VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ALTERNATIVNÍ TRANSFORMACE JAZYKOVÝCH MODELŮ

ALTERNATIVE TRANSFORMATIONS OF LANGUAGE MODELS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN HAVEL**
AUTHOR

VEDOUCÍ PRÁCE **prof. RNDr. ALEXANDER MEDUNA, CSc.**
SUPERVISOR

BRNO 2021

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Havel Martin**
Program: Informační technologie
Název: **Alternativní transformace jazykových modelů**
Alternative Transformations of Language Models
Kategorie: Teoretická informatika
Zadání:

1. Dle instrukcí od vedoucího se seznamte s jazykovými modely, např. gramatikami a automaty. Seznamte se s jejich transformacemi.
2. Dle instrukcí od vedoucího zaveďte a studujte nové způsoby transformací studovaných v bodě 1. Popište konstrukci a implementaci těchto transformací.
3. Dle konzultací s vedoucím, studujte využití transformací navržených v předchozím bodě. Implementujte je a ověřte je na řadě příkladů.
4. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Maxime Crochemore and Wojciech Rytter: Text algorithms. New York ; Oxford : Oxford University Press, 1994

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Meduna Alexander, prof. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt
Tato práce poskytuje ucelený přehled poznatků z oblasti regulárních výrazů, konečných automatů a transformací z regulárního výrazu na konečný automat. Práce navrhuje nové transformace se zaměřením na minimalizaci počtu stavů a počtu pravidel konečných automatů. Koncept alternativních transformací je zpracován do algoritmů a prokázán matematickými důkazy. Cílem práce je obohatit transformace o nové alternativy na poli regulárních výrazů a konečných automatů. Pozornost je především věnována ekonomické stránce finálního konečného automatu. V rámci práce se podařilo sestrojit algoritmy, které jsou schopny transformovat regulární výrazy na konečné automaty. Práce zároveň poskytuje návod k jejich implementaci. Prezentuje obecný koncept transformací, který umožňuje tvořit méně rozsáhlé konečné automaty. Využitím uvedeného přístupu je možné rozšířit řadu transformací o alternativní verze.

Abstract
This thesis provides a summary of knowledge of regular expressions, finite automatas and transformation from regular expression to finite automata. The thesis proposes new transformation focused on minimal count of states and rules of finite automatas. Concept of alternative transformation is processed into algorithms and proved by mathematical proofs. The aim of thesis is to introduce approaches of transformation with new ones in the field of regular expressions and finite automatas. Great attention is dedicated to economic perspective of final finite automata. There were created algorithms, which are capable of transformation regular expression to finite automata. This work also provides a simple recipe for implementation of these structures. We introduced generic concept of transformation, that allows to create less complicated finite automatas. Using presented techniques it is possible to expand known transformation with new ones.

Klíčová slova
formální jazyky, matematické modely, konečné automaty, regulární výrazy, transformace, alternativní transformace

Keywords
formal languages, mathematical models, finite automata, regular expression, transformation, alternative transformation

Citace
HAVEL, Martin. *Alternativní transformace jazykových modelů.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Alexander Meduna, CSc.

Alternativní transformace jazykových modelů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. RNDr. Alexandra Meduny CSc. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . .
Martin Havel
8. května 2021

Poděkování

Chtěl bych poděkovat vedoucímu práce, prof. RNDr. Alexandrovi Medunovi CSc., za poskytnutou odbornou pomoc a cenné rady a Mgr. Petru Havlovi za pravopisnou korekci.

Obsah

1 Úvod **2**

2 Základní definice a teorie **4**
2.1 Teoretická informatika . . . 4
2.2 Množina . . . 5
2.3 Matematická notace . . . 5
2.4 Abeceda . . . 7
2.5 Regulární výrazy (RV) . . . 9
2.6 Konečné automaty (KA) . . . 9

3 Transformace **15**
3.1 Transformace - Sjednocení . . . 15
3.2 Transformace - Konkatenace . . . 16
3.3 Transformace - Iterace . . . 16
3.4 Příklad užití . . . 17

4 Alternativní transformace **21**
4.1 Transformace - Sjednocení . . . 21
4.2 Transformace - Konkatenace . . . 24
4.3 Transformace - Iterace . . . 26
4.4 Srovnání . . . 29
4.5 Příklad užití . . . 30

5 Implementace **33**
5.1 Návrh implementace . . . 33
5.2 Implementační jazyk . . . 33
5.3 Struktura programu . . . 33
5.4 Výstup . . . 34
5.5 Spuštění . . . 35
5.6 Příklady užití . . . 36

6 Závěr **43**

Literatura **45**

Kapitola 1

Úvod

Jazyk je základním dorozumívacím prostředkem. Přirozený lidský jazyk je perfektní pro potřeby lidské komunikace. Pro potřebu komunikace s výpočetními zařízeními není nejvhodnější kvůli svojí komplikovanosti a nejednoznačnosti, která vede k problematickému systémovému popisu problému.

Proto jsou jednou ze základních a velmi významných oblastí informatiky formální jazyky a jejich teorie. Teorie formálních jazyků představuje jedno z nejdokonaleji prozkoumaných odvětví informačních technologií. Napříč tomu se neustále vyvíjí a neustále otevírá nové oblasti, které lze prozkoumat, rozšířit nebo ověřit. Zaměření této práce je na matematické modely formálních jazyků.

Formální jazyky jsou reprezentovány matematickými modely, mezi které patří gramatiky, regulární výrazy nebo automaty. Popis jazyka, který je zapotřebí zpracovat, zapsaného výčtem je sice po všech stránkách validní, ovšem nepříliš efektivní nebo přehledný. Z toho důvodu se začali využívat regulární výrazy pro zápis, které přinesly mnoho výhod. Ovšem pro činnost v rámci výpočetního zařízení je nezbytné převést tento regulární výraz na konečný automat, který je již pro zpracování do výpočetního jazyka realizovatelný.

Při tomto převodu (transformaci) lze prioritizovat mnoho různých faktorů. Cílem této bakalářské práce je navrhnout algoritmy pro transformaci regulárního výrazu na konečný automat tak, aby snížili počet přechodů a stavů na minimum a zároveň zachovaly validnost této transformace. Zachování validnosti transformace bude možné dokázat pomocí matematického důkazu.

Text práce je rozdělený do šesti kapitol.

Druhá kapitola [2] obsahuje úvod do teoretické informatiky se zaměřením na základní pojmy obsažené v této bakalářské práci. Dále obsahuje základní matematické pojmy, které jsou naprosto esenciální pro porozumění základním matematickým veličinám, jejich operacím a zápisům využívaných v této práci. Dále následují jednotlivé pojmy z teoretické informatiky se základním seznámením, matematickou definicí, příklady a případně operacemi nad danými pojmy.

Kapitola [3] je věnována zavedeným transformacím z regulárního výrazu na konečný automat. Tyto transformace jsou zaměřeny na složitost algoritmu nikoliv na finální konečný automat, který může při komplikovanějších regulárních výrazech nabírat zbytečně vysoké složitosti. Tato kapitola je dělena do tří sekcí, kde každá popisuje jednu operaci v posloupnosti sjednocení, konkatenace a iterace. Na závěr je potom vytvořen příklad, který ukazuje převod regulárního výrazu na konečný automat a umožňuje porovnání s alternativní transformací. S těmito transformacemi se můžeme setkat v několika odborných publikací zaměřených na toto téma.

Jádrem práce je kapitola [4] s navrženými alternativními transformacemi. Cílem těchto navržených alternativních transformací je snížit počet stavů a přechodů. Tato kapitola je opět rozdělena do tří sekcí v posloupnosti sjednocení, konkatenace a na závěr iterace. V každé sekci je myšlenka snížení počtu stavů a přechodů oproti kapitole [3] vyjádřena slovy. Následně je vytvořen algoritmus na základě myšlenky a jeho formální matematický důkaz, aby byla zachována ekvivalence matematických modelů neboli přijímaných jazyků. Následuje manuálně vytvořený příklad znázorňující transformaci stejného regulárního výrazu na konečný automat pomocí alternativních algoritmů, který umožňuje vizuální porovnání konečných automatů.

Pátá kapitola [5] je věnována implementační části bakalářské práce. Cílem programu implementovaného pro tuto práci je prokázání fungování algoritmů a jejich tvorbě v praxi. Začátek kapitoly zahrnuje popis nezbytných úkonů k zprovoznění programu. Následuje popis struktury programu a na závěr kapitoly je ukázka příkladů spuštění s očekávanými výstupy.

Poslední kapitola [6] rekapituluje zaměření textu této bakalářské práce, shrnuje dosažené výsledky, porovnává očekávání s realitou a uvažuje nad dalším možným vývojem tohoto tématu.

Kapitola 2

Základní definice a teorie

Tato kapitola se zabývá základními pojmy, které jsou nezbytné pro ustanovení základních celků a operacemi nad nimi. Definice množin jsou přejaty z [5], a ostatní definice jsou vzaty z [4]. Další informace jsou obsaženy v [6], [7] a [2].

2.1 Teoretická informatika

Základy teoretické informatiky byly vytvořeny už v předchozích stoletích. Můžeme uvažovat, že teoretická informatika vznikla na základě matematiky, nauky o jazyku a biologie. Na začátku 20. století byla specifická hlavně rozvojem logiky. Logika má počátky již v antice, ale až od 20. století pozorujeme snahu využívat logiku do všech existujících vědních oborů. Během rozkvětu logiky v tomto období největší mozkové kapacity této doby zjistili, že všechno nelze vypočítat, dokázat nebo matematicky odvodit. Proto se největší mozkové kapacity své doby zaměřili na to, co je možné vypočítat, dokázat či matematicky odvodit nikoliv na to co nelze matematicky odvodit, dokázat nebo vypočítat.

Tímto se mimo jiné zabýval a velké pokroky udělal Alan Turing, který na univerzitě v Cambridge v roce 1937 publikoval článek On Computable Numbers, kde poprvé představil koncept univerzálního matematického modelu, které při ideálním naprogramování umožňuje vypočítat to co lze vypočítat. Dnes tento model známe pod názvem Turingův stroj. Turingův stroj se skládá většinou ze tří částí. Za nejkomplikovanější část lze považovat konečnou řídicí jednotku, která využívá takzvaně stav. Na konečnou řídící jednotku je připojena čtecí a zapisovací hlava. Poslední částí je nekonečná páska, do které je na začátku vložen vstupní řetězec, který je poté načítán hlavou a zpracován řídící jednotkou. Rozhodování probíhá na základě dvou údajů a to stavu a obsahu pole, kde rozhodnutí určuje změnu stavu, přepsání obsahu pole v pásce nebo posun na pásce.

Pro jednodušší výpočty je Turingův stroj složitý a nepraktický. To mělo za následek vznik dalších jednodušších modelů například konečný automat nebo zásobníkový automat. Tyto automaty nesplňují podmínky, aby mohli považovány za univerzální jako Turingovy stroje, což znamená, že nedokáží vypočítat vše vypočitatelné, ale dokáží pracovat efektivněji ve vhodných případech a pro stanovené úkoly mohou zcela dostačovat.

Pro teoretickou informatiku je velmi významným oborem také jazykověda. Gramatiky se zaměřením na formální jazyky jsou jedním ze základních kamenů této vědní disciplíny. Formální gramatika se zabývá tvorbou slov. Noam Chomsky známý americký lingvista se ve svém výzkumu zabýval syntaxí jazyka. Cílem jeho zkoumání bylo zjistit zdali je možné syntaxi jakéhokoliv jazyka popsat sadou pravidel. Tuto sadu pravidel definoval jako for-

mální gramatiku. Každý člověk má vrozenou znalost takzvané univerzální gramatiky, která je pro všechny jazyky stejná a pomáhá mu naučit se konkrétní jazyky. Základní myšlenkou je, že věta se skládá ze slov, při skládání jsou zapotřebí pomocná slova, které můžeme nahradit kombinací slov a pomocných slov. Pomocná slova označujeme neterminální symboly, skutečná slova jazyka potom terminální symboly.

2.2 Množina

Množina je soubor objektů chápaných jako celek. Objekty množiny se nazývají prvky množiny. Charakterizující vlastnost množiny je, že je jednoznačně určena svými prvky. Označujeme je písmeny latinské abecedy. Například: $B = \{a_1, a_2, z\}$.

