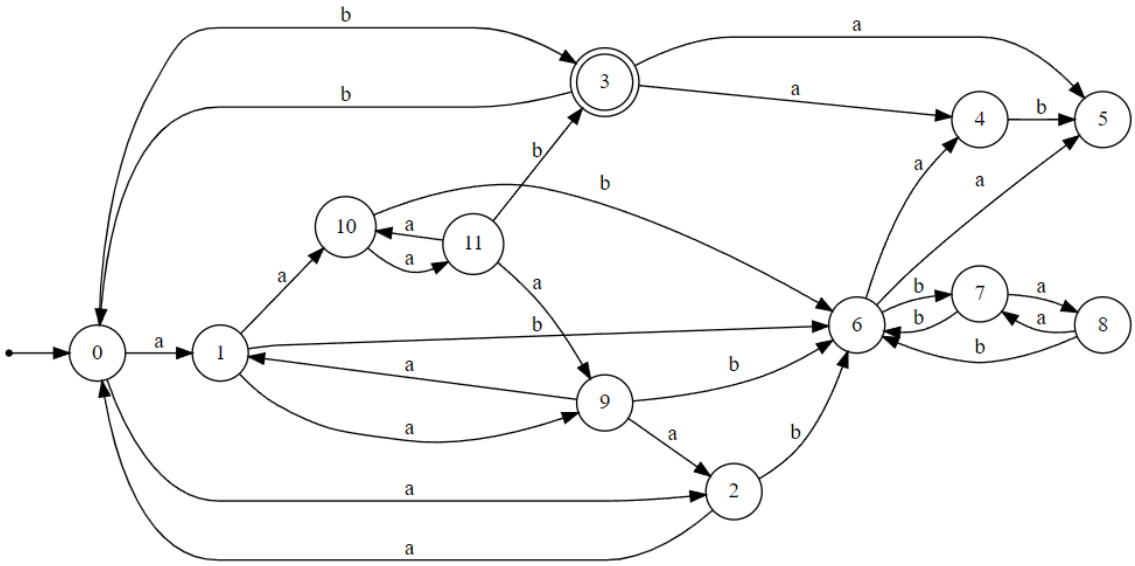
```
KA2 = NFA()
KA2.setSigma(['a','b'])
KA2.addState('Q0')
KA2.addState('Q1')
KA2.addState('Q2')
KA2.addState('Q3')
KA2.addInitial(0)
KA2.addFinal(1)
KA2.addFinal(2)
KA2.addTransition(0, 'a', 1)
KA2.addTransition(0, 'b', 2)
KA2.addTransition(0, 'a', 3)
KA2.addTransition(1, 'a', 1)
KA2.addTransition(1, 'a', 3)
KA2.addTransition(2, 'b', 0)
```

```
KA2.addTransition(3, 'a', 0)
KA2.display()
KS2 = KA2.star()
KS2.display()
```

Výstup:



Obrázek 61. Python automat KA2



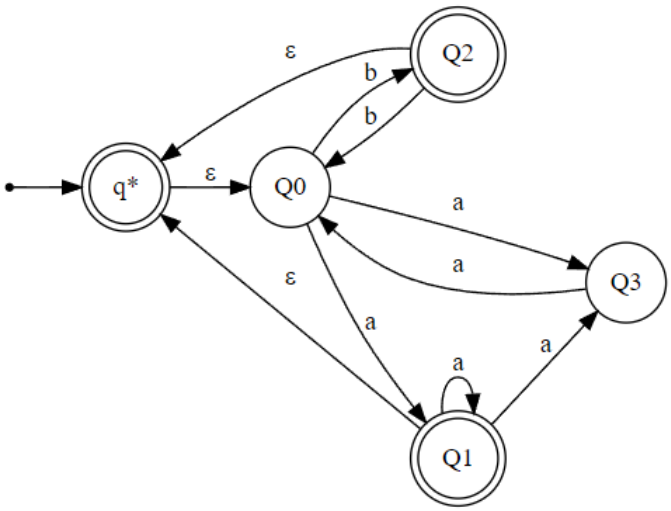
Obrázek 62. Python automat KS2

Opět vytvoříme konečný automat VA2, který je v příkladu 1.1 v příloze 1 značen A a porovnáme ekvivalence s automatem získaným pomocí funkce star.

```
VA2 = NFA()
VA2.setSigma(['a','b'])
VA2.addState('q*')
```

```
VA2.addState('Q0')
VA2.addState('Q1')
VA2.addState('Q2')
VA2.addState('Q3')
VA2.addInitial(0)
VA2.addFinal(0)
VA2.addFinal(2)
VA2.addFinal(3)
VA2.addTransition(0, '@epsilon', 1)
VA2.addTransition(1, 'a', 2)
VA2.addTransition(1, 'b', 3)
VA2.addTransition(1, 'a', 4)
VA2.addTransition(2, 'a', 2)
VA2.addTransition(2, 'a', 4)
VA2.addTransition(3, 'b', 1)
VA2.addTransition(4, 'a', 1)
VA2.addTransition(2, '@epsilon', 0)
VA2.addTransition(3, '@epsilon', 0)
VA2.display()
KS2.equivalentP(VA2)
```

Výstup:



Obrázek 63. Python automat VA2

```
True
```

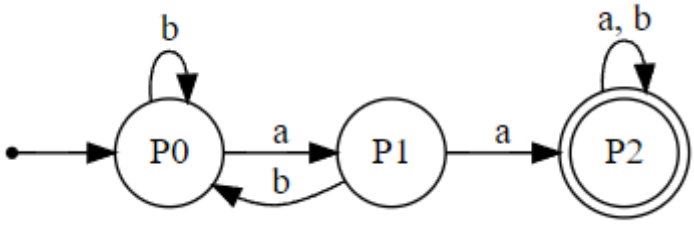
5.1.2 Sjednocení pomocí automatů

5.1.2.1 Sjednocení pomocí DKA

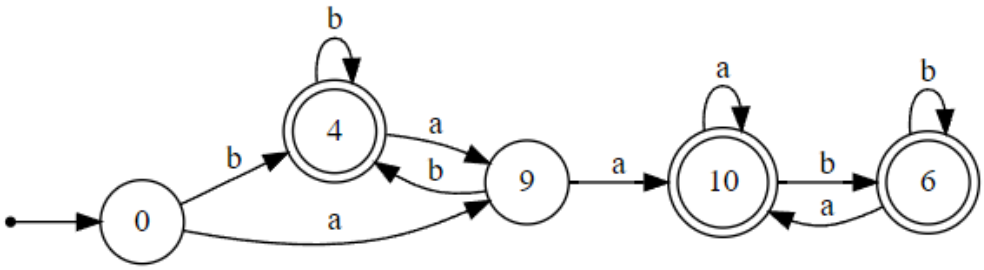
Zde vytvoříme DKA KA3 z příkladu 2.1, kde je zapsán jako automat A_2 a následně ho sjednotíme pomocí funkce union s KA1 vytvořeného v podkapitole iterace DKA

```
KA3 = DFA()
KA3.setSigma(['a','b'])
KA3.addState('P0')
KA3.addState('P1')
KA3.addState('P2')
KA3.setInitial(0)
KA3.addFinal(2)
KA3.addTransition(0, 'a', 1)
KA3.addTransition(0, 'b', 0)
KA3.addTransition(1, 'a', 2)
KA3.addTransition(1, 'b', 0)
KA3.addTransition(2, 'a', 2)
KA3.addTransition(2, 'b', 2)
KA3.display()
U1 = KA1.union(KA3)
U1.display()
```

Výstup:



Obrázek 64. Python automat KA3



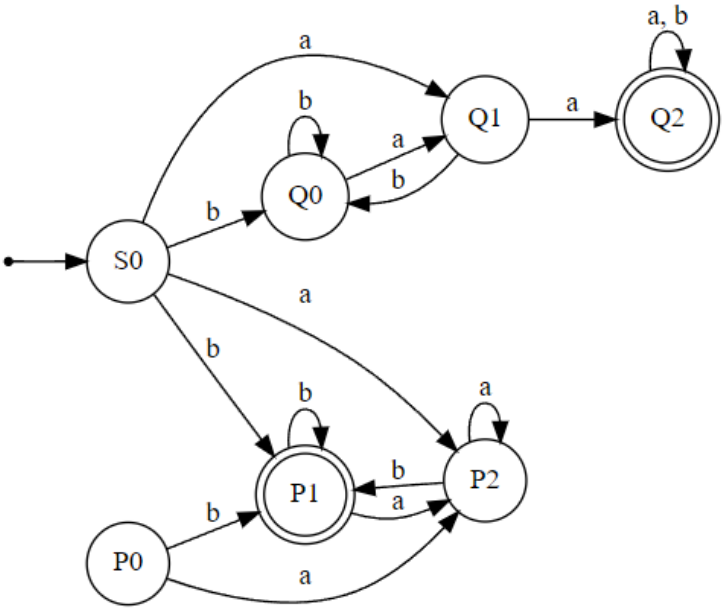
Obrázek 65. Python automat U1

Dále opět vytvoříme konečný automat A z příkladu 2.2 a porovnáme ho s automatem získaným pomocí operace union.

```
VA3 = NFA()
VA3.setSigma(['a','b'])
VA3.addState('S0')
VA3.addState('Q0')
VA3.addState('Q1')
VA3.addState('Q2')
VA3.addState('P1')
VA3.addState('P2')
VA3.addState('P0')
VA3.addInitial(0)
VA3.addFinal(3)
VA3.addFinal(4)
VA3.addTransition(0, 'b', 1)
VA3.addTransition(0, 'a', 2)
VA3.addTransition(1, 'a', 2)
VA3.addTransition(1, 'b', 1)
VA3.addTransition(2, 'a', 3)
VA3.addTransition(2, 'b', 1)
VA3.addTransition(3, 'a', 3)
VA3.addTransition(3, 'b', 3)
VA3.addTransition(0, 'b', 4)
VA3.addTransition(0, 'a', 5)
VA3.addTransition(4, 'b', 4)
VA3.addTransition(4, 'a', 5)
```

```
VA3.addTransition(5, 'b', 4)
VA3.addTransition(5, 'a', 5)
VA3.addTransition(6, 'b', 4)
VA3.addTransition(6, 'a', 5)
VA3.display()
U1.equivalentP(VA3)
```

Výstup:



Obrázek 66. Python automat VA3

```
True
```

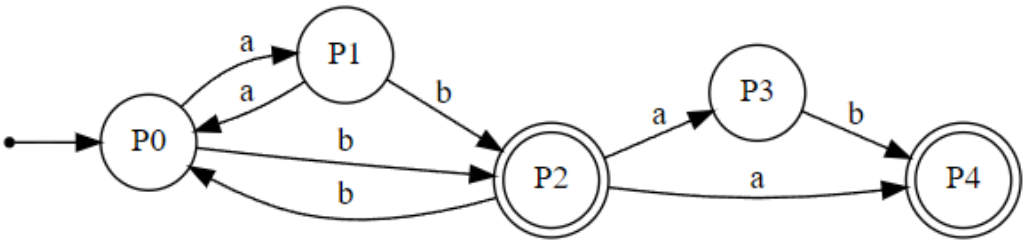
5.1.2.2 Sjednocení pomocí NKA

Sjednotíme NKA KA2 z příkladu 1.2 z přílohy 1 s nově vytvořeným NKA KA4 značený v tom samém příkladu A_2 pomocí funkce union.

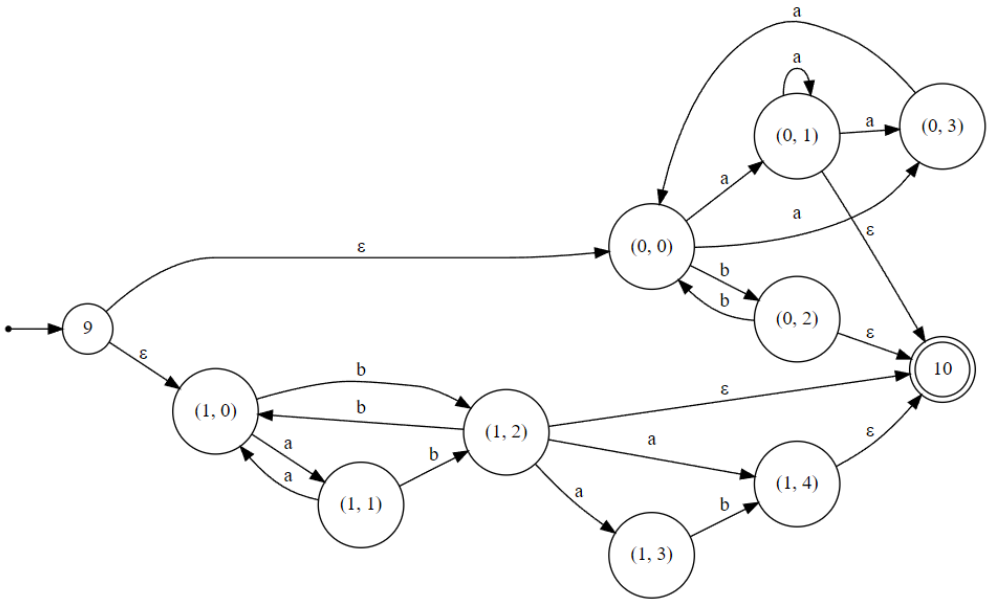
```
KA4 = NFA()
KA4.setSigma(['a','b'])
KA4.addState('P0')
KA4.addState('P1')
KA4.addState('P2')
KA4.addState('P3')
KA4.addState('P4')
KA4.addInitial(0)
```

```
KA4.addFinal(2)
KA4.addFinal(4)
KA4.addTransition(0, 'a', 1)
KA4.addTransition(0, 'b', 2)
KA4.addTransition(1, 'a', 0)
KA4.addTransition(1, 'b', 2)
KA4.addTransition(2, 'b', 0)
KA4.addTransition(2, 'a', 3)
KA4.addTransition(2, 'a', 4)
KA4.addTransition(3, 'b', 4)
KA4.display()
U2 = KA2.union(KA4)
U2.display()
```

Výstup:



Obrázek 67. Python automat KA4



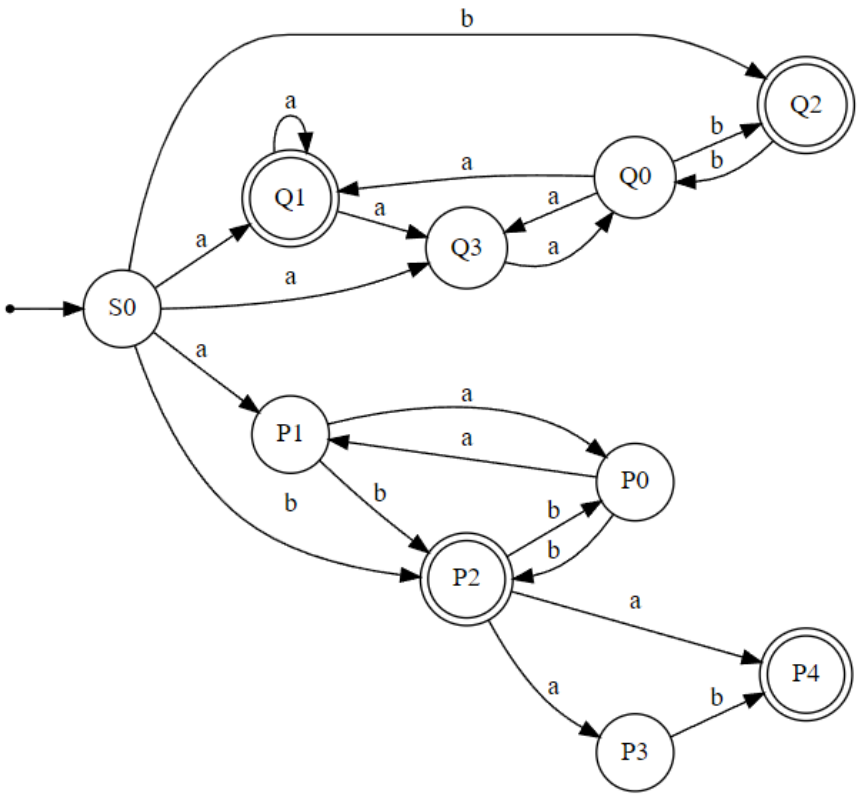
Obrázek 68. Python automat U2

Následně sestavíme automat A z příkladu 1.2 z přílohy 1 a porovnáme ekvivalence s automatem získaným pomocí funkce union.

```
VA4 = NFA()
VA4.setSigma(['a','b'])
VA4.addState('S0')
VA4.addState('Q1')
VA4.addState('Q2')
VA4.addState('Q3')
VA4.addState('Q0')
VA4.addState('P1')
VA4.addState('P2')
VA4.addState('P0')
VA4.addState('P3')
VA4.addState('P4')
VA4.addInitial(0)
VA4.addFinal(1)
VA4.addFinal(2)
VA4.addFinal(6)
VA4.addFinal(9)
VA4.addTransition(0, 'a', 1)
VA4.addTransition(0, 'b', 2)
VA4.addTransition(0, 'a', 3)
VA4.addTransition(1, 'a', 1)
VA4.addTransition(1, 'a', 3)
VA4.addTransition(2, 'b', 4)
VA4.addTransition(3, 'a', 4)
VA4.addTransition(4, 'a', 1)
VA4.addTransition(4, 'b', 2)
VA4.addTransition(4, 'a', 3)
VA4.addTransition(0, 'a', 5)
VA4.addTransition(0, 'b', 6)
VA4.addTransition(5, 'a', 7)
VA4.addTransition(5, 'b', 6)
```

```
VA4.addTransition(6, 'b', 7)
VA4.addTransition(7, 'a', 5)
VA4.addTransition(7, 'b', 6)
VA4.addTransition(6, 'a', 8)
VA4.addTransition(6, 'a', 9)
VA4.addTransition(8, 'b', 9)
VA4.display()
U2.equivalentP(VA4)
```

Výstup:



Obrázek 69. Python automat VA4

```
True
```

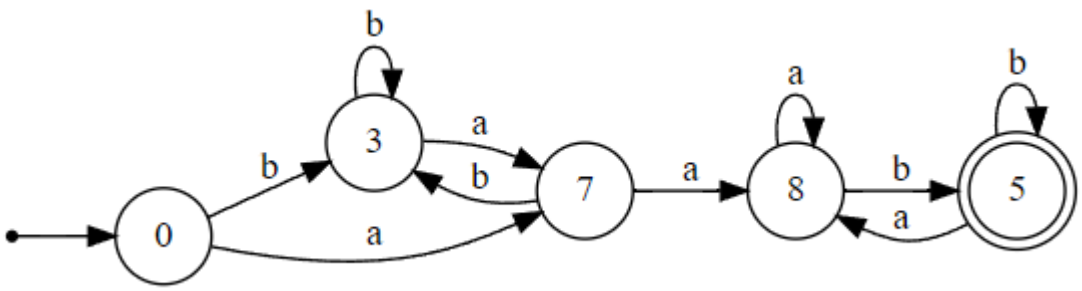
5.1.3 Průnik pomocí automatů

5.1.3.1 Průnik pomocí DKA

Pomocí operace conjunction provedeme průnik DKA KA1 a KA3.

```
I1 = KA1.conjunction(KA3)
I1.display()
```

Výstup:



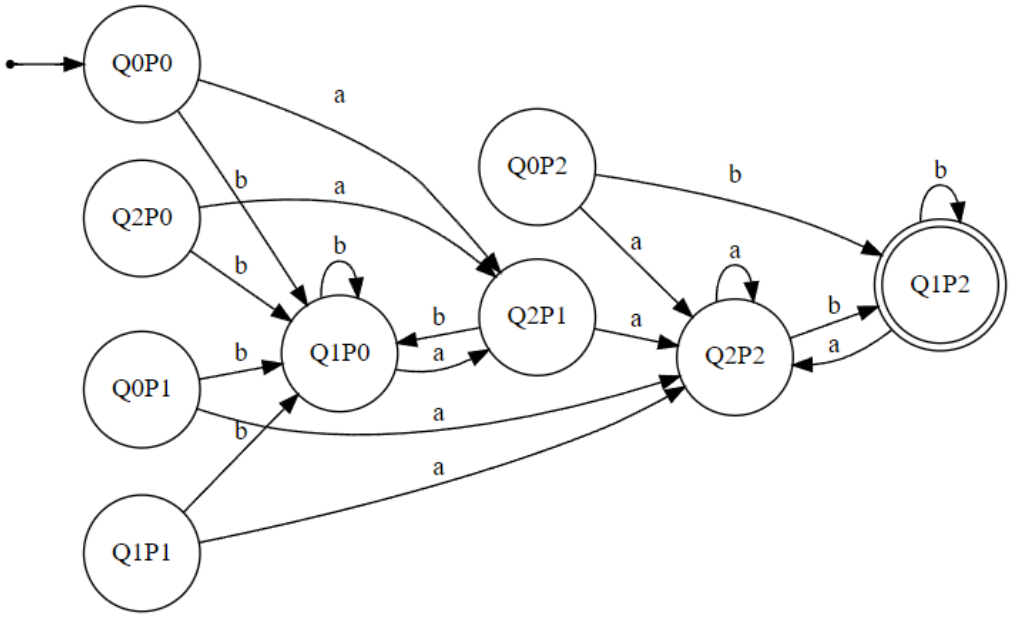
Obrázek 70. Python automat I1

Následně vytvoříme automat A získaný v příkladu 2.3 a porovnáme ekvivalence s automatem získaným pomocí funkce conjunction.

```
VA5 = NFA()
VA5.setSigma(['a','b'])
VA5.addState('Q0P0')
VA5.addState('Q1P0')
VA5.addState('Q2P1')
VA5.addState('Q2P2')
VA5.addState('Q1P2')
VA5.addState('Q0P2')
VA5.addState('Q2P0')
VA5.addState('Q0P1')
VA5.addState('Q1P1')
VA5.addInitial(0)
VA5.addFinal(4)
VA5.addTransition(0, 'b', 1)
VA5.addTransition(0, 'a', 2)
VA5.addTransition(1, 'a', 2)
VA5.addTransition(1, 'b', 1)
VA5.addTransition(2, 'a', 3)
VA5.addTransition(2, 'b', 1)
VA5.addTransition(3, 'a', 3)
VA5.addTransition(3, 'b', 4)
VA5.addTransition(4, 'b', 4)
```

```
VA5.addTransition(4, 'a', 3)
VA5.addTransition(5, 'b', 4)
VA5.addTransition(5, 'a', 3)
VA5.addTransition(6, 'b', 1)
VA5.addTransition(6, 'a', 2)
VA5.addTransition(7, 'b', 1)
VA5.addTransition(7, 'a', 3)
VA5.addTransition(8, 'b', 1)
VA5.addTransition(8, 'a', 3)
VA5.display()
I1.equivalentP(VA5)
```