Rozdělení množin

- Prázdná – $A = \emptyset$ (neobsahuje žádný prvek)
- Neprázdná – $A \neq \emptyset$ (obsahuje 1 až n prvků.)
- Konečné – mají konečný počet prvků.
- Nekonečné - nemají konečný počet prvků.
- Disjunktní - $A \cap B = \emptyset$ (nemají společné prvky, jejich průnik je prázdná množina)
- Nedisjunktní - $A \cap B \neq \emptyset$ (mají společné prvky, jejich průnik není prázdná množina)

Množinové operace

- Sjednocení množin $A \cup B$ (A a B je množina všech společných prvků, které patří aspoň do jedné z množin A nebo B)
- Průnik množin $A \cap B$ (A a B je množina všech prvků, které patří do množiny A a zároveň do množiny B)
- Rozdíl množin $A - B$ (A a B je množina všech prvků, které patří do množiny A a zároveň nepatří do množiny B)
- Doplněk množiny $A' = U - A$ (A' je množina všech prvků, které nepatří do množiny A)

2.3 Matematická notace

Jeden matematický výraz lze zapsat více způsoby. Tento jev může mít velké množství příčin. Mezi nejznámější můžeme zařadit komutativnost, asociativnost a distributivnost.

- komutativnost - komutativnost je ze všech operací asi nejsnáze pochopitelnou, podvědomě se s ní totiž setkáme již při úplných začátcích matematiky. Binární operace je komutativní, pokud při ní nezáleží na pořadí operandů. Jako komutativní operaci můžeme uvažovat sčítání, kde $6 + 9 = 9 + 6$, to ovšem neplatí například o dělení. Příkladem může být $8/4 \neq 4/8$.

- asociativnost - Matematická definice asociativnosti zní následovně: Binární operace & na množině S je asociativní, když pro všechna $x, y, z \in S$ platí, že $x\&(y\&z) = (x\&y)\&z$. Jako příklad asociativní operace můžeme uvažovat znovu +, protože $6 + (9 + 3) = (6 + 9) + 3$. Oproti tomu odčítání asociativní není. Toto tvrzení můžeme jednoduše dokázat $(9 - 5) - 1 \neq 9 - (5 - 1)$.

- distributivnost - je definována jako: Binární operace & je distributivní vůči operaci \$ na množině S, jestliže pro všechna $x, y, z \in S$ platí, že $x\&(y\$z) = (x\&y)\$(x\&z)$. Jako triviální příklad lze uvést $5 * (2 + 3) = 5 * 2 + 5 * 3$

Ovšem tyto triviální znalosti pouze nastiňují oblast matematiky, kde se můžeme zabývat ekvivalencí výrazů do hloubky. Matematická pravidla jsou ustanovena specificky pro určité problémy a proto pro jednu informaci lze uvažovat více zápisů, dle potřeby osoby, která se daným problémem zabývá. Pro programování matematických či jiných operací je často nezbytné převést infixovou notaci na jinou a ani pro tuto práci tomu není výjimkou. Proto je nezbytné být obeznámen s těmito pojmy:

- Infixová notace je běžný způsob zápisu matematických výrazů, ve kterém jsou operátory napsány mezi operandy, se kterými pracují (např. $7 + 7$). Její počítačová analýza není tak jednoduchá a proto se provádí převod na jiné notace. V infixové notaci jsou, na rozdíl od prefixové nebo postfixové notace, závorky nutností. V případě, že závorky chybějí, záleží na prioritě početních operací.

- Prefixová notace (někdy označovaná jako polská notace) je způsob zápisu logických, aritmetických a algebraických výrazů. Její charakteristikou je především zápis operátorů vlevo před operandy. Protože je pořadí operátorů fixní, syntaxe nepotřebuje žádné závorky a zápis je tedy jednoznačný. Polský logik Jan Łukasiewicz vymyslel tuto notaci okolo roku 1920, proto je tato notace známá jako polská notace.

- Postfixová notace (též reverzní polská notace, zkráceně jako RPN) je způsob zápisu matematického výrazu, kde operátor následuje své operandy, přičemž je odstraněna nutnost používat závorky a priorita operátorů se vyjadřuje samotným zápisem výrazu. Vytvořil ji filozof a počítačový vědec Charles Hamblin v polovině padesátých let. Oblíbená je při implementaci vyhodnocování výrazů, například při programování překladače nebo interpretu pro různé programovací jazyky.

Pro pochopení implementace je nezbytné být seznámen s převodem infixové notace na postfixovou notaci. Pro převod jsou využity tři základní komponenty: vstupní výraz v infixovém tvaru, zásobník a výstup. Pokud má algoritmus k dispozici tyto komponenty, může převést infixovou notaci na postfixovou na základě těchto pravidel:

- Vstupní symbol přepiš na výstup.

- Otevírací závorku ulož na vrchol zásobníku.

- Pokud je v zásobníku operátor o stejné či vyšší prioritě přesuň operátor ze zásobníku na výstup, jinak ulož vstupní operátor na vrchol zásobníku.

- Při načtení uzavírací závorky přesunuj operátory ze zásobníku na výstup dokud nenarazíš na otevírací závorku.

2.4 Abeceda

Abeceda je konečná, neprázdná množina elementů, kterým říkáme symboly. Příkladem abecedy je třeba množina $\{a, b\}$, množina číslic $\{0, 1, 2, .., 9\}$, nebo prázdná množina $\emptyset$.

Řetězec

Nechť Σ je abeceda a ϵ značí tzv. prázdný řetězec (neobsahuje žádný symbol)

- ϵ je řetězec nad abecedou Σ
- pokud x je řetězec nad Σ a $a \in \Sigma$, potom xa je řetězec nad abecedou Σ

Příkladem pro abecedu $\{a, b\}$ je řetězec $aabb$. Počet členů v řetězci x značíme $|x|$ a nazýváme délka řetězce. Pro tento případ je $|x| = 4$. Dále můžeme zjišťovat počet výskytů v řetězci, který pro znak a značíme $\#_a(x)$. Jako příklad můžeme uvést $\#_b(abbaab) = 3$. Jak je výše zmíněno prázdný řetězec neboli ϵ neobsahuje žádné symboly, a z toho důvodu má nulovou délku.

Nechť Σ^* značí množinu všech řetězců nad Σ. Množinu všech neprázdných řetězců je zvykem značit Σ^+. Platí například:

- $\{a\}^* = \{\epsilon, a, aa, aaa, aaaa, ...\}$
- $\{a\}^+ = \{a\}^* - \{\epsilon\}$
- $\{a, b\}^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, abb, baa, bab, bba, bbb, ...\}$

Dále kvůli matematické kompletnosti se definuje $\emptyset^* = \{\epsilon\}$ a $\emptyset^+ = \emptyset$. U řetězců můžeme uvažovat, že pro každé dvě slova u a v lze aplikovat binární operaci konkatenace (zřetězení), která je označována symbolem "." a bývá definována předpisem $u.v = uv$. Jako příklad lze uvést konkatenaci dvou řetězců $aabb$ a $abbaab$ jako $aabbabbaab$. Tato binární operace je asociativní, proto platí $u.(v.w) = (u.v).w$ pro u, v, w symbolizující libovolné řetězce. Konkatenace s ϵ se chová jako prvek řetězce neboli $u.\epsilon = \epsilon.u = u$ pro libovolný řetězec u. Často se symbol konkatenace "." vynechává a pokud není v konkrétním případě objektem pozornosti je tomu tak i v této práci. Každý neprázdný řetězec nad abecedou lze získat konkatenací symbolů této abecedy.

Pro jednodušší specifikaci jazyka je zvykem zavádět unární operace *i-té* mocniny řetězce. Tato mocnina je definována induktivně pro každé $i \in N_0$: nechť Σ je libovolná abeceda a $u \in \Sigma$ libovolné slovo. Potom platí:

- $u^0 = \epsilon$
- $u^{i+1} = u.u^i$

Například $(01a)^3 = 01a01a01a$, kde kulaté závorky nejsou součástí slova, ale slouží jako metasymboly pro vymezení rozsahu unární operace.

Dále je nezbytné nadefinovat podřetězec, kdy řetězec u je podřetězcem řetězce v, v případě že existují řetězce w, x takové, že $v = wux$. Pokud je $w = \epsilon$ tak u označujeme jako prefix (též předpona) řetězce v. Tím ovšem vzniká nezbytnost $x \neq \epsilon$, jinak by se nejednalo o podřetězec, ale o ekvivalenci, protože pro $v = wux$, kde $w = \epsilon \cap x = \epsilon$ můžeme upravit $v = wux = \epsilon u \epsilon = u\epsilon = u$, takže $v = u$. Také můžeme uvažovat, že $x = \epsilon$. V tomto případě je řetězec u suffixem (příponou) řetězce v. I v tomto případě vzniká nezbytnost $w \neq \epsilon$, jinak by řetězec u nebyl podřetězcem v, ale řetězcem ekvivalentním. Například řetězec aa je suffixem řetězce $abbaa$ a není podřetězcem řetězce bab.

Jazyk

Každá podmnožina $L \subseteq \Sigma^*$ je jazyk nadΣ. Například $\{01, 1, 01001\}$ je jazyk nad abecedou $\{0, 1\}$. Jazyky mohou být také nekonečné. Např. $\{w \in \{0,1\}^* | \#_0(w) = \#_1(w)\}$ je jazyk nad abecedou $\{0, 1\}$ obsahující všechna slova, ve kterých se 0 i 1 vyskytují ve stejném počtu. Tedy 001101 a ϵ jsou součástí jazyka a 00001 nebo 111111 nejsou součástí jazyka.

Jazykové operace

- Konkatenace jazyků - Nechť L_1 a L_2 jsou dva jazyky nad abecedou Σ. Konkatenace jazyků L_1 a L_2, L_1L_2 je definována jako $L_1L_2 = \{xy : x \in L_1 ax \in L_2\}$.
- Mocnina jazyka - Nechť L je jazyk nad abecedou Σ. Pro $i \geq 0$, i-tá mocnina jazyka L, L^i, je definována:
  - $L^0 = \{\epsilon\}$
  - pro $i \geq 1$:$L^i = LL^{(i-1)}$
- Iterace jazyka - Nechť L je jazyk nad abecedou Σ. Iterace jazyka L, L^* je definována $\sum_{i=0}^{\inf} L^i$ a pozitivní iterace jazyka L, L^+ je definována $\sum_{i=1}^{\inf} L^i$.
- Doplněk jazyka - Nechť L je jazyk nad abecedou Σ. Doplněk jazyka je $L_D = \Sigma^* \setminus L$

Reprezentace jazyka

V případě konečného jazyka lze jazyk reprezentovat výčtem. Tomuto zobrazení se říká konečná reprezentace. Tato reprezentace je naprosto dostatečná pro konečné jazyky, ovšem pro nekonečné jazyky problém reprezentace nabírá na komplikovanosti. Za předpokladu, že cílem by bylo vytvořit konečnou reprezentaci, která by měla být formálně vzato řetězcem symbolů, který budeme vhodně interpretovat, aby danému jazyku odpovídala nějaká konkrétní konečná reprezentace, a aby každé konkrétní reprezentaci odpovídal jeden jazyk. Jak výše napsané věty implikují pro každý jazyk nelze najít konečnou reprezentaci, protože množina všech řetězců nad abecedou je spočetně nekonečná, ale podmnožiny spočetně nekonečné množiny je nespočetná množina. Za předpokladu, že bychom měli být schopni zapsat libovolnou definici konečné reprezentace jako řetězec symbolů je triviální očekávat, že získáme spočetnou množinu konečných reprezentací. Tedy existuje více jazyků než konečných reprezentací. Za takové reprezentace můžeme uvažovat například gramatiky, které nejsou stěžejním pilířem této práce a konečné automaty na což se tato práce zaměřuje.

Gramatika

I když gramatiky nejsou hlavním cílem této práce je vhodné seznámit se základními definicemi a pojmy. Gramatiku lze definovat jako čtveřici:
$G = (N, \Sigma, R, S)$, kde:

- N je neprázdná konečná množina neterminálních symbolů
- Σ je neprázdná konečná množina terminálních symbolů, kde $N \cap \Sigma = \emptyset$
- R je konečná množina pravidel tvaru: $pa \rightarrow q$, kde $p, q \in Q, a \in \Sigma \cup \{\epsilon\}$
- S je počáteční neterminální symbol

2.5 Regulární výrazy (RV)

V teoretické informatice a teorii formálního jazyka je běžný jazyk formálním jazykem, který lze definovat regulárním výrazem v užším slova smyslu v teoretické informatice na rozdíl od mnoho moderních modelů regulárních výrazů, které jsou rozšířeny o funkce, které umožňují rozpoznávání nepravidelných jazyků.

Alternativně lze běžný jazyk definovat jako jazyk rozpoznávaný konečným automatem. Ekvivalence regulárních výrazů a konečných automatů je známá jako Kleeneova věta podle amerického matematika Stephena Colea Kleena. Stephen Cole Kleene proslul podílením se na vědeckých dílech jako A Theory of Positive Integers in Formal Logic, The foundations of intuitionistic mathematics, especially in relation to recursive functions, Introduction to Metamathematics nebo Mathematical Logic, což položilo základ pro dnešní regulární výrazy široce podporováné v programovacích jazycích, programech pro zpracování textu, pokročilých textových editorech a některých dalších programech. Podpora Regex je součástí standardní knihovny mnoha programovacích jazyků, včetně Java a Python, a je zabudována do syntaxe ostatních, včetně Perl a ECMAScript. Implementace funkce regulárního výrazu se často nazývá modul regulárního výrazu a pro opakované použití je k dispozici řada knihoven. V poslední době několik společností začalo nabízet hardware, FPGA, GPU s implementací PCRE kompatibilních s regex modely, které jsou rychlejší ve srovnání s implementacemi CPU.

Definice

Nechť Σ je abeceda. Regulární výrazy nad abecedou Σ a jazyky, které značí, jsou definovány následovně:

- $\emptyset$ je RV značící prázdnou množinu (prázdný jazyk)
- ϵ je RV značící jazyk $\{\epsilon\}$
- a, kde $a \in \Sigma$, je RV značící jazyk $\{a\}$

Operace nad RV

Nechť r a s jsou regulární výrazy značící po řadě jazyky L_r a L_s, potom:

- $(r.s)$ je RV značící jazyk $L = L_r L_s$
- $(r + s)$ je RV značící jazyk $L = L_r \cup L_s$
- (r^*) je RV značící jazyk $L = (L_r)^*$

2.6 Konečné automaty (KA)

S konečným automatem se setkává každý během každodenních činností všedního dne. Ať už se jedná o automat na kávu, infopanel pro turisty či automat na jízdenky. Například automat na jízdenky je jednoduchým konečným automatem, který lze lehce popsat. Stroj je vybaven otvorem pro mince, několika tlačítky pro výběr jízdenky, displejem ukazující množství přijaté hotovosti a systémem tisknoucí jízdní doklady. Interakce s automatem probíhá pomocí zmíněných součástí. Displej je jediný pasivní prvek, který pouze oznamuje hodnotu vložených peněz, případně stav jednotlivých komponent. I když se displej komunikace přímo

neúčastní, oznamuje nám momentální stav, ve kterém se automat právě nachází. Hodnota peněz je také hlavní a jediný prvek, který ovlivňuje další činnost automatu. Určuje, která tlačítka jsou aktivována a lze je použít pro nákup. Tlačítka jsou samozřejmě aktivována ve chvíli, kdy množství peněz v automatu převyšuje nebo je rovno hodnotě konkrétního jízdního dokladu pro dané tlačítko. Automat se na základě interakce s vnějšími vlivy (člověkem) chová podle množiny přesně definovaných pravidel. Například vhozením mince se navýší hodnota peněz v automatu nebo stisknutí tlačítka o nižší či ekvivalentní hodnotě než je množství peněz v automatu spustí tisknutí jízdního dokladu. Odebrání jízdního dokladu již za interakci považováno není, protože konečný automat tohoto automatu na jízdenky končí vrácením peněz a vrací se do původního stavu. Protože do automatu lze dostat jen určité množství peněz, lze z toho lehce odvodit, že takovýto automat má konečné množství stavů, které jsou zobrazitelné na displeji, a proto je konečným automatem.

Jako automat ovšem nemusí být jen stroje s výpočetními jednotkami. Automatem lze popsat mnoho systémů. Jedinými podmínkami je konečnost stavů, akcí a přechodů. Jako další příklad lze uvést deskovou hru Dáma. Konečná množina stavů jsou všechny kombinace rozložení sady hracích kamenů na šachovnici, kdy víme že je konečná díky konečnosti šachovnice a konečnému počtu hracích kamenů. Akce jsou definované jako pohyb hracích kamenů. Např. pohyb diagonálně dopředu o jedno pole nebo přeskok nepřátelského hracího kamenu, který se nachází o jedno diagonální pole před zvoleným kamenem. Množství akcí je ovlivněno aktuálním stavem, protože daná situace nemusí umožňovat všechny akce. Jako počáteční stav označíme výchozí rozložení kamenů na šachovnici a koncové stavy, kdy hrací sada kamenů již nemá umožněnou žádnou další akci.

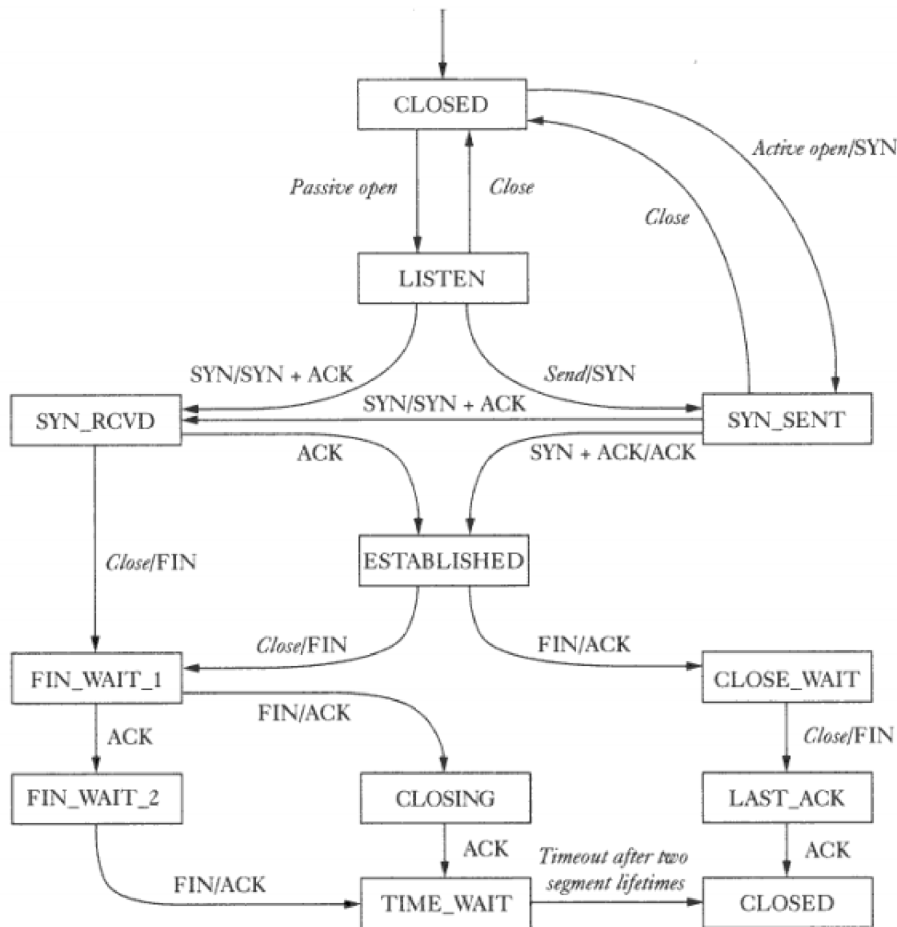
Abstraktním modelem těchto konečných stavových systémů jsou konečné automaty. Konečný automat je vybaven konečnou stavovou jednotkou, čtecí hlavou a páskou, na které je zapsané vstupní slovo. Na začátku výpočtu je hlava umístěna na prvním poli pásky zleva. Automat na základě přečteného symbolu a momentálního stavu svůj stav změní a posune čtecí hlavu o jedno pole vpravo. Výpočet končí, pokud se automat zasekne, nebo přečte celé vstupní slovo. Slovo zapsané na pásce je automatem akceptováno, pokud je celé přečteno a výsledný stav je některý z předem určených koncových stavů. Množina všech slov, která daný konečný automat M tvoří, označujeme jazyk akceptovaný automatem M.

Dalším příkladem může být stavový diagram, který popisuje část závazné definice protokolu TCP v RFC 793, který popisuje navázání a ukončení spojení viz Obrázek [2.1].

Konečný automat lze definovat matematickým popisem jako pětice:
$M = (Q, \Sigma, R, s, F)$, kde:

- Q je konečná neprázdná množina stavů
- Σ je vstupní abeceda
- R je konečná množina pravidel tvaru: $pa \to q$, kde $p, q \in Q, a \in \Sigma \cup \{\epsilon\}$
- $s \in Q$ je počáteční stav
- $F \subseteq Q$ je množina koncových stavů

Uvažujeme, že se KA nachází v některém ze stavů Q, kde tento stav bývá označován jako *konfigurace*. Pro každý okamžik uvažujeme, že se KA nachází právě v jednom stavu. Tento model je nejčastější a je přezdívaný **Rozpoznávací konečný automat**. Tento typ je také využíván při transformacích v této práci. Výstupem tohoto typu je odpověď true neboli vstupní řetězec akceptuje a naopak false, kdy je vstupní řetězec zamítnut.



Obrázek 2.1: KA popisující tcp spojení z [6]

Jako další typ můžeme chápat **klasifikační konečný automat**, který je pěticí: $M = (Q, \Sigma, R, s, \{Q_i\})$, kde:

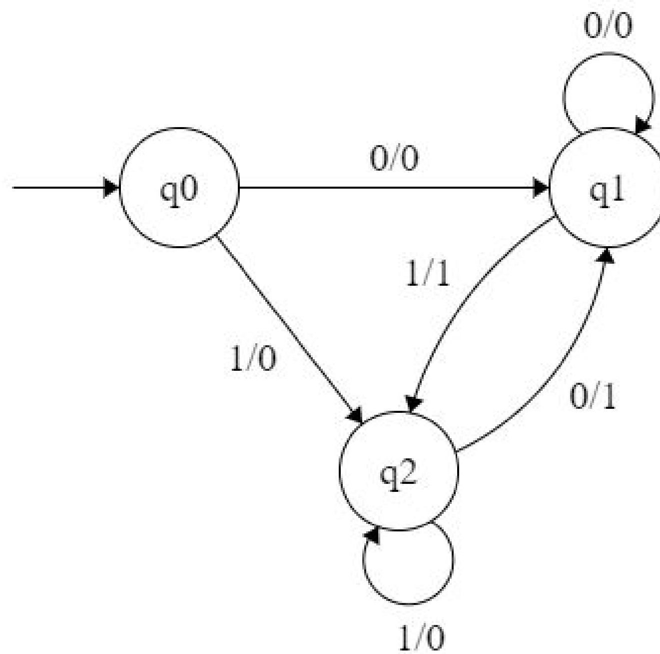
- Q je konečná neprázdná množina stavů
- Σ je vstupní abeceda
- R je konečná množina pravidel tvaru: $pa \to q$, kde $p, q \in Q, a \in \Sigma \cup \{\epsilon\}$
- $s \in Q$ je počáteční stav
- $\{Q_i\}$ je rozklad množiny stavů

Rozkladem množiny stavů $\{Q_i\}$ značí systém podmnožiny stavů Q, kde Q se nachází právě v jednom z prvku $\{Q_i\}$, kde prvky $\{Q_i\}$ odpovídají jedné klasifikační třídě. Přechodem do prvku $\{Q_i\}$ je jednoznačně rozhodnuta kategorizace vstupního řetězce do konkrétní třídy.

Automat s výstupní funkcí Mealyho typu je šesticí: $M = (Q, \Sigma, \Phi, R, s, \lambda)$, kde:

- Q je konečná neprázdná množina stavů
- Σ je vstupní abeceda
- Φ je výstupní abeceda
- R je konečná množina pravidel tvaru: $pa \rightarrow q$, kde $p, q \in Q, a \in \Sigma \cup \{\epsilon\}$
- $s \in Q$ je počáteční stav
- λ je výstupní funkce, ve tvaru $qa \rightarrow u$, kde $q \in Q$, $a \in \Sigma$ a $u \in \Phi$

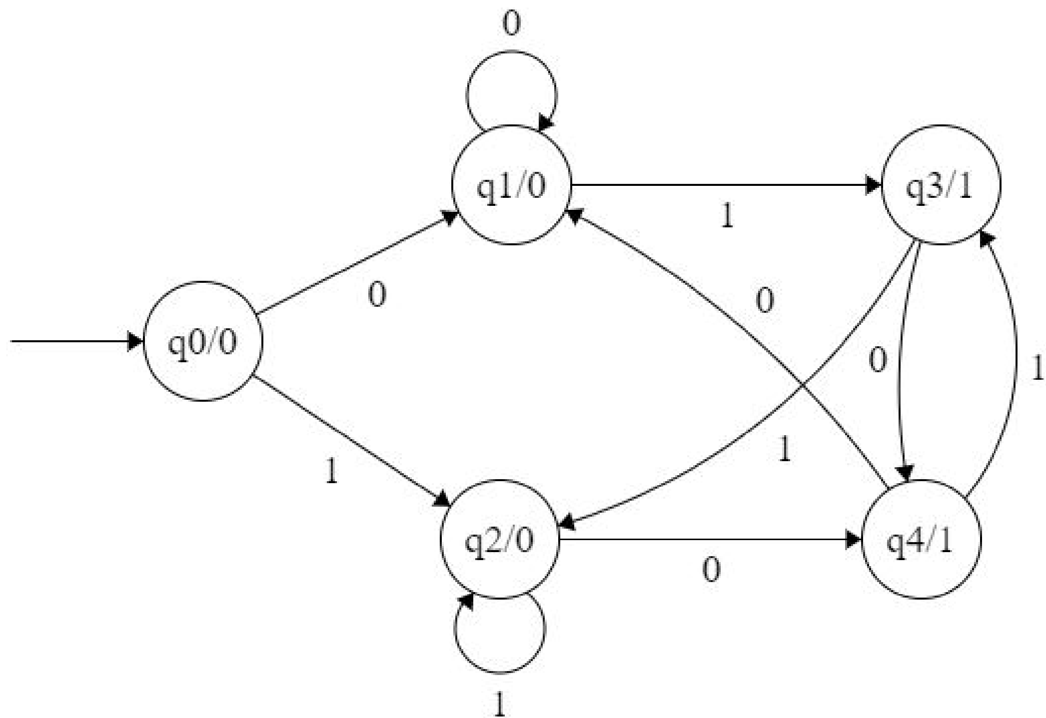
Příklad automatu Mealyho typu viz obrázek [2.2]



Obrázek 2.2: KA reprezentující automat Mealyho typu ze stránky [1]

Automat s výstupní funkcí Moorova typu je šesticí:
$M = (Q, \Sigma, \Phi, R, s, \lambda)$, kde:

- Q je konečná neprázdná množina stavů
- Σ je vstupní abeceda
- Φ je výstupní abeceda
- R je konečná množina pravidel tvaru: $pa \rightarrow q$, kde $p, q \in Q, a \in \Sigma \cup \{\epsilon\}$
- $s \in Q$ je počáteční stav
- λ je výstupní funkce, ve tvaru $q \rightarrow u$, kde $q \in Q$ a $u \in \Phi$



Obrázek 2.3: KA reprezentující automat Moorova typu ze stránky [1]

Příklad automatu Moorova typu viz obrázek [2.3].