Výstup:



Obrázek 71. Python automat VA5

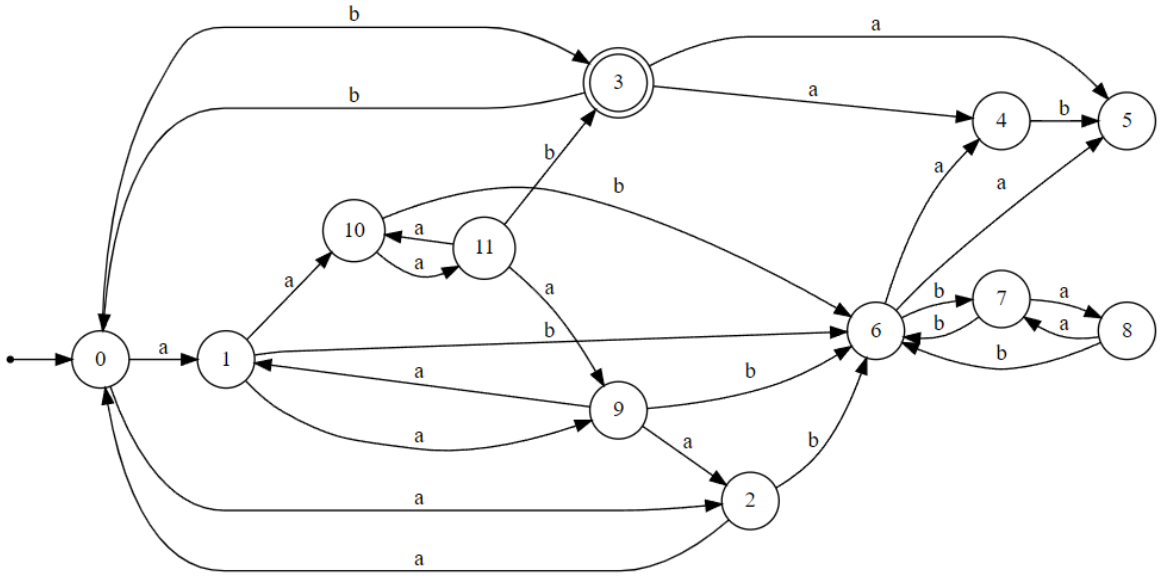
```
True
```

5.1.3.2 Průnik pomocí NKA

Opět provedeme průnik NKA, tentokrát KA2 a KA4 pomocí funkce conjunction.

```
I2 = KA2.conjunction(KA4)
I2.display()
```

Výstup:



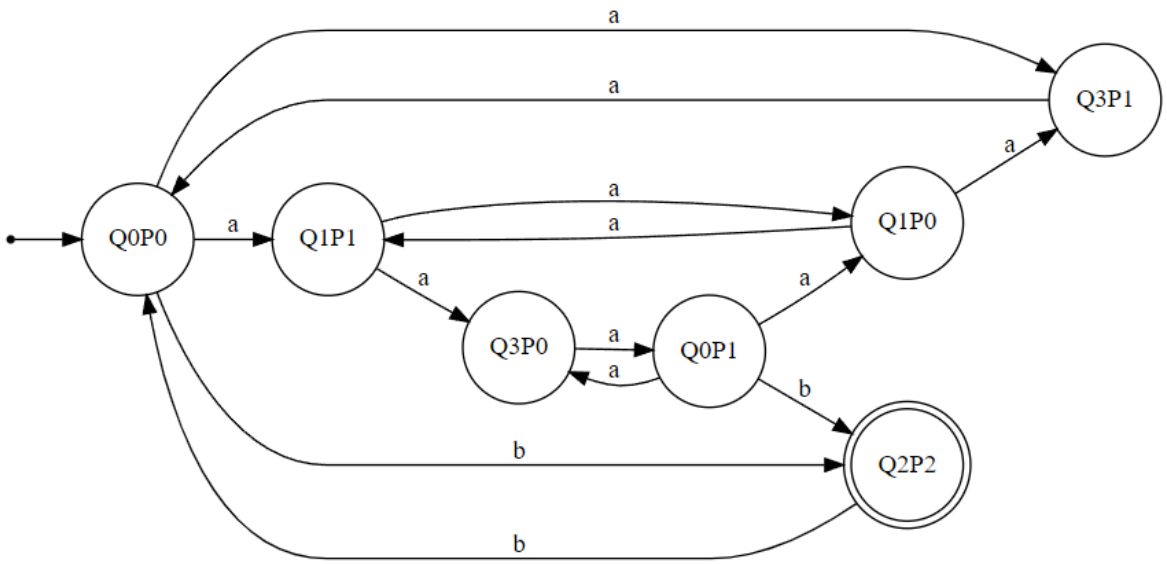
Obrázek 72. Python automat I2

Po provedení průniku sestavíme konečný automat získaný v příkladu 1.3 v příloze 1 a porovnáme ekvivalence s automatem získaným pomocí funkce conjunction.

```
VA6 = NFA()
VA6.setSigma(['a','b'])
VA6.addState('Q0P0')
VA6.addState('Q1P1')
VA6.addState('Q3P1')
VA6.addState('Q2P2')
VA6.addState('Q1P0')
VA6.addState('Q3P0')
VA6.addState('Q0P1')
VA6.addInitial(0)
VA6.addFinal(3)
VA6.addTransition(0, 'a', 1)
VA6.addTransition(0, 'a', 2)
VA6.addTransition(0, 'b', 3)
VA6.addTransition(1, 'a', 4)
VA6.addTransition(1, 'a', 5)
VA6.addTransition(2, 'a', 0)
```

```
VA6.addTransition(3, 'b', 0)
VA6.addTransition(4, 'a', 1)
VA6.addTransition(4, 'a', 2)
VA6.addTransition(5, 'a', 6)
VA6.addTransition(6, 'a', 4)
VA6.addTransition(6, 'a', 5)
VA6.addTransition(6, 'b', 3)
VA6.display()
I2.equivalentP(VA6)
```

Výstup:



Obrázek 73. Python automat VA6

```
True
```

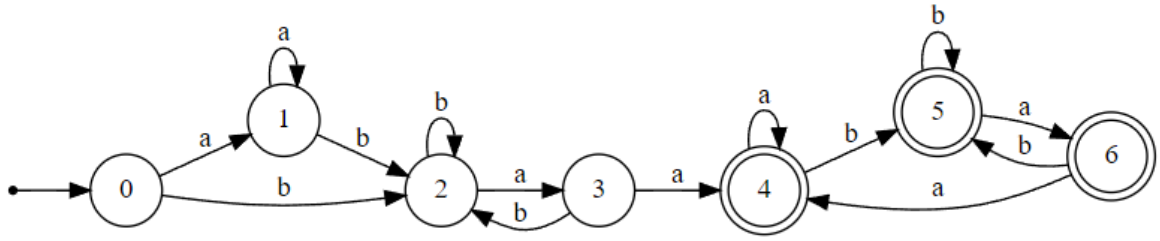
5.1.4 Zřetězení pomocí automatů

5.1.4.1 Zřetězení DKA

Zřetězení DKA KA1 a KA3 můžeme provést pomocí funkce concat.

```
C1 = KA1.concat(KA3)
C1.display()
```

Výstup:



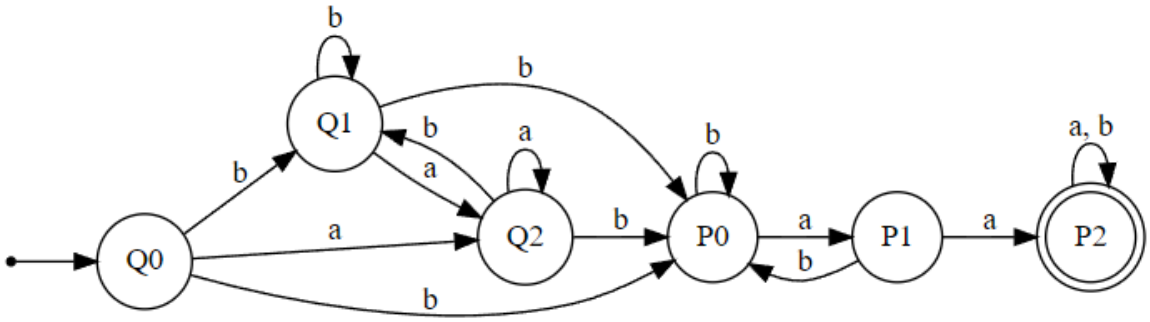
Obrázek 74. Python automat C1

Zde opět porovnáme konečný automat A získaný z příkladu 2.4 s automatem získaným pomocí funkce concat.

```
VA7 = NFA()
VA7.setSigma(['a','b'])
VA7.addState('Q0')
VA7.addState('Q1')
VA7.addState('Q2')
VA7.addState('P0')
VA7.addState('P1')
VA7.addState('P2')
VA7.addInitial(0)
VA7.addFinal(5)
VA7.addTransition(0, 'b', 1)
VA7.addTransition(0, 'a', 2)
VA7.addTransition(1, 'a', 2)
VA7.addTransition(1, 'b', 1)
VA7.addTransition(2, 'a', 2)
VA7.addTransition(2, 'b', 1)
VA7.addTransition(0, 'b', 3)
VA7.addTransition(1, 'b', 3)
VA7.addTransition(2, 'b', 3)
VA7.addTransition(3, 'b', 3)
VA7.addTransition(3, 'a', 4)
VA7.addTransition(4, 'b', 3)
VA7.addTransition(4, 'a', 5)
```

```
VA7.addTransition(5, 'a', 5)
VA7.addTransition(5, 'b', 5)
VA7.display()
C1.equivalentP(VA7)
```