Jak můžeme vypozorovat u Mealyho konečného automatu vstupní symbol je přesně definován aktuálním stavem a aktuálním symbolem. Naproti Moorův typ má závislost u výstupních symbolů pouze na aktuálním stavu. Tato rozdílná závislost se projevuje na délkách vstupních a výstupních řetězců z automatu. U Mealyho typu je délka vstupního řetězce ekvivalentní k délce výstupního řetězce, oproti Moorovu typu, kde je délka výstupního řetězce o jeden znak delší než délka vstupního řetězce.

Konfigurace

Konfigurace konečného automatu je reprezentace aktuálního stavu daného automatu a musí pokrývat aktuální stav stavového řízení a doposud nezpracovanou část vstupního řetězce. Lze ji vyjádřit jako prvek kartézského součinu uložených prvků, tedy $X \in Q \times \Sigma^*$. Lze konfiguraci zapisovat jako řetězec za předpokladu, že jsou množiny sousedících prvků disjunktní.

Přechod mezi konfiguracemi

Základní funkcí konečného automatu je přechod mezi konfiguracemi. Automat přejde z konfigurace $X_1 = (q1, xy)$, do konfigurace $X_2 = (q2, y)$, $x \in \Sigma \cup \{\epsilon\}, y \in \Sigma^*, q_1, q_2 \in Q$ pokud $R = (q_1, x, q_2) \in P$. Toto je zapisováno $X_1 \to X_2[R]$, případně $X_1 \to X_2$ pokud $x = \epsilon$ označujeme, že ze vstupní pásky není přečten žádný symbol a takový přechod označujeme

jako ϵ-přechod. Jinak dojde k přečtení symbolu x ze vstupní pásky, posunutí čtecí hlavy na symbol následující a stavové řízení se po vykonání přechodu nachází ve stavu q_2.

Přijímaný jazyk

KA M přijme vstupní řetězec, pokud existuje posloupnost přechodů $(s, x) \rightarrow^* (q_i, \epsilon), x \in \Sigma^*, q_* \in F$. To znamená, že k přijetí řetězce dojde, pokud aktuální stav automatu po přečtení posledního symbolu vstupního řetězce bude patřit mezi koncové stavy, nebo do takového stavu bude moci vykonat přechod nějakou posloupností ϵ-přechodů. Přijímaným jazykem $L(M)$ je označována množina všech řetězců, které daný automat přijímá.

Matematický popis výčtem je úplný. V praxi se ovšem využívají i další popisy. Prvním typem popisu je **tabulka**, kde řádky tabulky odpovídají stavům a sloupce znázorňují vstupní symboly, potom každé pole tabulky znázorňuje dvojici: stav a vstupní symbol, kde každé pole tabulky označuje unikátní dvojici. Dále je nezbytné označit počáteční a koncové stavy, které jinak z tabulky nejsou patrné. Pro zorientování v KA je tento popis nepraktický, ovšem pro implementaci KA je obvykle preferovanou variantou.

Jako další alternativu popisu můžeme využít **stavový strom**. Jak je u stromu zvykem, tvoříme ho z vrchu dolů, kde prvními vrcholy jsou počáteční stavy a následně každá hrana směřující dolů reprezentuje jeden vstupní symbol. Rozvoj stromu pokračuje, dokud se neobjeví stavy, které jsou již ve stromu zpracovány. Je vhodné koncové symboly označit šipkami vystupujícími ven ze stromu, aby nebylo nezbytné procházet celý strom pro zjištění koncových stavů.

Popisem zvoleným v této bakalářské práci je vybrán **stavový diagram**. Stavy jsou reprezentovány vrcholy grafu a přechody jsou značeny orientovanými hranami mezi stavy. Počáteční stav je značen vstupní šipkou, koncový stav je značen dvojitou hranou stavu (vrcholu grafu). Tato grafická reprezentace, je pro méně složité KA přehledná a proto je využita v této práci pro ukázku příkladů fungování algoritmů.

KA je většinou modelem reálného systému, který má v informatice širokého využití. Například:

- v překladačích
- modelů architektur softwarových komponent
- řešení úloh umělé inteligence
- návrhy sekvenčních logických obvodů

Je ovšem důležité vzít v potaz, že KA jsou omezeny množinou problémů, kterou lze řešit (relativně slabý výpočetní model) a existují obecnější alternativy jako například Turingův stroj.

Ekvivalentní modely

Dva modely pro popis formálních jazyků jsou ekvivalentní, pokud specifikují tentýž jazyk.

Kapitola 3

Transformace

Transformace jsou jsou specifikovány v publikaci [3]. Uvažujeme RV na abecedou Δ, který byl vytvořen dle definice v [2.5]. Aby byla transformace na KA provedena validně je nezbytné dodržet tyto pravidla:

1. Existuje KA ekvivalentní jednoduchým RV $\emptyset$, ϵ a $ain\Delta$.
2. Ke každé dvojici KA, I a M, musí existovat KA přijímající $L(I) \cup L(M)$ a $L(I)L(M)$.
3. Pro každý KA M musí existovat KA přijímající $L(M)^*$.

Kde první pravidlo je splněno na základě definic a 2. a 3. pravidlo je popsáno algoritmy následujícími v této kapitole.

3.1 Transformace - Sjednocení

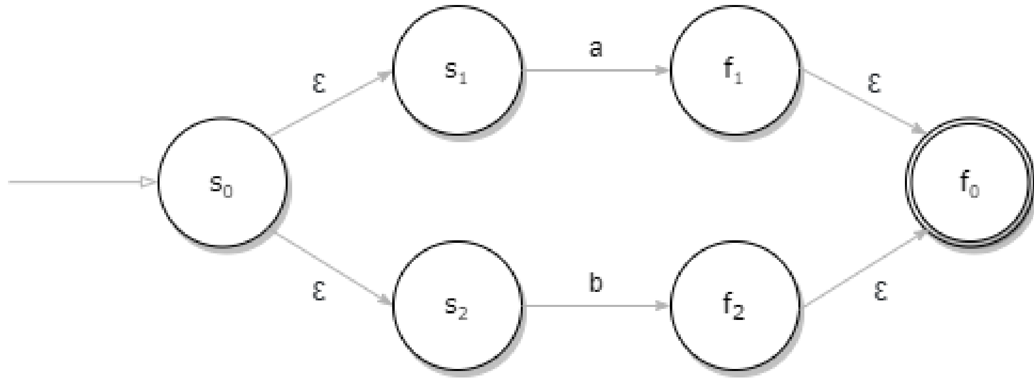
Algoritmus

- Vstup - 2 KA $M_1 = (\Sigma_1, R_1)$ a $M_2 = (\Sigma_2, R_2)$, kde $Q_1 \cap Q_2 = \emptyset$
- Výstup - KA $M_3 = (\Sigma_3, R_3)$, kde $L(M_3) = L(M_1) \cup L(M_2)$

Metoda

začni
polož $\Delta_3 = \Delta_1 \cup \Delta_2$
polož $Q_3 = Q_1 \cup Q_2$
jsou vytvořeny 2 nové stavy s_0 a f_0, kde s_0 je počáteční stav M_3
polož $F_3 = \{f_0\}$
polož $R_3 = R_1 \cup R_2 \cup \{s_0 \to s_1, s_0 \to s_2\} \cup \{p \to f_0 | p \in F_1 \cup F_2\}$
ukonči

Příklad



Obrázek 3.1: KA M_3 reprezentující RV $(a|b)$

3.2 Transformace - Konkatenace

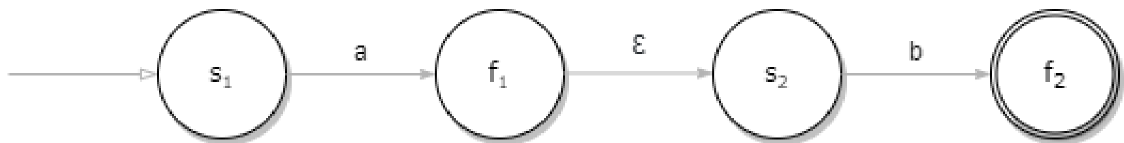
Algoritmus

- Vstup - 2 KA $M_1 = (\Sigma_1, R_1)$ a $M_2 = (\Sigma_2, R_2)$, kde $Q_1 \cap Q_2 = \emptyset$, $F_1 = \{f_1\}$, $F_2 = \{f_2\}$, f_1 a f_2 jsou oba koncové stavy
- Výstup - KA $M_3 = (\Sigma_3, R_3)$, kde $L(M_3) = L(M_1) \cup L(M_2)$

Metoda

začni
  polož $\Delta_3 = \Delta_1 \cup \Delta_2$
  polož $Q_3 = Q_1 \cup Q_2$
  polož $s_0 = s_1$
  polož $F_3 = \{f_2\}$
  polož $R_3 = R_1 \cup R_2 \cup \{f_1 \rightarrow s_2\}$
ukonči

Příklad



Obrázek 3.2: KA M_3 reprezentující RV (ab)

3.3 Transformace - Iterace

Algoritmus

- Vstup - KA $M_1 = (\Sigma_1, R_1)$, kde $F_1 = \{f_1\}$ a f_1 je koncový stav
- Výstup - KA $M_2 = (\Sigma_2, R_2)$, kde $L(M_2) = L(M_1)^*$

Metoda

začni

polož $\Delta_2 = \Delta_1$

polož $Q_2 = Q_1$

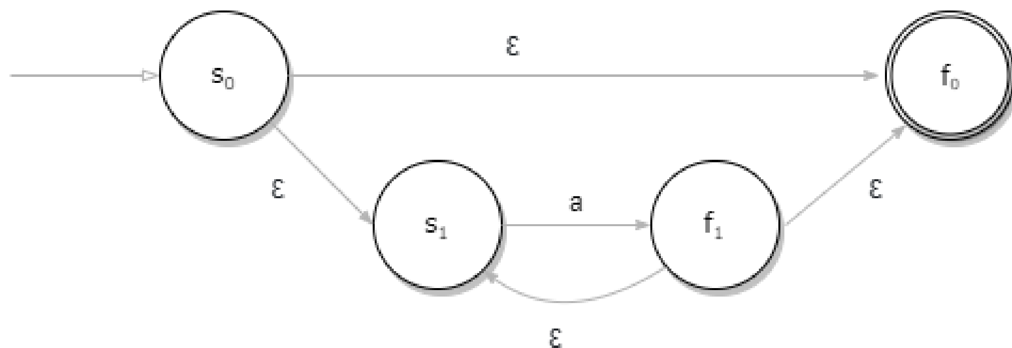
jsou vytvořeny 2 nový stav s_0 a f_0, kde s_0 je počáteční stav M_2

polož $F_2 = \{f_0\}$

polož $R_2 = R_1 \cup \{s_0 \to s_1, s_0 \to f_0, f_1 \to s_1, f_1 \to f_0\}$

ukonči

Příklad



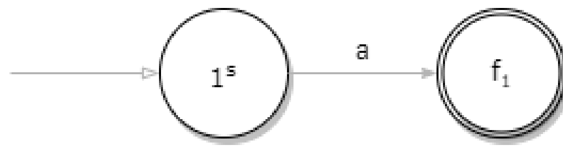
Obrázek 3.3: KA M_2 reprezentující RV $(a)^*$

3.4 Příklad užití

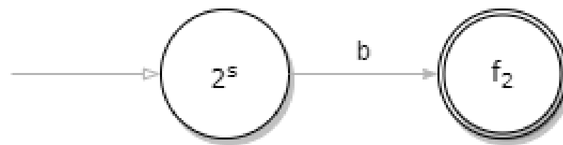
V příkladu užití transformujeme RV $((a|b)^*)|(ca)$ na KA

	RV	KA
1	a	M_1
2	b	M_2
3	c	M_3
3	$a\|b$	M_4
4	$(a\|b)^*$	M_5
5	ca	M_6
7	$((a\|b)^*)\|(ca)$	M_7

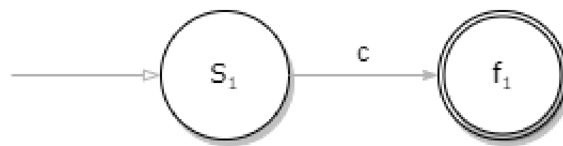
Tabulka 3.1: Tabulka pro RV a KA pro převod



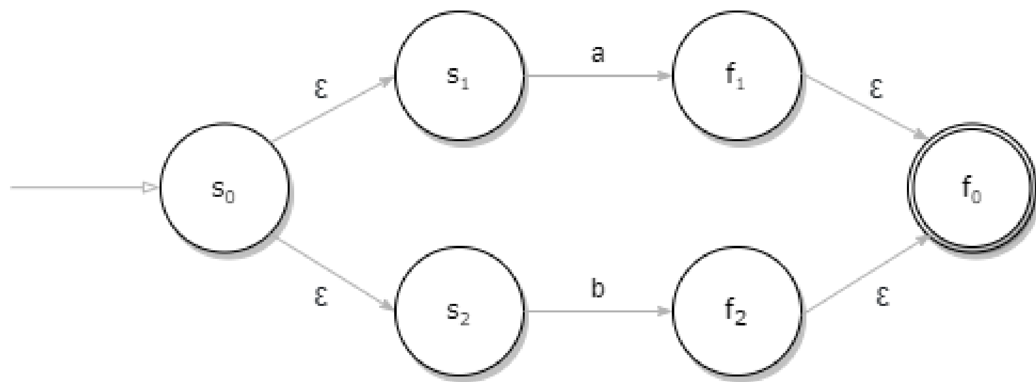
Obrázek 3.4: KA M_1 reprezentující RV a



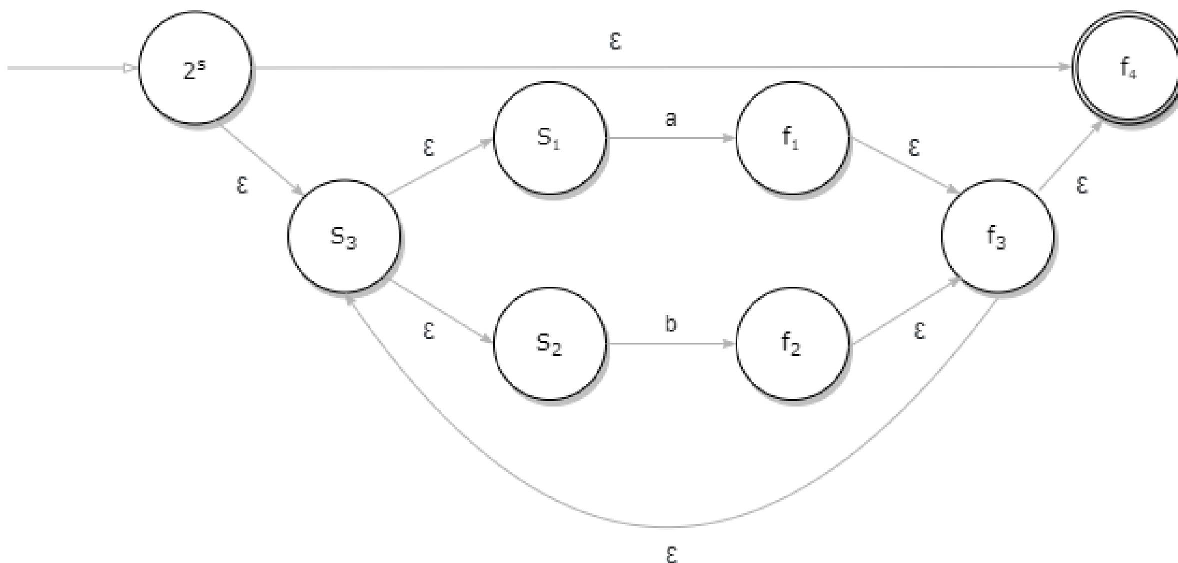
Obrázek 3.5: KA M_2 reprezentující RV b



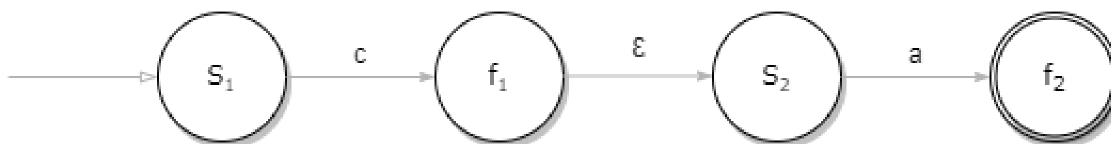
Obrázek 3.6: KA M_3 reprezentující RV c



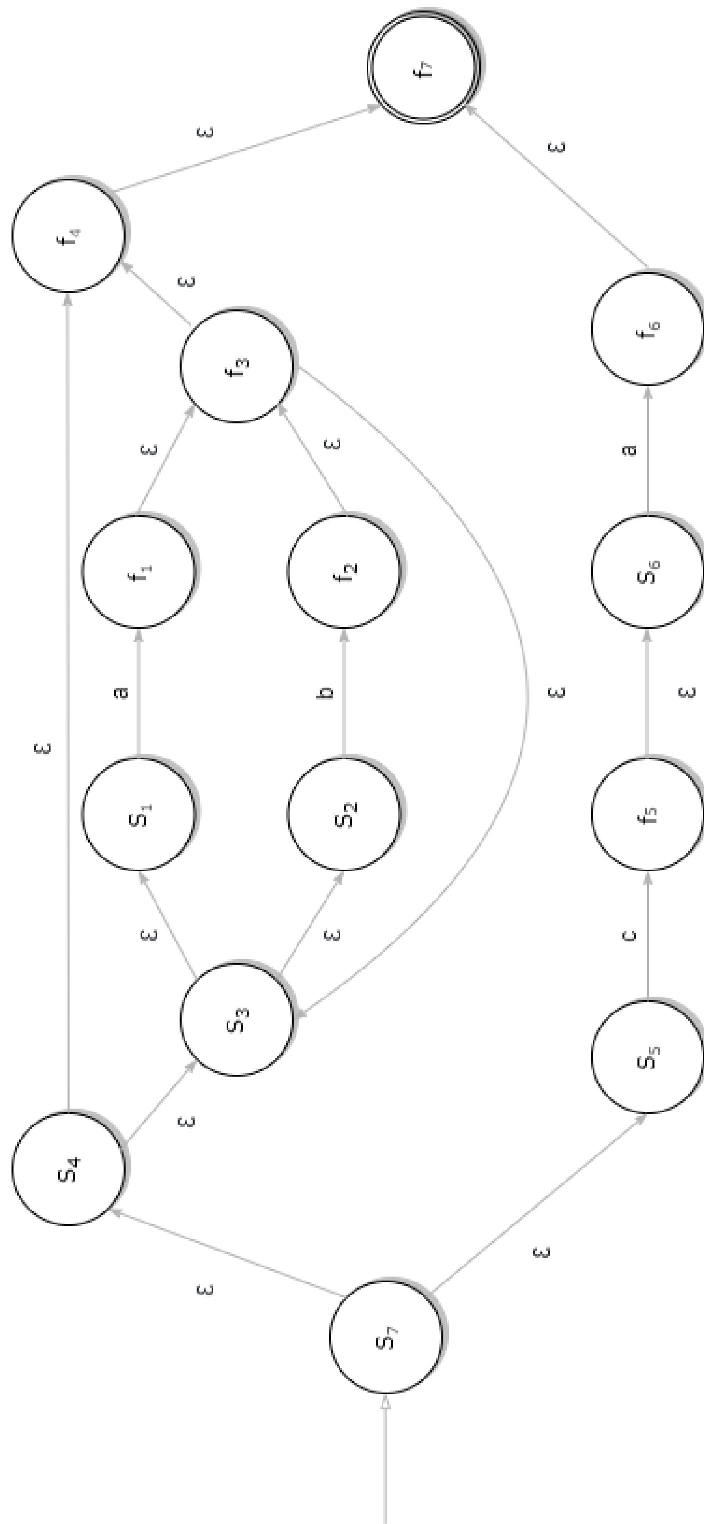
Obrázek 3.7: KA M_4 reprezentující RV $a|b$



Obrázek 3.8: KA M_5 reprezentující RV $(a|b)^*$



Obrázek 3.9: KA M_6 reprezentující RV ca



Obrázek 3.10: KA M_7 reprezentující RV $((a|b)^*)|(ca)$

Kapitola 4

Alternativní transformace

Návrh algoritmu, lze provést několika různými způsoby, musí být ovšem dodržena ekvivalence přijímaných jazyků. Při návrhu algoritmů pro alternativní transformace je brána v potaz přehlednost a zároveň snížení počtu stavů a ϵ přechodů, které v příliš vysoké míře mohou snižovat rychlost zpracování, tak i přehlednost při složitějších přijímaných jazycích. To lze vypozorovat z kapitol Příklad užití u finálních KA a pro zavedený KA M_7 [3.10] a nově navržený KA M_7 [4.10]. Důkazy lze porovnat s důkazy v publikaci [3].

4.1 Transformace - Sjednocení

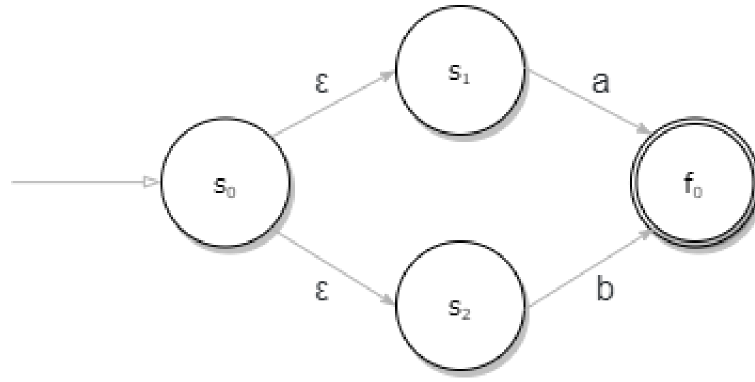
Teze: Pro každý KA M, který existuje v rámci sjednocení s koncovým stavem f_m, je stav f_m zrušitelný.
Dle zavedené definice transformace je zaveden společný koncový stav f_0, a pravidlo $R : f_m \to f_0$ pro každý KA M. Pokud stav f_m má jako výstup pouze pravidlo R, může být tento stav zrušen za předpokladu, že všechna pravidla $XY \to f_m$, kde $X \in Q$ a $Y \in \Sigma$ jsou nahrazena $XY \to f_0$.

Algoritmus

- Vstup - 2 KA $M_1 = (\Sigma_1, R_1)$ a $M_2 = (\Sigma_2, R_2)$, kde $Q_1 \cap Q_2 = \emptyset$
- Výstup - KA $M_3 = (\Sigma_3, R_3)$, kde $L(M_3) = L(M_1) \cup L(M_2)$

Metoda

začni
  polož $\Delta_3 = \Delta_1 \cup \Delta_2$
  polož $Q_3 = (Q_1 - F_1) \cup (Q_2 - F_2)$
  jsou vytvořeny 2 nové stavy s_0 a f_0, kde s_0 je počáteční stav M_3
  polož $F_3 = \{f_0\}$
  polož R_t množinu pravidel, kde $Q_1 \to F_1$ je nahrazena $Q_1 \to f_0$ a $Q_2 \to F_2$
    nahrazena $Q_2 \to f_0$
  polož $R_3 = R_1 \cup R_2 \cup \{s_0 \to s_1, s_0 \to s_2\} \cup R_t$
ukonči



Obrázek 4.1: KA M_3 reprezentující RV $(a|b)$

Příklad

Lemma

Jsou dány dva KA $M_1 = (Q_1, \Sigma_1, R_1, s_1, F_1)$ a $M_2 = (Q_2, \Sigma_2, R_2, s_2, F_2)$ potom algoritmus [4.1] validně vytváří KA $M_3 = (Q_3, \Sigma_3, R_3, s_3, F_3)$, kde $L(M_3) = L(M_1) \cup L(M_2)$.