Výstup:



Obrázek 75. Python automat VA7

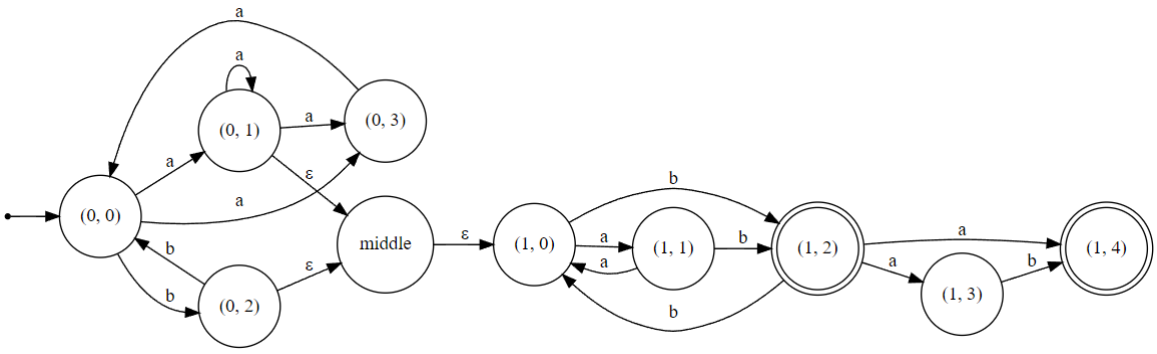
```
True
```

5.1.4.2 Zřetězení pomocí NKA

Zřetězení NKA KA2 a KA4 provedeme stejným způsobem jako u DKA.

```
C2 = KA2.concat(KA4)
C2.display()
```

Výstup:



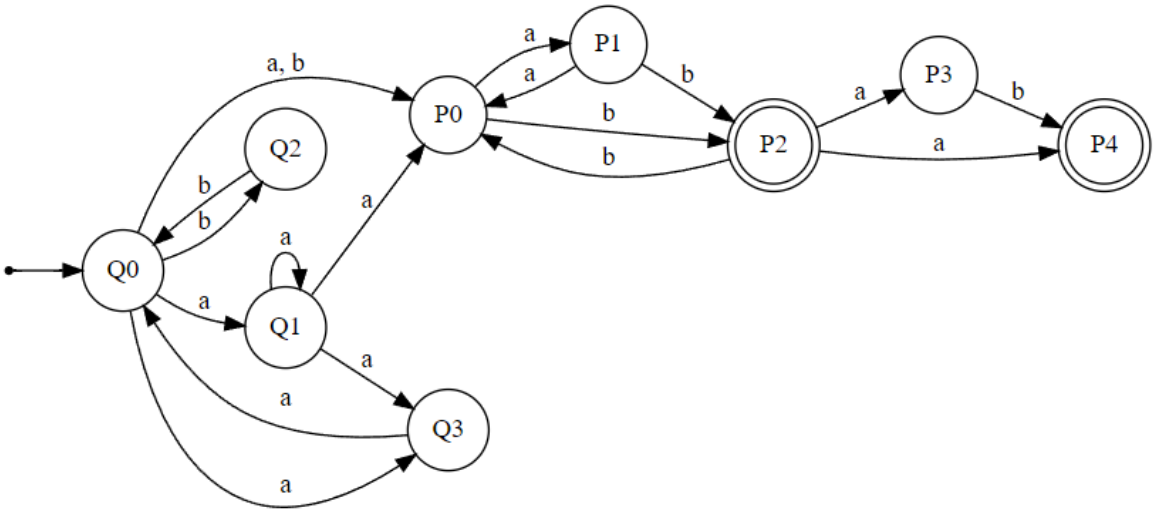
Obrázek 76. Python automat C2

Následně opět porovnáme ekvivalence s automatem A z příkladu 1.4 z přílohy 1

```
VA8 = NFA()
```

```
VA8.setSigma(['a','b'])
VA8.addState('Q0')
VA8.addState('Q1')
VA8.addState('Q2')
VA8.addState('Q3')
VA8.addState('P0')
VA8.addState('P1')
VA8.addState('P2')
VA8.addState('P3')
VA8.addState('P4')
VA8.addInitial(0)
VA8.addFinal(6)
VA8.addFinal(8)
VA8.addTransition(0, 'a', 1)
VA8.addTransition(0, 'b', 2)
VA8.addTransition(0, 'a', 3)
VA8.addTransition(1, 'a', 1)
VA8.addTransition(1, 'a', 3)
VA8.addTransition(2, 'b', 0)
VA8.addTransition(3, 'a', 0)
VA8.addTransition(0, 'a', 4)
VA8.addTransition(1, 'a', 4)
VA8.addTransition(0, 'b', 4)
VA8.addTransition(4, 'a', 5)
VA8.addTransition(4, 'b', 6)
VA8.addTransition(5, 'a', 4)
VA8.addTransition(5, 'b', 6)
VA8.addTransition(6, 'b', 4)
VA8.addTransition(6, 'a', 7)
VA8.addTransition(6, 'a', 8)
VA8.addTransition(7, 'b', 8)
VA8.display()
C2.equivalentP(VA8)
```

Výstup:



Obrázek 77. Python automat VA8

True

5.1.5 Doplněk pomocí automatů

5.1.5.1 Doplněk pomocí DKA

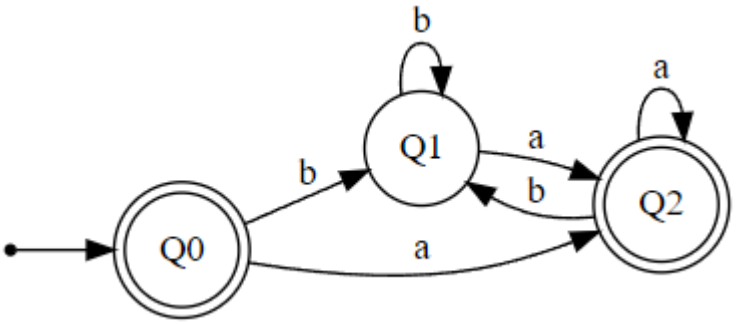
Balíček fado nemá funkci pro operaci doplňku u konečných automatů. Zde je sestrojena funkce, která přijímá konečný automat, kontroluje, zdali je deterministický a podle potřeby jej determinizuje a minimalizuje. Následně jsou pomocí cyklu změněny všechny nekoncové stavy na koncové a naopak. Funkce poté vrací DKA, na kterém byla provedena operace doplňku.

```
def complement(automat):
    if automat.deterministicP():
        doplnek = automat.dup()
    else:
        doplnek = automat.toDFA().minimal()
    for stav in range(len(doplnek.States)):
        if doplnek.finalP(stav):
            doplnek.delFinal(stav)
        else:
            doplnek.addFinal(stav)
    return doplnek
```

Zde je funkce complement použita na DKA KA1.

```
CO1 = complement(KA1)
CO1.display()
```

Výstup:

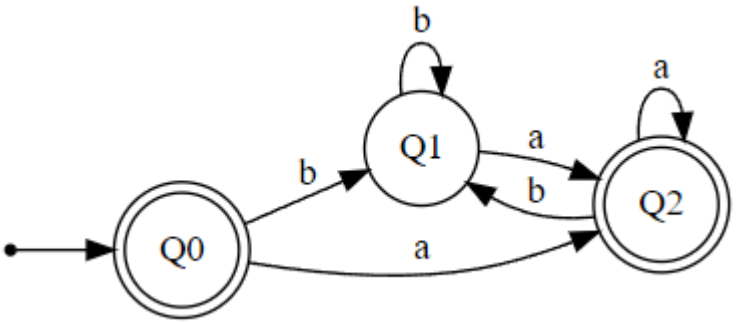


Obrázek 78. Python automat CO1

Následně je výstup funkce complement porovnán s automatem A získaným v příkladu 2.5

```
VA9 = DFA()
VA9.setSigma(['a','b'])
VA9.addState('Q0')
VA9.addState('Q1')
VA9.addState('Q2')
VA9.setInitial(0)
VA9.addFinal(0)
VA9.addFinal(2)
VA9.addTransition(0, 'b', 1)
VA9.addTransition(0, 'a', 2)
VA9.addTransition(1, 'b', 1)
VA9.addTransition(1, 'a', 2)
VA9.addTransition(2, 'b', 1)
VA9.addTransition(2, 'a', 2)
VA9.display()
CO1.equivalentP(VA9)
```

Výstup:



Obrázek 79. Python automat VA9

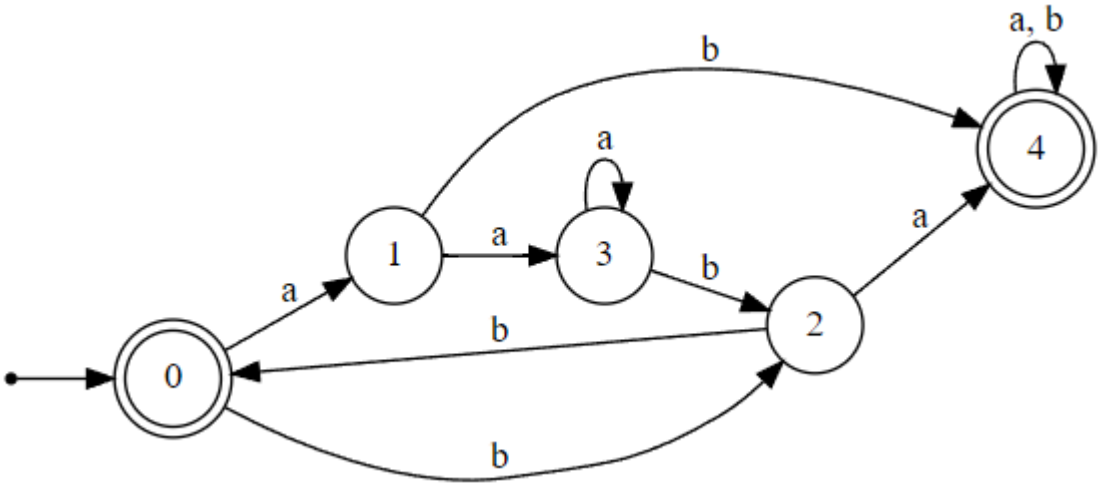
```
True
```

5.1.5.2 Doplněk pomocí NKA

Opět ukázka použití funkce complement tentokrát na NKA KA2.

```
CO2 = complement(KA2)
CO2.display()
```

Výstup:



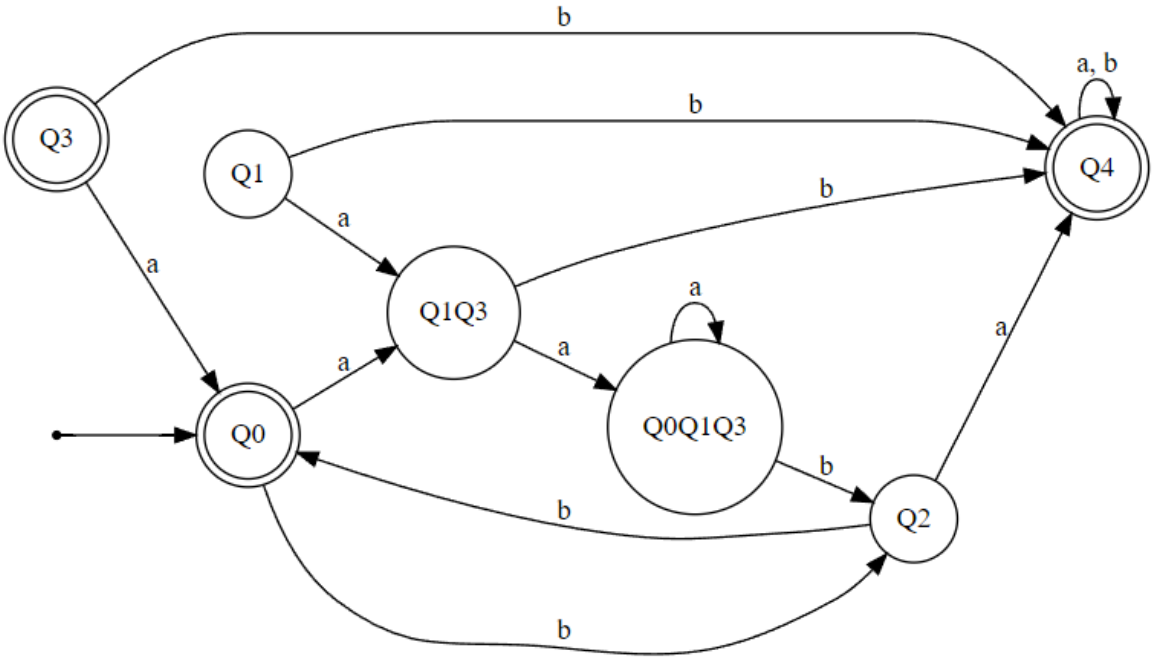
Obrázek 80. Python automat CO2

Po provedení funkce porovnáme ekvivalenci výstupního automatu s automatem A získaným v příkladu 1.5 v příloze 1

```
VA10 = NFA()
VA10.setSigma(['a','b'])
VA10.addState('Q0')
```

```
VA10.addState('Q1')
VA10.addState('Q2')
VA10.addState('Q3')
VA10.addInitial(0)
VA10.addFinal(0)
VA10.addFinal(2)
VA10.addTransition(0, 'a', 1)
VA10.addTransition(0, 'b', 2)
VA10.addTransition(0, 'a', 3)
VA10.addTransition(1, 'a', 1)
VA10.addTransition(2, 'b', 1)
VA10.addTransition(2, 'a', 2)
VA10.display()
CO2.equivalentP(VA10)
```

Výstup:



Obrázek 81. Python automat VA10

```
True
```

5.2 Převody

5.2.1 Převod regulárního výrazu na konečný automat

5.2.1.1 Thompsonův algoritmus

První vytvoříme regulární výraz RE1 z příkladu 3.1 pomocí funkce str2regexp. Následně převedeme regulární výraz na konečný automat pomocí funkce nfaThompson.

```
RE1 = str2regexp("a*b(b+aa*b)*")
KR1 = RE1.nfaThompson()
```

Konečný automat KR1 zde bohužel vzhledem k jeho velikosti nelze vyobrazit, jeho vyobrazení se nachází v souboru Python.

Následně vytvoříme výsledný konečný automat z příkladu 3.1 a porovnáme ho s automatem z funkce nfaThompson.

```
VA11 = NFA()
VA11.setSigma(['a','b'])
VA11.addState('Q0')
VA11.addState('Q1')
VA11.addState('Q2')
VA11.addState('Q3')
VA11.addState('Q4')
VA11.addState('Q5')
VA11.addState('Q6')
VA11.addState('Q7')
VA11.addState('Q8')
VA11.addState('Q9')
VA11.addState('Q10')
VA11.addState('Q11')
VA11.addState('Q12')
VA11.addState('Q13')
VA11.addState('Q14')
VA11.addState('Q15')
VA11.addInitial(2)
VA11.addFinal(15)
VA11.addTransition(0, 'a', 1)
```

```
VA11.addTransition(1, '@epsilon', 0)
VA11.addTransition(2, '@epsilon', 0)
VA11.addTransition(2, '@epsilon', 3)
VA11.addTransition(1, '@epsilon', 3)
VA11.addTransition(3, 'b', 4)
VA11.addTransition(5, 'b', 6)
VA11.addTransition(7, 'a', 8)
VA11.addTransition(8, '@epsilon', 9)
VA11.addTransition(9, 'a', 10)
VA11.addTransition(10, '@epsilon', 9)
VA11.addTransition(10, '@epsilon', 11)
VA11.addTransition(8, '@epsilon', 11)
VA11.addTransition(11, 'b', 12)
VA11.addTransition(4, '@epsilon', 13)
VA11.addTransition(6, '@epsilon', 14)
VA11.addTransition(12, '@epsilon', 14)
VA11.addTransition(14, '@epsilon', 13)
VA11.addTransition(13, '@epsilon', 5)
VA11.addTransition(13, '@epsilon', 7)
VA11.addTransition(14, '@epsilon', 13)
VA11.addTransition(4, '@epsilon', 15)
VA11.addTransition(14, '@epsilon', 15)
KR1.equivalentP(VA11)
```

Výstup:

Opět bohužel není možné vyobrazit Automat VA11 vzhledem k jeho velikosti. Stejně jako u předchozího automatu i jeho vyobrazení se nachází v souboru Python.

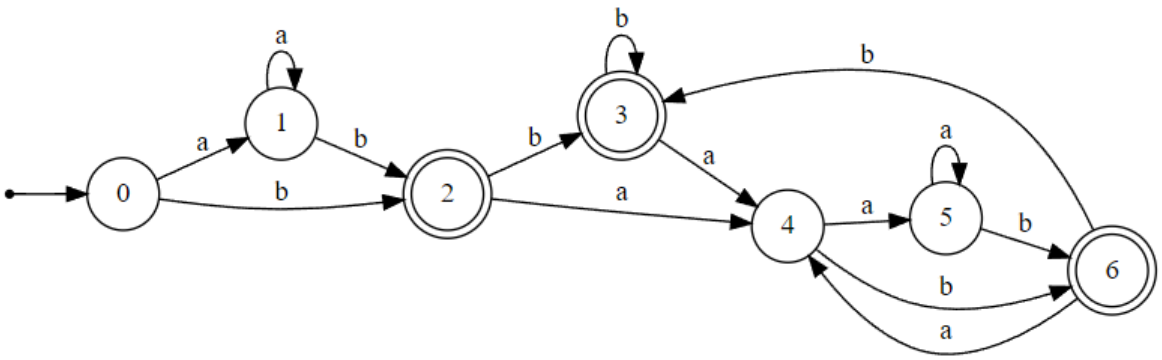
```
True
```

5.2.1.2 Glushkův algoritmus

Zde opět převedeme regulární výraz z příkladu 3.1 na konečný automat, tentokrát pomocí funkce nfaGlushkov.

```
KR2 = RE1.nfaGlushkov()
KR2.display()
```

Výstup:



Obrázek 82. Wolfram automat KR2

Následně zkontrolujeme výsledný konečný automat s automatem, který jsme získaly v příkladu 3.2

```
VA12 = DFA()
VA12.setSigma(['a','b'])
VA12.addState('Q0')
VA12.addState('Q1')
VA12.addState('Q2')
VA12.addState('Q3')
VA12.addState('Q4')
VA12.addState('Q5')
VA12.addState('Q6')
VA12.setInitial(0)
VA12.addFinal(2)
VA12.addFinal(3)
VA12.addFinal(6)
VA12.addTransition(0, 'a', 1)
VA12.addTransition(0, 'b', 2)
VA12.addTransition(1, 'a', 1)
VA12.addTransition(1, 'b', 2)
VA12.addTransition(2, 'b', 3)
VA12.addTransition(2, 'a', 4)
VA12.addTransition(3, 'a', 4)
VA12.addTransition(3, 'b', 3)
VA12.addTransition(4, 'a', 5)
```

```
VA12.addTransition(4, 'b', 6)
VA12.addTransition(5, 'a', 5)
VA12.addTransition(5, 'b', 6)
VA12.addTransition(6, 'a', 4)
VA12.addTransition(6, 'b', 3)
VA12.display()
KR2.equivalentP(VA12)
```

Výstup:

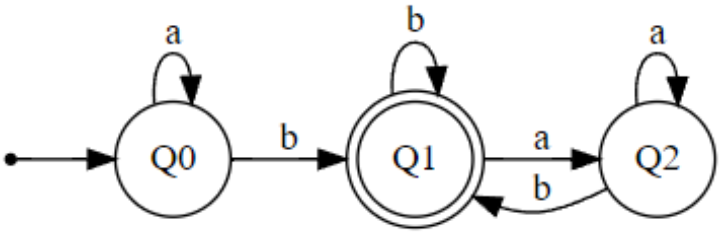
Výsledný automat je identický s automatem KR2.

```
True
```

Zde ještě vytvoříme konečný automat VA13, který jsme získali v příkladu 3.3. A porovnáme ekvivalence mezi automaty.

```
VA13 = DFA()
VA13.setSigma(['a','b'])
VA13.addState('Q0')
VA13.addState('Q1')
VA13.addState('Q2')
VA13.setInitial(0)
VA13.addFinal(1)
VA13.addTransition(0, 'a', 0)
VA13.addTransition(0, 'b', 1)
VA13.addTransition(1, 'a', 2)
VA13.addTransition(1, 'b', 1)
VA13.addTransition(2, 'a', 2)
VA13.addTransition(2, 'b', 1)
VA13.display()
KR1.equivalentP(KR2)
KR2.equivalentP(VA13)
```

Výstup:



Obrázek 83. Wolfram automat VA13

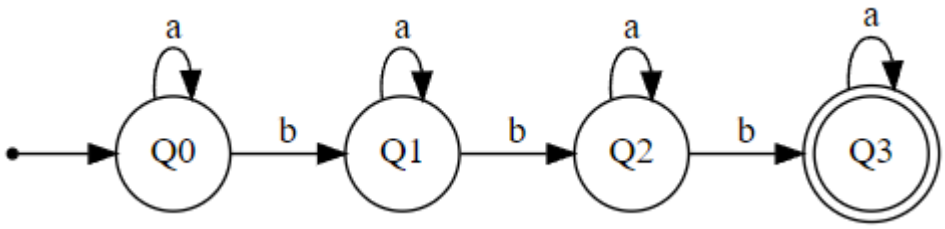
```
True, True
```

5.2.2 Převod konečného automatu na regulární výraz

První vytvoříme DKA KA5 z příkladu 3.4, který následně převedeme pomocí funkce FA2regexpCG na regulární výraz.

```
KA5 = DFA()
KA5.setSigma(['a','b'])
KA5.addState('Q0')
KA5.addState('Q1')
KA5.addState('Q2')
KA5.addState('Q3')
KA5.setInitial(0)
KA5.addFinal(3)
KA5.addTransition(0, 'a', 0)
KA5.addTransition(0, 'b', 1)
KA5.addTransition(1, 'a', 1)
KA5.addTransition(1, 'b', 2)
KA5.addTransition(2, 'a', 2)
KA5.addTransition(2, 'b', 3)
KA5.addTransition(3, 'a', 3)
KA5.display()
RE2 = FA2regexpCG(KA5)
```

Výstup:



Obrázek 84. Wolfram automat KA5

```
((((a* b) a*) b) a*) (b a*)
```

Po získání regulárního výrazu ho můžeme porovnat s regulárním výrazem z příkladu 3.4 pomocí funkce compare.

```
RE3 = str2regexp("a*ba*ba*ba*")
RE2.compare(RE3)
```

Výstup:

```
True
```

5.2.3 Převod DKA na monoid

První vytvoříme DKA KA6 z příkladu 3.6.

```
KA6 = DFA()
KA6.setSigma(['a','b'])
KA6.addState('Q0')
KA6.addState('Q1')
KA6.addState('Q2')
KA6.setInitial(0)
KA6.addFinal(2)
KA6.addTransition(0, 'a', 1)
KA6.addTransition(0, 'b', 0)
KA6.addTransition(1, 'a', 0)
KA6.addTransition(1, 'b', 2)
KA6.addTransition(2, 'a', 1)
KA6.addTransition(2, 'b', 0)
KA6.display()
```

Následně ho převedeme pomocí funkce sMonoid na monoid. Po převedení vypíšeme prvky monoidu pomocí cyklu a následně tabulku 11 z příkladu 3.6 pomocí posledního příkazu.

```
M1 = SemigroupToMonoid[SemigroupGenerate[KA6]];
Pairs[ SemigroupWitnesses[M1], SemigroupElements[M1]] // Grid
RewriteRulesSimplify@SemigroupEquations[M1] // Column
```

Výstup:

```
Monoid:

(0, 1, 2)
(1, 0, 1)          [a]
(0, 2, 0)          [b]
(0, 1, 0)          [a, a]
(2, 0, 2)          [b, a]
(1, 1, 1)          [a, b]
(0, 0, 0)          [b, b]
(3, 3, 3)          [b, a, b]
```

ZÁVĚR

Úvod teoretické části je zaměřen na studium a popis různých způsobů zadání regulárního jazyka, konkrétně regulárního výrazu, konečného automatu a monoidu, a související teorie. Cílem práce bylo popsat různé způsoby zadání regulárních jazyků a podrobněji popsat vybrané souvislosti a ilustrovat je na konkrétních příkladech. Tímto bylo dosaženo hlubšího porozumění tomu, jak tyto způsoby zadání spolu souvisejí a jak se vzájemně doplňují.

Další teoretická část práce byla zaměřena na popis realizace vybraných základních operací s regulárními jazyky v jednotlivých přístupech. Byly popsány operace iterace, sjednocení, průnik, zřetězení a doplněk. Pro každou operaci byla prezentována odpovídající metoda realizace pomocí regulárního výrazu, konečného automatu a monoidů, což umožnilo porovnání a pochopení výhod a omezení jednotlivých přístupů. Vybrané operace byly znázorněny na příkladech.

V poslední teoretické části bakalářské práce byly znázorněny možnosti převodů vybraných způsobů zápisu regulárního jazyka na jiné. Specificky je zde znázorněn převod regulárního výrazu na konečný automat, převod konečného automatu na regulární výraz a převod konečného automatu na monoid.

V praktické části byly zpracovány, pomocí Wolframu Mathematica, jenž využíval balíček Automata a Pythonu, který využíval knihovnu fado, jednotlivé příklady z části teoretické a přiložené přílohy. Byly také vyobrazeny rozdíly v zadávání konečných automatů a regulárních výrazů a v jednotlivých operacích s nimi v jazyce Wolfram a Python. U zpracovaných příkladů v praktické části byly také porovnány ekvivalence výsledků v jednotlivých programovacích jazycích s výsledky z příkladů sestrojených v teoretické části.

SEZNAM POUŽITÉ LITERATURY

[1] LINZ, Peter, 2011. An Introduction to FORMAL LANGUAGES and AUTOMATA [online]. 5th ed. Sudbury: Cathleen Sether [cit. 2023-04-10]. ISBN 978-1-4496-1552-9. Dostupné z: https://fall14cs.files.wordpress.com/2017/04/an-introduction-to-formal-languages-and-automata-5th-edition-2011.pdf

[2] ČEŠKA, Milan, 2002. Teoretická informatika [online]. Brno [cit. 2023-05-25]. Dostupné z: http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/ti.pdf. Učební text. Vysoké učení technické, Fakulta informačních technologií

[3] MARTINEK, Pavel, 2006. Základy teoretické informatiky [online]. Olomouc [cit. 2023-04-25]. Dostupné z: https://phoenix.inf.upol.cz/esf/ucebni/zti.pdf. Učební text. Univerzita Palackého, Přírodovědecká fakulta, Katedra informatiky

[4] ČERNÁ, Ivana, Mojmír KŘETÍNSKÝ a Antonín KUČERA, 2002. Automaty a formální jazyky I [online]. Brno [cit. 2023-05-25]. Dostupné z: https://is.muni.cz/elportal/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf. Učební text. Masarykova univerzita, Fakulta informatiky.

[5] GUPTA, Chandrakishor. Chomsky Hierarchy | Everything You Need to Know | DataTrained. DataTrained [online]. [cit. 2023-05-25]. Dostupné z: https://www.datatrained.com/post/chomsky-hierarchy/

[6] Chomsky Hierarchy, 2021. Devopedia [online]. [cit. 2023-05-25]. Dostupné z: https://devopedia.org/chomsky-hierarchy

[7] Definitions of Regular Language and Regular Expression [online]. [cit. 2023-05-25]. Dostupné z: https://www.cs.odu.edu/~toida/nerzic/390teched/regular/reg-lang/definitions.html

[8] MOORE, Karlelgh et al., 2023. Regular Languages. Brilliant [online]. [cit. 2023-05-25]. Dostupné z: https://brilliant.org/wiki/regular-languages/

[9] LUMUNGE, Erick. Regular Operations in Theory of Computation [online]. [cit. 2023-05-25]. Dostupné z: https://iq.opengenus.org/regular-operations/

[10] VAVREČKOVÁ, Šárka, 2016. Teorie Jazyků a automatů I, Základy teoretické informatiky I [online]. Opava [cit. 2023-05-25]. Dostupné z: http://vavreckova.zam.slu.cz/obsahy/tja/skripta1/tja1.pdf. Skripta do předmětů. Slezská univerzita v Opavě, Filozoficko-přírodovědecká fakulta v Opavě

[11] HALFELD-FERRARI, Mirian. Properties of Regular Languages [online]. [cit. 2023-03-11]. Dostupné z: https://www.univ-orleans.fr/lifo/Members/Mirian.Halfeld/Cours/TLComp/TLComp-ProRegLang.pdf

[12] FIORE, Marcelo, 2010. Regular Languages and Finite Automata [online]. Cambridge [cit. 2023-03-22]. Dostupné z: https://www.cl.cam.ac.uk/teaching/1011/RLFA/LectureNotes.pdf. Lecture notes. Cambridge University Computer Laboratory.

[13] Is a Regex the Same as a Regular Expression?. RexEgg [online]. [cit. 2023-03-29]. Dostupné z: https://www.rexegg.com/regex-vs-regular-expression.html

[14] CHATTOPADHYAY, Agnishom, Karleigh MOORE a Sam SOLOMON. Regular Expressions. Brilliant [online]. [cit. 2023-03-29]. Dostupné z: https://brilliant.org/wiki/regular-expressions/

[15] ŠEVO, Igor, 2016. Are regular expressions (regex) really regular?. StackOverflow [online]. [cit. 2023-03-29]. Dostupné z: https://stackoverflow.com/questions/36283119/are-regular-expressions-regex-really-regular

[16] Finite automata. Engati [online]. [cit. 2023-04-15]. Dostupné z: https://www.engati.com/glossary/finite-automata

[17] VAVREČKOVÁ, Šárka, 2015. Teorie Jazyků a automatů II, Základy teoretické informatiky II [online]. Opava [cit. 2023-04-20]. Dostupné z: http://vavreckova.zam.slu.cz/obsahy/tja2/skripta/tja2.pdf. Skripta do předmětů. Slezská univerzita v Opavě, Filozoficko-přírodovědecká fakulta v Opavě.

[18] Difference between DFA and NFA, 2023. GeeksForGeeks [online]. [cit. 2023-05-05]. Dostupné z: https://www.geeksforgeeks.org/difference-between-dfa-and-nfa

[19] Conversion from NFA to DFA, 2023. GeeksForGeeks [online]. [cit. 2023-05-05]. Dostupné z: https://www.geeksforgeeks.org/conversion-from-nfa-to-dfa/

[20] Minimization of Finite Automata, 2020. TutorialAndExample [online]. [cit. 2023-05-05]. Dostupné z: https://www.tutorialandexample.com/minimization-of-finite-automata

[21] PIN, Jean-Eric, 2022. Mathematical Foundations of Automata Theory [online]. Version of February 18, 2022. Institut de Recherche en Informatique Fondamentale [cit. 2023-04-13]. Dostupné z: https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf

[22] VÁVROVÁ, Pavlína, 2011. Operace s regulárními jazyky pomocí rozpoznávání monoidy [online]. Brno [cit. 2023-05-15]. Dostupné z: https://is.muni.cz/th/psp6g/. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Doc. Mgr. Michal Kunc, Ph. D.