Důkaz

Konečnost
Algoritmus je vždy konečný, protože konstrukce z Q, Σ, R obsahuje pouze konečné množiny.

Správnost
Abychom dokázali, že $L(M_3) = L(M_1) \cup L(M_2)$, demonstrujeme, že M_3 přijímá slovo $w \in \Sigma^*$ pokud M_1 a M_2 přijímá w. První následující výrok musí být ustanoven.

Výrok
Pro každé $w \in \Sigma^*$,

$$qw \rightarrow^* f_0 \text{ v } M_3 \text{ když } qw \rightarrow^* f_i \text{ v } M_i$$

kde $q \in Q_i$ a $f_i \in F_i$ pro $i = \{0, 1\}$.

Důkaz
Když
Indukcí na j tato část důkazu demonstruje, že každé $w \in \Sigma^*$

$$qw \rightarrow^j f_0 \text{ v } M_3 \text{ implikuje } qw \rightarrow^* f_i \text{ v } M_i$$

kde $q \in Q_i$ a $f_i \in F_i$ pro $i = \{0, 1\}$.

Základ
Nechť $j = 0$. To znamená $qw \rightarrow^0 f_0$ v M_3. V tomto bodě $q = f_0$ a $w = \epsilon$. Stejně pro $qw \rightarrow^0 f_i$ v M_i, kde $q \in Q_i$ a $f_i \in F_i$ pro $i = \{0, 1\}$ je $q = f_i$ a $w = \epsilon$, takže základ z *Když* části platí.

Indukční hypotéza

Předpokládejme, že *když* část z výroku platí pro všechny $j \leq n$ pro všechna $n \geq 0$.

Indukční krok
Zvažme výpočet

$$qw \rightarrow^{n+1} f_0$$

v M_3. Vyjádříme tento výpočet jako

$$\begin{aligned} qw &\rightarrow pv[qa \rightarrow p] \\ &\rightarrow^n f_0 \end{aligned}$$

kde $qa \rightarrow p \in R_3$ a $w = av$ pro nějaké $a \in \Sigma \cup \{\epsilon\}$ Pozorujeme, že $pv \rightarrow^n f_0$ v M_3. Indukční hypotézou

$$pv \rightarrow^* f_i \text{ v } M_i$$

kde $p \in Q_i$ a $f_i \in F_i$ pro $i = \{0,1\}$. Jak $Q_0 \cap Q_1 = \emptyset, q \in Q_i, qa \rightarrow p \in R_i - R_{1-i}$, a

$$\begin{aligned} qw &\rightarrow pv[qa \rightarrow p] \\ &\rightarrow^n f_i \end{aligned}$$

v M_i. To je

$$qw \rightarrow^* f_i$$

kde $q \in Q_i$ a $f_i \in F_i$ pro $i = \{0,1\}$. Proto, *když* část důkazu platí.

Nechť $i \in \{0,1\}$. Uvažujme $w \in \Sigma_i^*$. Očividně

$$w \in L(M_i)) \text{ když } s_i w \rightarrow^* f_i \text{ v } M_i$$

kde $f_i \in F_i$. Dle předchozího výroku $q = s_i$,

$$s_i w \rightarrow^* f_i \text{ v } M_i \text{ když } s_i w \rightarrow^* f_0 \text{ v } M_3.$$

Očividně

$$s_i w \rightarrow^* f_0 \text{ v } M_3 \text{ když } sw \rightarrow s_i w \rightarrow^* f_0 \text{ v } M_3.$$

Každý přijímající výpočet v M_3 má formu $sw \rightarrow s_i w \rightarrow^* f_0$, tak

$$sw \rightarrow s_i w \rightarrow^* f_0 \text{ v } M_3 \text{ když } w \in L(M).$$

Tyto ekvivalence implikují

$$w \in L(M_i) \text{ když } w \in L(M).$$

kde $i \in \{0,1\}$. Tudíž

$$L(M) = L(M_0) \cup L(M_1)$$

Proto Lemma platí.

Shrnutí

Jak můžeme pozorovat nově navržený algoritmus pro transformaci regulárního výrazu v základní podobě oproti algoritmu uvedenému v předchozí kapitole snižuje počet stavů o dva a počet pravidel o dvě pravidla s ϵ přechodem na úkor jednoho průchodu skrz pravidla automatu a jejich úpravou. Tento průchod zvyšuje náročnost algoritmu ovšem za cenu přehlednějšího a méně rozsáhlého automatu.

4.2 Transformace - Konkatenace

Teze: Pro každý KA M, který existuje v rámci konkatenace a není posledním KA vstupujícím do konkatenace, s koncovým stavem f_m a počátečním stavem s_0, je stav f_m zrušitelný. Dle zavedené definice transformace je koncovým stavem konkatenace koncový stav posledního KA vstupujícího do konkatenace. A je vytvořeno pravidlo $R : f_m \to s_0$ pro každý KA M mimo posledního, kde s_0 je počátečním stavem dalšího KA M. Pokud stav f_m má jako výstup pouze pravidlo R, může být tento stav zrušen za předpokladu že všechna pravidla $XY \to f_m$, kde $X \in Q$ a $Y \in \Sigma$ jsou nahrazena $XY \to s_0$.

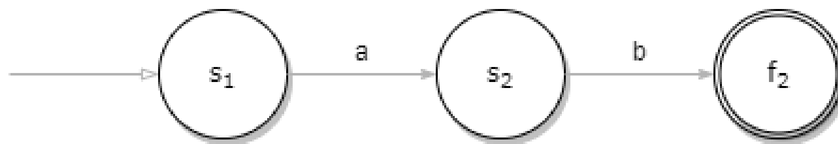
Algoritmus

- Vstup - 2 KA $M_1 = (\Sigma_1, R_1)$ a $M_2 = (\Sigma_2, R_2)$, kde $Q_1 \cap Q_2 = \emptyset$, $F_1 = \{f_1\}$, $F_2 = \{f_2\}$, f_1 a f_2 jsou oba koncové stavy
- Výstup - KA $M_3 = (\Sigma_3, R_3)$, kde $L(M_3) = L(M_1) \cup L(M_2)$

Metoda

začni
  polož $\Delta_3 = \Delta_1 \cup \Delta_2$
  polož $Q_3 = (Q_1 - F_1) \cup Q_2$
  polož $s_0 = s_1$
  polož $F_3 = F_2$
  polož R_t množinu pravidel, kde $Q_1 \to F_1$ je nahrazena $Q_1 \to s_2$
  polož $R_3 = R_1 \cup R_2 \cup R_t$
ukonči

Příklad



Obrázek 4.2: KA M_3 reprezentující RV (ab)

Lemma

Jsou dány dva KA $M_1 = (Q_1, \Sigma_1, R_1, s_1, F_1)$ a $M_2 = (Q_2, \Sigma_2, R_2, s_2, F_2)$ potom algoritmus [4.2] validně vytváří KA $M_3 = (Q_3, \Sigma_3, R_3, s_3, F_3)$, kde $L(M_3) = L(M_1)L(M_2)$.

Důkaz

Konečnost
Algoritmus je vždy konečný, protože konstrukce z Q, Σ, R obsahuje pouze konečné množiny.

Správnost
Abychom dokázali, že $L(M_3) = L(M_1)L(M_2)$, demonstrujeme, že M_3 přijímá slovo $w \in \Sigma^*$

pokud M_1 s M_2 přijímá w. První následující výrok musí být ustanoven.

Výrok
Pro každé $w \in \Sigma^*$,

$$q_1 w_1 \to^* q_2 w_2 \to^* f_0 \text{ v } M_3 \text{ když } q_1 w_1 \to^* f_1 \text{ v } M_1 \text{ a } q_2 w_2 \to^* f_2 \text{ v } M_2$$

kde $w = w_1 w_2$

Důkaz
Když
Indukcí na j, tato část důkazu demonstruje, že každé $w \in \Sigma^*$

$$q_1 w_1 \to^j q_2 w_2 \to^j f_0 \text{ v } M_3 \text{ když } q_1 w_1 \to^j f_1 \text{ v } M_1 \text{ a } q_2 w_2 \to^j f_2 \text{ v } M_2$$

Základ
Nechť $j = 0$. To znamená $q_1 w_1 \to^0 q_2 w_2 \to^0 f_0$ v M_3. V tomto bodě $q = f_0$ a $w = \epsilon$. Stejně pro $q_1 w_1 \to^0 f_1$ v M_1, kde $q = f_1$ a $w = \epsilon$ a pro $q_2 w_2 \to^0 f_2$ v M_2, kde $q = f_2$ a $w_2 = \epsilon$ a proto $w = w_1 w_2 = \epsilon$, takže základ z *Když* části platí.

Indukční hypotéza
Předpokládejme, že *když* část z výroku platí pro všechny $j \leq n$ pro všechna $n \geq 0$.

Indukční krok
Zvažme výpočet

$$q_1 w_1 \to^{n+1} q_2 w_2 \to^{n+1} f_0$$

v M_3. Vyjádříme tento výpočet jako

$$\begin{gathered} q_1 w_1 \to p_1 v_1 [q_1 a \to p_1] \\ \to^n q_2 w_2 \to p_2 v_2 [q_2 a \to p_2] \\ \to^n f_0 \end{gathered}$$

kde $q_1 a \to p_1 \cap q_2 a \to p_2 \in R_3$ a $w = a v_1 a v_2$ pro nějaké $a \in \Sigma \cup \{\epsilon\}$ Pozorujeme, že $p_1 v_1 \to^n p_2 v_2 \to^n f_0$ v M_3. Indukční hypotézou

$$p_1 v_1 \to^* f_1 \text{ v } M_1 \text{ a } p_2 v_2 \to^* f_2 \text{ v } M_2$$

kde $p_1 \in Q_1$ a $f_1 \in F_1$ a $p_2 \in Q_2$ a $f_2 \in F_2$. Kde $q \in Q_1 \cup Q_2, qa \to p \in R_1 \cup R_2$, a

$$\begin{gathered} q w_i \to p v [qa \to p] \\ \to^n f_i \end{gathered}$$

v M_i pro $i = \{0, 1\}$. To je

$$qw \to^* f_i$$

kde $q \in Q_i$ a $f_i \in F_i$ pro $i = \{0, 1\}$. Proto, *když* část důkazu platí.

Nechť $i \in \{0, 1\}$. Uvažujme $w \in \Sigma_i^*$. Jasně,

$$w_i \in L(M_i)) \text{ když } s_i w \rightarrow^* f_i \text{ v } M_i$$

kde $f_i \in F_i$. Dle předchozího výroku $q = s_i$,

$$s_i w_i \rightarrow^* f_i \text{ v } M_i \text{ když } s_i w_1 \rightarrow^* s_{i+1} w_2 \rightarrow^* f_0 \text{ v } M_3.$$

Každý přijímající výpočet v M_3 má formu $sw_1 \rightarrow^* s_i w_1 \rightarrow^* s_{i+1} w_2 \rightarrow f_0$, tak

$$sw_1 \rightarrow^* s_i w_1 \rightarrow^* s_{i+1} w_2 \rightarrow f_0 \text{ v } M_3 \text{ když } w \in L(M_3).$$

Tyto ekvivalence implikují

$$L(M_3) = L(M_1)L(M_2)$$

Proto Lemma platí.

Shrnutí

Při konkatenaci nově navržený algoritmus pro transformaci z regulárního výrazu v základní podobě oproti algoritmu uvedenému v předchozí kapitole snižuje počet stavů o jeden a počet pravidel o jedno pravidlo s ϵ přechodem. Díky této eliminaci tohoto pravidla jsou eliminovány všechny pravidla s ϵ přechodem a tím i všechny pravidla obsahující ϵ za předpokladu, že jednotlivé automaty ke konkatenaci neobsahují taková pravidla. Složitost algoritmu se zvyšuje o jeden průchod skrz pravidla prvního automatu vstupujícího do transformace a jejich úpravou. Tento průchod opět zvyšuje náročnost algoritmu za cenu přehlednějšího a méně rozsáhlého automatu.

4.3 Transformace - Iterace

Teze: Pro každý KA M, nad kterým je prováděna iterace, s koncovým stavem f_0 a počátečním stavem s_1, není potřeba zavádět nový počáteční stav.
Dle zavedené definice transformace je vytvořeno pravidlo $R : s_0 \rightarrow s_1$, kde s_0 je počátečním stavem a pravidlo $S : s_0 \rightarrow f_0$. Pokud stav s_0 má jako výstup pouze pravidla R a S, může být tento stav zrušen za předpokladu že vznikne pravidlo $S : s_1 \rightarrow f_0$.

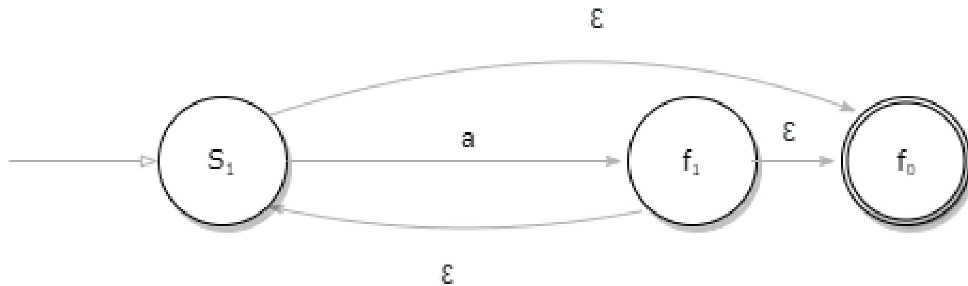
Algoritmus

- Vstup - KA $M_1 = (\Sigma_1, R_1)$, kde F_1 je množina koncových stavů
- Výstup - KA $M_2 = (\Sigma_2, R_2)$, kde $L(M_2) = L(M_1)^*$

Metoda

začni
  polož $\Delta_2 = \Delta_1$
  polož $Q_2 = Q_1$
  je vytvořen 1 nový stav f_0, kde f_0 je koncový stav M_2
  polož $F_2 = \{f_0\}$
  polož $R_2 = R_1 \cup \{F_1 \rightarrow f_0, s_1 \rightarrow f_0, F_1 \rightarrow s_1\}$
ukonči

Příklad



Obrázek 4.3: KA M_2 reprezentující RV $(a)^*$

Lemma

Je dán KA $M_1 = (Q_1, \Sigma_1, R_1, s_1, F_1)$ potom algoritmus [4.3] validně vytváří KA $M_2 = (Q_2, \Sigma_2, R_2, s_2, F_2)$, kde $L(M_2) = L(M_1)^*$.

Důkaz

Konečnost
Algoritmus je vždy konečný, protože konstrukce z Q, Σ, R obsahuje pouze konečné množiny.

Výrok
Pro každé $w \in \Sigma^*$,

$$s_1 w \rightarrow^* f_1$$
$$\rightarrow f_2\{f_1 \rightarrow f_2\} \text{ v } M_2$$
$$\text{když}$$
$$\text{existuje } n \leq 1 \text{ a } w_1, ..., w_n \in \Sigma^* \text{ tak}$$
$$w = w_1...w_n \text{ a } s_1 w_i \rightarrow^* f_1 \text{v } M_1 \text{ pro každé } i = 1, ..., n.$$

Důkaz
Když
Když část tohoto výroku stanovuje pro všechna $n \leq 1$,

$$\text{Když } s_1 w \to^* f_1 \text{ v } M_1,\ 1 \leq i \leq m \text{ a } w = w_1 ... w_n$$
$$\text{potom } s_1 w \to^* f_1 \to f_2 \{f_1 \to f_2\} \text{ v } M_2$$

Tato implikace je dokázána indukcí na m.

Základ
Nechť $n = 0$. To znamená $s_1 w \to^0 f_1 \to f_2$ v M_2. V tomto bodě $s_1 = f_1$ a $w = \epsilon$. Stejně pro $s_1 w \to^0 f_1 \to f_2$ v M_1 je $s_1 = f_1$ a $w = \epsilon$, takže základ z *Když* části platí.

Indukční hypotéza
Předpokládejme, že *Když* část z výroku platí pro všechny $n = 1, ...m$ pro některá $m \geq 1$.

Indukční krok
Nechť

$$s_1 w_i \to^* f_1 \text{ v } M_1, \text{ pro } i = 1, ..., m+1, \text{ a } w = w_1 ... w_m w_{m+1}$$

Uvažujme

$$\text{Když } s_1 w_i \to^* f_1 \text{ v } M_1,\ 1 \leq i \leq m, \text{ a } w^{'} = w_1 ... w_m$$

Indukční hypotézou

$$s_1 w^{'} \to^* f_1$$
$$\to f_2 \{f_1 \to f_2\} \text{ v } M_2$$

Jako $s_0 w_{m+1} \to^* f_1$ v M_1, $R_1 \subseteq R_2$ a $f_1 \to f_2 \in R_2$, M_2 také počítá

$$s_1 w_{m+1} \to^* f_1$$
$$\to f_2 \{f_1 \to f_2\}$$

Proto

$$s_1 w^{'} w_{m+1} \to^* f_1 w_{m+1}$$
$$\to s_1 w_{m+1} \{f_1 \to s_1\}$$
$$\to^* f_1$$
$$\to f_2 \{s_1 \to f_2$$

v M_2. Jako $w^{'} = w_1 ... w_m$ a tím pádem, $w = w^{'} w_{m+1}$,

$$s_1 w \to^* s_1$$
$$\to f_2 \{s_1 \to f_2$$

v M_2, proto výraz platí.

Nechť $w \in \Sigma^*$. Pozorujme, že M_2 akceptuje w skrz formu výpočtu

$$s_1 w \rightarrow^* f_1$$
$$\rightarrow f_2\{f_1 \rightarrow f_2$$

Formálněji

$$w \in L(M_2), \text{ když } s_1 w \rightarrow^* f_1 \rightarrow f_2\{f1 \rightarrow f2\} \text{ v } M_2$$

Podle předchozího výroku

$$s_1 w \rightarrow^* f_1 \rightarrow f_2 \text{ v } M_2$$
$$\text{když}$$
$$s_1 w_i \rightarrow^* f_1 \text{ v } M_1 \text{ a } w = w_1 ... w_n$$

kde $i = 1, ..., n$, pro některé $n \geq 1$. Očividně,

$$s_1 w_i \rightarrow^* f_1 \text{ v } M_1 \text{ když } w_i \in L(M_1)$$

Tudíž

$$w \in L(M2) \text{ když } w = w_1 ... w_n \text{ s } w_i \in L(M_2)$$

kde $i = 1, ..., n$, pro některé $n \geq 1$. Proto,

$$L(M_2) - \{\epsilon\} = L(M_1)^+$$

Použitím $s \rightarrow f$, M_2 akceptuje ϵ, takže $\epsilon \in L(M)$. To shrnuje že

$$L(M_2) = L(M_1)^*$$

Proto Lemma platí.

Shrnutí

Nově navržený algoritmus pro iteraci na transformaci z regulárního výrazu v základní podobě oproti algoritmu uvedenému v předchozí kapitole snižuje počet stavů o jeden a počet pravidel o jedno pravidlo s ϵ přechodem. Složitost algoritmu popisující iteraci se nezvyšuje naopak je ušetřena tvorba stavu a pravidla, která je pro náročnost skoro bezvýznamná, přesto se jedná o jednodušší algoritmus. Tento algoritmus je z toho důvodu méně náročný a generuje méně rozsáhlý automat.

4.4 Srovnání

Pro zhodnocení navržených algoritmů je vhodné provést srovnání oproti zavedeným algoritmům. V této práci se srovnání provede vůči zavedené transformaci v kapitole [3]. Cílem implementace bude umožnit toto srovnání jakémukoliv algoritmu díky podstatě aplikace, která poskytuje výčet stavů a pravidel, kterými umožňuje srovnání. Pro srovnání v rámci této kapitoly následující podkapitoly [4.5] je náhodně zvolen regulární výraz $((a|b)^*)|(ca)$

na kterém v rámci jednotlivých kroků lze pozorovat rozdílnost produkovaných konečných automatů.

Dle tabulky [4.1] a [3.1] můžeme porovnávat jednotlivé kroky, na kterém budeme odkazovat prvním sloupcem tabulky a k nim budeme přiřazovat automaty v grafické podobě, které jsou formou obrázků reprezentované v následující kapitole a jejím porovnáním sledovat rozdílnost mezi transformací a alternativní transformací.

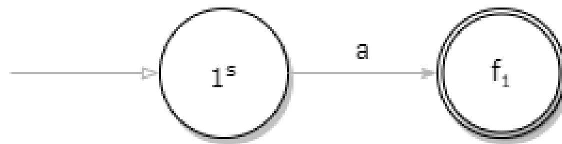
V kroku 4 můžeme vidět základní sjednocení, které v zavedených transformacích [3.8] konstruuje 6 stavů a 6 pravidel. Oproti tomu alternativní transformace [4.7] vytváří 4 stavy a 4 přechody. Tedy jak je uvedeno o 2 stavy a 2 pravidla méně. V kroku 5 lze pozorovat regulární výraz, který se skládá ze sjednocení a iterace, kde dle zavedených transformací [3.9] obsahuje 8 stavů a 10 pravidel a oproti tomu konečný automat vzniklý alternativní transformací [4.8], který obsahuje 5 stavů a 7 pravidel, což je o 3 stavy a 3 pravidla méně. Jako poslední lze uvést finální konečný automat popisující celý regulární výraz $((a|b)^*)|(ca)$, který po aplikaci zavedené transformace [3.10] obsahuje 14 stavů a 17 pravidel a konečný automat vytvořený alternativní transformací [4.10] obsahující 8 stavů a 11 pravidel, což je rozdíl 6 stavů a 6 pravidel a to už není zanedbatelný rozdíl.

4.5 Příklad užití

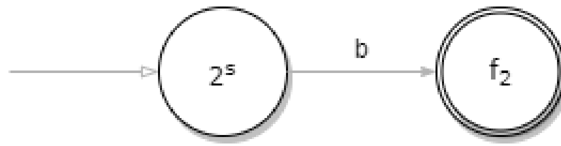
V příkladu užití transformujeme RV $((a|b)^*)|(ca)$ na KA, který je identický s RV v předchozí kapitole.

	RV	KA
1	a	M_1
2	b	M_2
3	c	M_3
4	$a|b$	M_4
5	$(a|b)^*$	M_5
6	ca	M_6
7	$((a|b)^*)|(ca)$	M_7

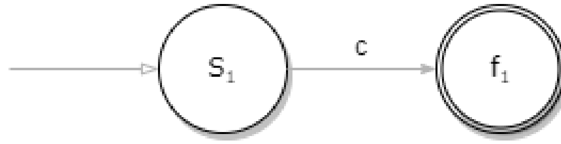
Tabulka 4.1: Tabulka pro RV a KA pro převod dle alternativních pravidel