[23] BARTÁK, Roman. Automaty a gramatiky [online]. [cit. 2023-04-20]. Dostupné z: https://ktiml.mff.cuni.cz/~bartak/automaty/lectures/lecture04.pdf

[24] Regular Expressions. Https://cs.uwaterloo.ca/ [online]. [cit. 2023-04-05]. Dostupné z: https://cs.uwaterloo.ca/~s4bendav/CS360/CS360Lec2.pdf

[25] The algebra of sets [online], 2004. [cit. 2023-05-16]. Dostupné z: https://www.umsl.edu/~siegelj/SetTheoryandTopology/The_algebra_of_sets.html

[26] Conversion of Regular Expression to Finite Automata. GeeksForGeeks [online]. [cit. 2023-04-16]. Dostupné z: https://www.geeksforgeeks.org/conversion-of-regular-expression-to-finite-automata/

[27] ARNAIZ-GONZÁLEZ, Álvar, Ismael RAMOS-PÉREZ a César GARCÍA-OSORIO, 2018. McNaughton-Yamada-Thompson algorithm. Seshat [online]. [cit. 2023-04-27]. Dostupné z: http://cgosorio.es/Seshat/thompson?expr=a.(a%7Cb.a)*%7Cc*.a

[28] Regular Expression Matching with Glushkov Automata, 2019. Simon Fraser University [online]. [cit. 2023-04-27]. Dostupné z: https://coursys.sfu.ca/2019sp-cmpt-384-d1/pages/GlushkovRE

[29] Gloushkov's algorithm, 2023. Complex systems and AI [online]. [cit. 2023-04-27]. Dostupné z: https://complex-systems-ai.com/en/language-theory/gloushkov-algorithm/

[30] KANWAL, Ayesha. How to convert finite automata to regular expressions. Educative [online]. [cit. 2023-05-20]. Dostupné z: https://www.educative.io/answers/how-to-convert-finite-automata-to-regular-expressions

[31] NISHIMURA, Naomi. Regular expression identities. University of Waterloo [online]. [cit. 2023-05-10]. Dostupné z: https://cs.uwaterloo.ca/~nishi/360W16/Resources/identities.pdf

[32] Arden's Theorem. Tutorialspoint [online]. [cit. 2023-05-10]. Dostupné z: https://www.tutorialspoint.com/automata_theory/ardens_theorem.htm

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

DKA	Deterministický konečný automat
NKA	Nedeterministický konečný automat
HTML	HyperText Markup Language
URL	Uniform Resource Locator
*	Iterace
∪, +	Sjednocení
∩	Průnik
•, .	Zřetězení

SEZNAM OBRÁZKŮ

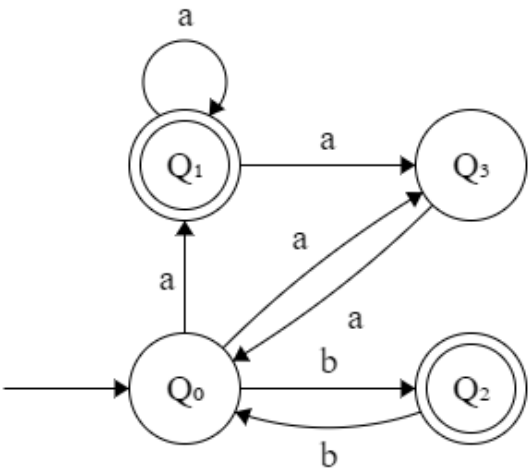
SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OPERACE POMOCÍ NKA

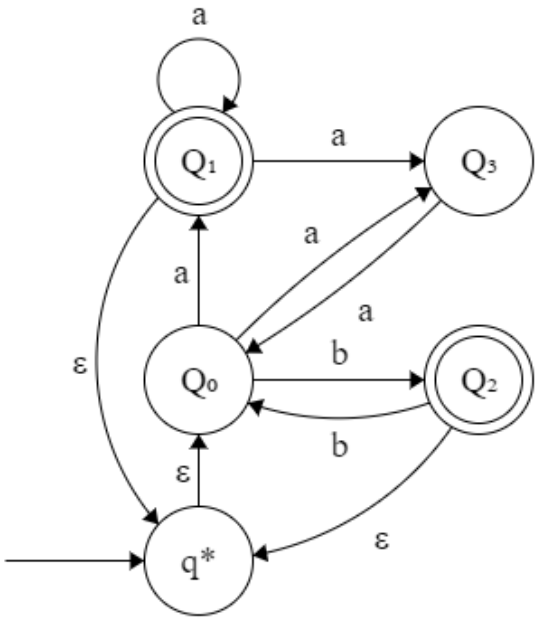
Příklad iterace pomocí NKA 1.1:

Iteraci konečného automatu $A_1 = (Q_1, \Sigma_1, \delta_1, q\_1, F_1)$, kde $Q_1 = \{Q_0, Q_1, Q_2, Q_3\}$, $\Sigma_1 = \{a, b\}$, $q\_1 = Q_0$, $F_1 = \{Q_1, Q_2\}$, přechody jsou znázorněny na obrázku, můžeme provést stejným způsobem jako u DKA.



Obrázek automatu A_1

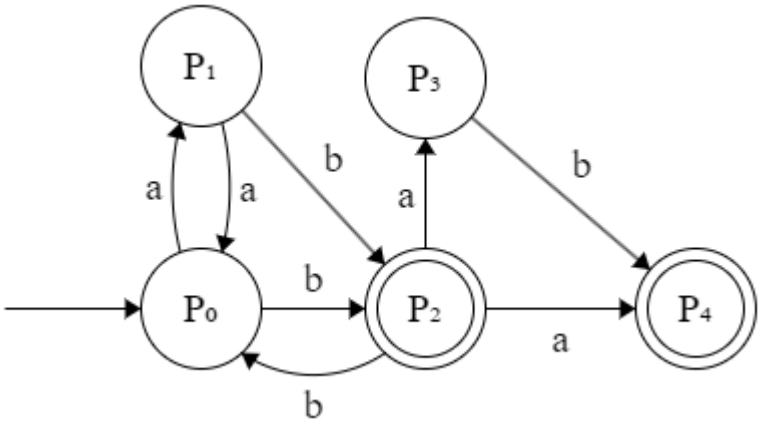
Opět přidáme nový stav q*, který se stane současně počátečním i koncovým stavem. Z q* povede epsilon přechod do stavu Q_0 a ze stavu Q_1 i Q_2 povede epsilon přechod do q*. Získáme konečný automat $A = (Q, \Sigma, \delta, q\_, F)$, kde $Q = \{q^*, Q_0, Q_1, Q_2, Q_3\}$, $\Sigma = \{a, b\}$, $q\_ = q^*$, $F = \{q^*, Q_1, Q_2\}$, který je iterací automatu A_1.



Obrázek automatu A

Příklad sjednocení pomocí dvou NKA 1.2:

Konečné automaty A_1 definovaný v příkladu iterace pomocí NKA, a automat $A_2 = (P, \Sigma_2, \delta_2, q\_2, F_2)$, kde $P = \{P_0, P_1, P_2, P_3, P_4\}$, $\Sigma_2 = \{a, b\}$, $q\_2 = P_0$, $F_2 = \{P_2, P_4\}$, přechody jsou znázorněny na obrázku níže.



Obrázek automatu A_2

Stejně jako u sjednocení pomocí DKA i zde zkontrolujeme, zdali platí podmínka $Q \cap P = \emptyset$.

Následně si sestavíme tabulky přechodů pro automaty A_1 a A_2, stejně jako v příkladu sjednocení DKA.

A_1	a	b
$\rightarrow Q_0$	$\{Q_1, Q_3\}$	Q_2
$\leftarrow Q_1$	$\{Q_1, Q_3\}$	$\{\}$
$\leftarrow Q_2$	$\{\}$	Q_0
Q_3	Q_0	$\{\}$

Tabulka přechodů NKA A_1

A_2	a	b
$\rightarrow P_0$	P_1	P_2
P_1	P_0	P_2
$\leftarrow P_2$	$\{P_3, P_4\}$	P_0
P_3	$\{\}$	P_4
$\leftarrow P_4$	$\{\}$	$\{\}$

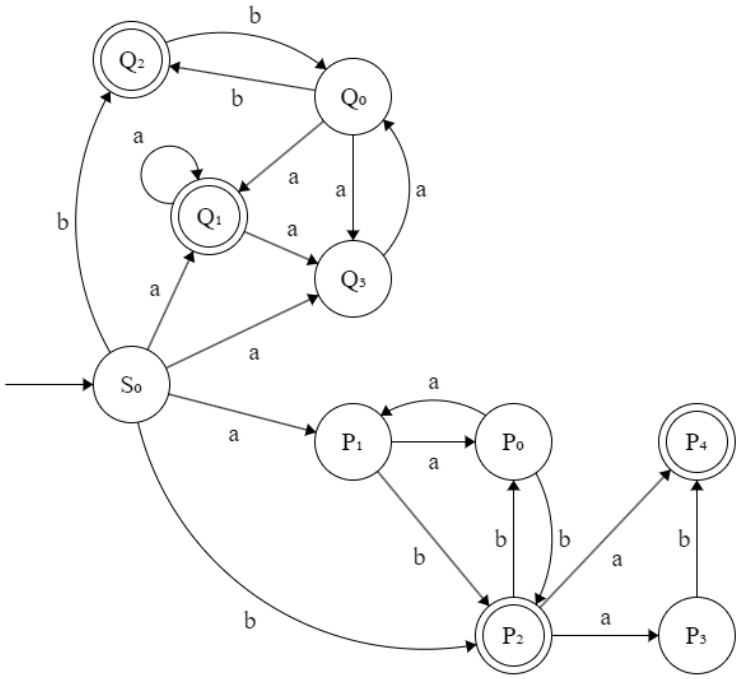
Tabulka přechodů NKA A_2

V dalším korku můžeme z tabulek přechodů automatů A_1 a A_2 sestavit tabulku přechodů pro náš nový automat.

A	a	b
$\rightarrow S_0$	$\{Q_1, Q_3, P_1\}$	$\{Q_2, P_2\}$
Q_0	$\{Q_1, Q_3\}$	Q_2
$\leftarrow Q_1$	$\{Q_1, Q_3\}$	$\{\}$
$\leftarrow Q_2$	$\{\}$	Q_0
Q_3	Q_0	$\{\}$
P_0	P_1	P_2
P_1	P_0	P_2
$\leftarrow P_2$	$\{P_3, P_4\}$	P_0
P_3	$\{\}$	P_4
$\leftarrow P_4$	$\{\}$	$\{\}$

Tabulka přechodů automatu A, který je sjednocením A_1 a A_2

Získáme automat A = (R, Σ, δ, q_, F), kde R = {S_0, Q_0, Q_1, Q_2, Q_3, P_0, P_1, P_2, P_3, P_4}, Σ = {a, b}, q_ = S_0, F = {Q_1, Q_2, P_2, P_4}, který je sjednocením automatu A_1 a A_2. Přechody automatu A jsou znázorněny na obrázku níže.



Obrázek automatu A

Příklad průniku pomocí dvou NKA 1.3:

Konečné automaty A_1 definovaný v příkladu iterace pomocí NKA, a A_2 definovaný v příkladu sjednocení pomocí NKA.

Znovu zkontrolujeme, zdali platí podmínka Q ∩ P = ∅.

Ze sestavených tabulek přechodů automatu A_1 a A_2 z příkladu sjednocení pomocí DKA vytvoříme tabulku přechodů pro náš nový automat.

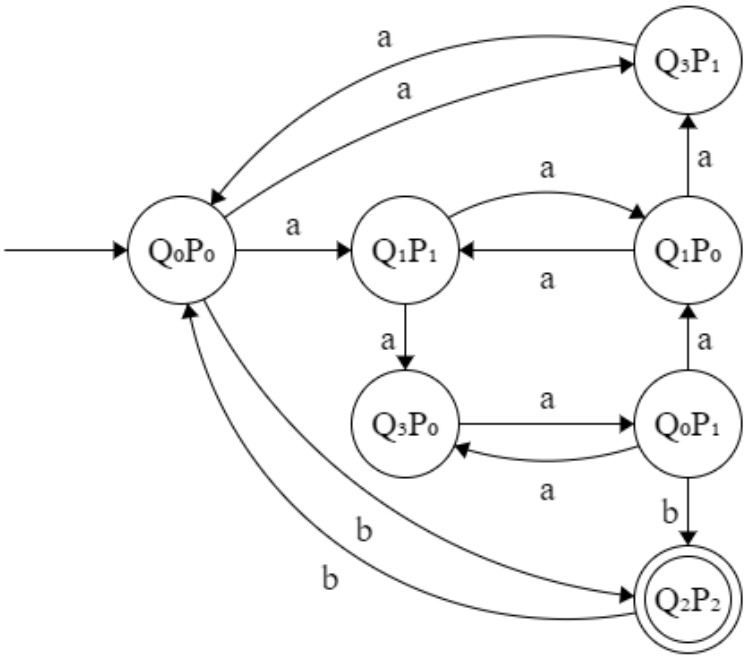
A	a	b
→Q_0P_0	{Q_1P_1, Q_3P_1}	Q_2P_2
Q_0P_1	{Q_1P_0, Q_3P_0}	Q_2P_2
Q_0P_2	{Q_1P_3, Q_1P_4, Q_3P_3, Q_3P_4}	Q_2P_0
Q_0P_3	{Q_1, Q_3}	Q_2P_4
Q_0P_4	{Q_1, Q_3}	Q_2

Q_1P_0	$\{Q_1P_1, Q_3P_1\}$	P_2
Q_1P_1	$\{Q_1P_0, Q_3P_0\}$	P_2
$\leftarrow Q_1P_2$	$\{Q_1P_3, Q_1P_4, Q_3P_3, Q_3P_4\}$	P_0
Q_1P_3	$\{Q_1, Q_3\}$	P_4
$\leftarrow Q_1P_4$	$\{Q_1, Q_3\}$	$\{\}$
Q_2P_0	P_1	Q_0P_2
Q_2P_1	P_0	Q_0P_2
$\leftarrow Q_2P_2$	$\{P_3, P_4\}$	Q_0P_0
Q_2P_3	$\{\}$	Q_0P_4
$\leftarrow Q_2P_4$	$\{\}$	Q_0
Q_3P_0	Q_0P_1	P_2
Q_3P_1	Q_0P_0	P_2
Q_3P_2	$\{Q_0P_3, Q_0P_4\}$	P_0
Q_3P_3	Q_0	P_4
Q_3P_4	Q_0	$\{\}$

Tabulka přechodů automatu A, který je průnikem A_1 a A_2

Při konstruování automatu z tabulky nemusíme využít veškeré přechody. V tomto případě přechody do stavů jako jsou Q_2, P_2 atd. budou vždy mrtvé stavy a není třeba je vyobrazovat. Dále obvykle nemusíme vyobrazovat stavy do kterých nic nevede (mimo počáteční stav), jelikož jejich absence nezmění přijímaná slova konečného automatu.

Získaný konečný automat A = (R, Σ, δ, q_, F), kde R = {Q_0P_0, Q_0P_1, Q_1P_0, Q_1P_1, Q_2P_2, Q_3P_0, Q_2P_0, Q_3P_0}, Σ = {a, b}, q_ = Q_0P_0, F = {Q_2P_2}, který je průnikem automatů A_1 a A_2.



Obrázek automatu A

Příklad zřetězení pomocí dvou NKA 1.4:

Konečné automaty A_1 definovaný v příkladu iterace pomocí NKA, a A_2 definovaný v příkladu sjednocení pomocí NKA.

Jako první znovu zkontrolujeme, zdali platí podmínka Q ∩ P = ∅. Pokud podmínka neplatí, je nutné stavy jednoho z automatů přejmenovat, aby podmínka platila.

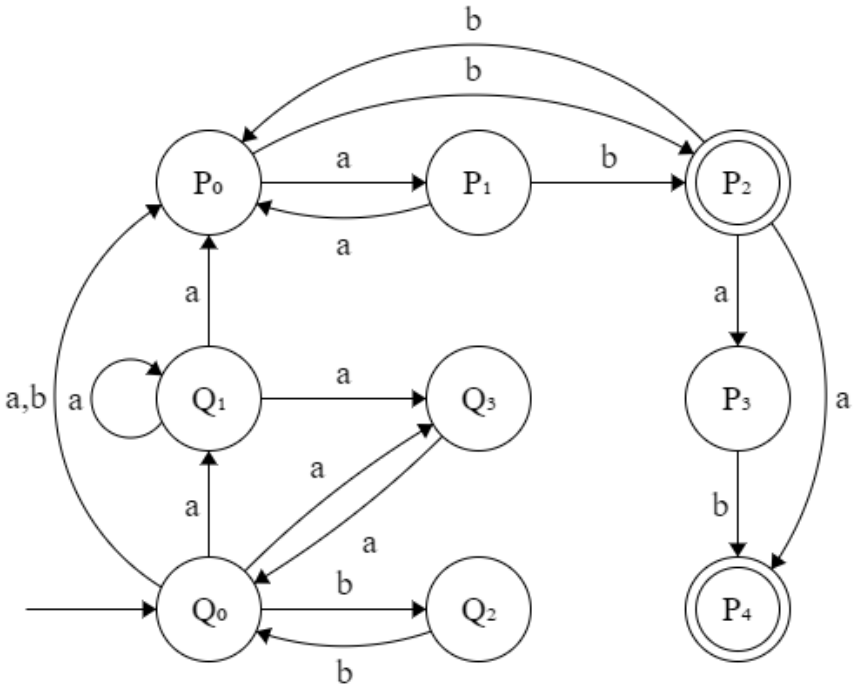
Následně použijeme tabulky přechodů pro automaty A_1 a A_2 z příkladu sjednocení pomocí NKA pro sestavení tabulky přechodů pro náš nový automat.

A	a	b
→Q_0	{Q_1, Q_3, P_0}	{Q_2, P_0}
Q_1	{Q_1, Q_3, P_0}	{}
Q_2	{}	Q_0
Q_3	Q_0	{}
P_0	P_1	P_2

P_1	P_0	P_2
←P_2	{P_3, P_4}	P_0
P_3	{}	P_4
←P_4	{}	{}

Tabulka přechodů automatu A, který je zřetězením A_1 a A_2

Z tabulky přechodů můžeme sestavit automat A = (R, Σ, δ, q_, F), kde R = {Q_0, Q_1, Q_2, Q_3, P_0, P_1, P_2, P_3, P_4}, Σ = {a, b}, q_ = Q_0, F = {P_2, P_4}, který je zřetězením automatu A_1 a A_2.



Obrázek automatu A

Příklad doplňku pomocí NKA 1.5:

Konečný automat A_1 definovaný v příkladu iterace pomocí NKA.

Nejprve převedeme NKA na DKA pomocí tabulky přechodů z příkladu iterace pomocí NKA.

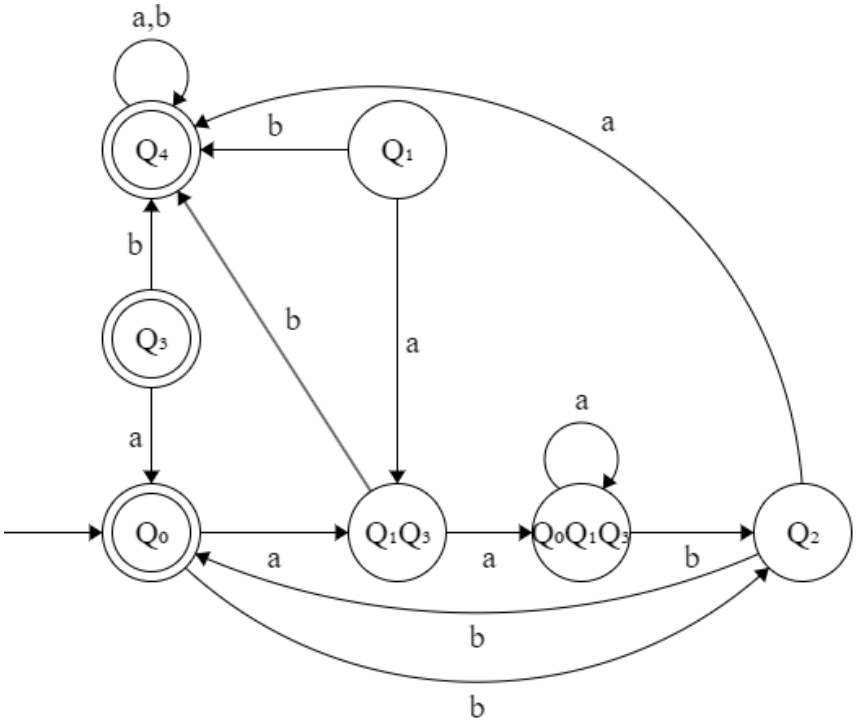
A2	a	b
→Q_0	Q_1Q_3	Q_2
←Q_1	Q_1Q_3	Q_4

←Q_2	Q_4	Q_0
Q_3	Q_0	Q_4
Q_4	Q_4	Q_4
←Q_1Q_3	$Q_0Q_1Q_3$	Q_4
← $Q_0Q_1Q_3$	$Q_0Q_1Q_3$	Q_2

Tabulka přechodů automatu A, který je doplňkem A_1

Následně provedeme operaci doplňku a změníme koncové stavy na nekoncové a naopak.

Výsledný automat A = (R, Σ, δ, q_, F), kde R = {Q_0, Q_1, Q_2, Q_3, Q_4, Q_1Q_3, $Q_0Q_1Q_3$}, Σ = {a, b}, q_ = Q_0, F = {Q_0, Q_3, Q_4}, který je doplňkem automatu A_1.



Obrázek automatu A

PŘÍLOHA P II: CD

Obsah CD:

- Bakalářská práce v elektronické podobě
- BP_Wolfram_Mathematica – Obsahuje zdrojový kód se zpracovanými příklady ve Wolframu Mathematice
- BP_Python – Obsahuje zdrojový kód se zpracovanými příklady v Pythonu