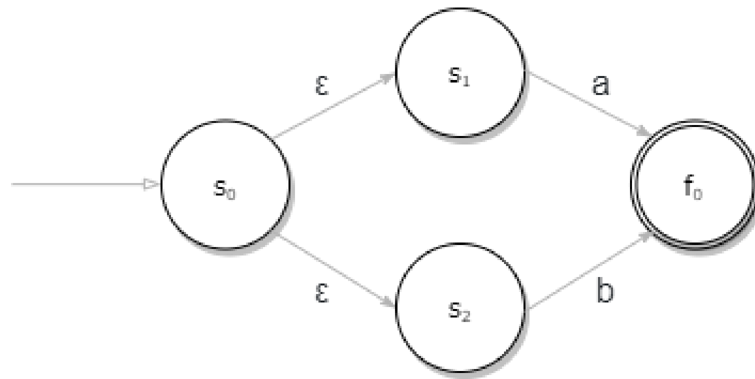
Obrázek 4.4: KA M_1 reprezentující RV a



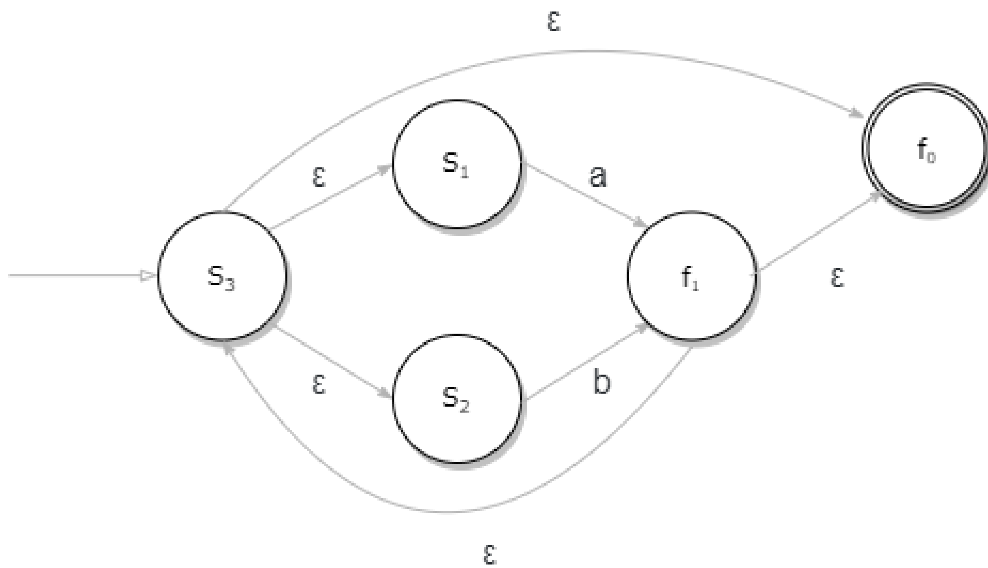
Obrázek 4.5: KA M_2 reprezentující RV b



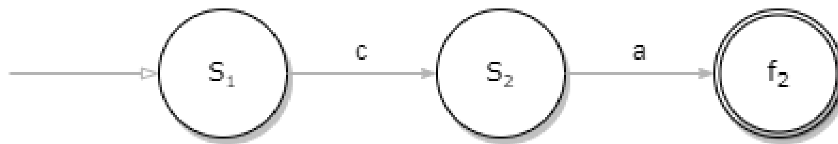
Obrázek 4.6: KA M_3 reprezentující RV c



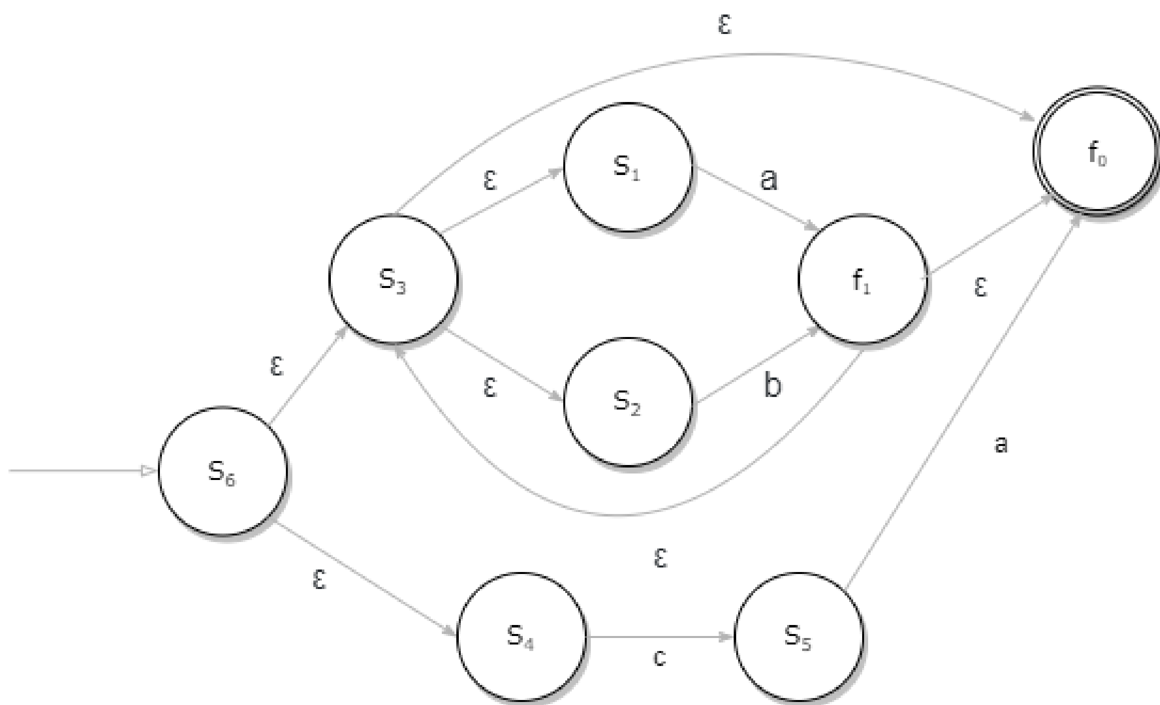
Obrázek 4.7: KA M_4 reprezentující RV $a|b$



Obrázek 4.8: KA M_5 reprezentující RV $(a|b)^*$



Obrázek 4.9: KA M_6 reprezentující RV ca



Obrázek 4.10: KA M_7 reprezentující RV $((a|b)^*)|(ca)$

Kapitola 5

Implementace

V předchozí kapitole jsou představeny a popsány tři algoritmy pro tvorbu konečných automatů z regulárních výrazů.

5.1 Návrh implementace

Je potřeba implementovat tyto algoritmy do programu, který bude umožňovat výše zmíněnou konverzi. Formu aplikace jsem zvolil konzolové rozhraní, které pracuje s jednotlivými algoritmy. Jedná se o rozhraní, které umožňuje vstup regulárního výrazu do programu formou argumentu, což umožňuje lepší návaznost s různými jinými aplikacemi nebo soubory, které takový regulární výraz obsahují nebo tvoří.

5.2 Implementační jazyk

Před samotnou implementací bylo nutné zvolit vhodnou platformu implementace. Zvolil jsem programovací jazyk Python ve verzi 3.8. Jedná se o vysokoúrovňový objektově orientovaný programovací jazyk. Poskytuje vysokou míru abstrakce nad jednotlivými konstrukty, které obsahuje společně s bohatou standardní knihovnou. Pomocí nich jazyk Python umožňuje rychlý vývoj. Díky těmto pozitivům se hodí na implementaci algoritmů definovaných v této práci. Hlavní nevýhodou jazyka Python je, že se jedná o interpretovaný jazyk. na jeho spuštění tedy musíme mít nainstalovaný interpret. Obecně jsou interpretované jazyky pomalejší, což platí i pro Python. Pro tuto práci to však nepředstavuje problém. Dalším významným pozitivem je množství knihoven o které lze Python rozšířit. To umožňuje ještě vyšší míru abstrakce, která nesnižuje rychlost či efektivitu programu za předpokladu, že jsou knihovny využity účelně. Jako příklad lze uvést knihovnu numpy, která zvyšuje efektivitu ukládání informací do polí a graphviz umožňující generaci grafických výsledků. Obě tyto knihovny jsou využity v tomto programu.

Na základě těchto informací by neměl být problém vytvořit regulární výraz a spustit program. Příklady spuštění v podkapitole **Příklady užití** [5.6].

5.3 Struktura programu

Na začátku jsou importy knihoven nezbytných pro fungování programu. Následují funkce a na závěr main, který je koncipován tak, aby bylo možné využít soubor i jako knihovnu. Soubor obsahuje tyto funkce:

- **getarg()** - jednoduchá funkce která kontroluje množství argumentů a vrací regulární výraz jako string. Nedoporučuji tuto funkci využívat při použití souboru jako knihovny.

- **infixtopostfix(regex)** - funkce která převadí infixovou notaci na postfixovou dle pravidel v tabulce [5.1]. Funkce obsahuje pole a zásobník pro převod. Pro více informací

operátor	priorita	asociativita
*	3	←
|	2	←
.	1	←

Tabulka 5.1: Tabulka priorit a asociativity pro regulární výrazy

  viz **Matematické notace** [2.3]. Funkce vrací regulární výraz v postfixové notaci ve formátu string.

- **createnodesandrules(regex)** - Tato funkce vytváří z RV v postfixové notaci, který dostane v argumentu, seznam stavů a *numpy.array* pravidel. Funkce postupně prochází řetězec s regulárním výrazem a ukládá do zásobníku přečtené symboly, či ukazatele na již vytvořené KA. Pro lepší pochopení lze demonstrovat na příkladu. Např.: "ab*."

  1. Funkce přečte symbol "a"vloží na vrchol zásobníku.
  2. Funkce přečte symbol "b"vloží na vrchol zásobníku.
  3. Funkce přečte symbol "*"přečte první symbol na vrcholu zásobníku, což je "b", provede operaci pop, vytvoří automat dle kapitoly [4], který popisuje "b*"a pushne ukazatel na zásobník.
  4. Funkce přečte symbol "."přečte první dva symboly na zásobníku, což jsou ukazatel na automat a symbol "b", provede dvakrát operaci pop, vytvoří automat dle kapitoly [4] a pushne ukazatel na zásobník.

  Funkce potom vrací tuple, které obsahuje pole všech stavů, kde na první pozici je počáteční stav a na poslední pozici je koncový stav. Tuple dále obsahuje *numpy.array*, které je výčtem všech pravidel, obsahující automat.

- **createautomata(tuple)** - Je funkcí očekávající tuple zmíněné v předchozí funkci a generuje grafickou podobu.

5.4 Výstup

Komplikovanou volbou bylo zvolit vhodný výstup. Výstupem programu je konečný automat, který může být reprezentován mnoha způsoby. Cílem aplikace je demonstrovat počet stavů a přechodů v konečném automatu, což umožňuje grafické rozhraní ve velmi čitelné a srozumitelné podobě. Výstupní grafická podoba musí vzniknout z popisu algoritmu. Zvolená metoda je DOT soubor, který díky knihovně graphviz následně umožňuje přehledné zobrazení. Výstup se ukládá do složky output. Ve složce output se nachází 2 soubory automata, který obsahuje zápis automatu a automata.pdf, který je grafickou podobou.

DOT soubor

DOT je jazyk popisu grafu. Grafy DOT jsou obvykle soubory s příponou .dot nebo tečkou. Upřednostňuje se rozšíření .dot, aby nedošlo k záměně s tečkou rozšíření používanou verzemi aplikace Microsoft Word před rokem 2007. Soubory DOT mohou zpracovávat různé programy. Některé, například dot, neato, twopi, circo, fdp a sfdp, mohou číst soubor DOT a vykreslit jej v grafické podobě. Jiné, například gvpr, gc, ccomps, sccmap a tred, čtou soubory DOT a provádějí výpočty na znázorněném grafu. A konečně další rozhraní, například lefty, dotty a grappa poskytují interaktivní rozhraní. Nástroj GVedit kombinuje textový editor s neinteraktivním prohlížečem obrázků. Většina programů je součástí balíčku Graphviz nebo jej používá interně. Pro konečné automaty popsané DOT jazykem je popis následující.

Informace o typu grafu ve zmíněném případě konečném automatu je označena třídou **digraph**. Digraph je jediná použitá třída při tvorbě DOT souboru. Všechny informace o konečném automatu jsou zahrnuty v množinových závorkách {}. Informace jsou reprezentovány následujícími zápisy. SN a FN, kde N značí přirozené číslo, zaznamenává jednotlivé stavy, které konečný automat obsahuje. Zápis dále obsahuje výčet pravidel ve formátu Q_2 -> Q_1, kde $Q_1 \in Q$ a $Q_2 \in Q$. Každý automat dále obsahuje metastav $fake[style = invisible]$, který značí počátek automatu a tedy každý stav SN nebo FN, který má pravidlo $fake- > SN$ nebo $fake- > FN$ je stavem počátečním. Pro rozpoznání ucelených celků pravidel je každá informace oddělena znakem $\backslash n$.

Jednotlivé stavy nebo pravidla dále mají specifikace, které rozšiřují množství informací. Tyto specifikace musí být obsaženy vždy v hranatých závorkách []. Program využívá pro bod specifikace [*root* = *true*] označující počáteční stav a [*shape* = *doublecircle*] označující koncové stavy. Pro pravidla jsou využita specifikace [*style* = *bold*], která mění grafickou podobu šipky pro vstupní stav a [*label* = &], která označuje přechod se symbolem &.

5.5 Spuštění

Makefile

Informace o Makefilu lze získat pomocí **make** nebo **make help**. Makefile obsahuje vše nezbytné pro instalaci a spuštění programu. V Makefilu lze využít následující příkazy:

- **make** a **make help** - zobrazení nápovědy
- **make install** - instalace Pythonu a knihoven pro Ubuntu
- **make example-cmd** - ukázka spuštění z příkazové řádky
- **make example-make** - ukázka spuštění pomocí Makefilu
- **make run** - spuštění pomocí Makefilu

Manuální

Pro funkční spuštění je nezbytné mít nainstalované knihovny numpy a graphviz. Obě knihovny jsou součástí nástroje pip, a proto je lze nainstalovat příkazy `pip install numpy` a `pip install graphviz`. Knihovna graphviz je kompatibilní s verzí Pythonu 2.7 a 3.6+. Mnou používaná a tedy i doporučená verze je 3.8. Program je ovládán pomocí příkazového řádku, kde očekává právě jeden argument ve formátu string například: "a.b", "a|b.c*"nebo

"(b.a)". Jak lze odvodit s příkazů je vyhrazeno několik metaznaků, které slouží jako operace a nemohou být zahrnuty v abecedě RV a tedy i KA. Jsou to tyto znaky:

- | - slouží jako operátor pro sjednocení
- * - slouží jako operátor pro iteraci
- . - slouží jako operátor pro konkatenaci
- (- slouží jako metaznak pro začátek přednostně vyhodnocené části regulárního výrazu
-) - slouží jako metaznak pro konec přednostně vyhodnocené části regulárního výrazu

5.6 Příklady užití

Za předpokladu, že je nainstalován interpret Pythonu a knihovny numpy a graphviz, lze program spustit přímo z příkazové řádky například:

- **python3 gen_ka.py "a.b|c"** generuje automata viz výpis [5.1] automata.pdf viz obrázek [5.1]
- **python3 gen_ka.py "(a|b)*"** generuje automata viz výpis [5.2] automata.pdf viz obrázek [5.2]
- **python3 gen_ka.py "((a|b)*|(c.a)"** generuje automata viz výpis [5.3] automata.pdf viz obrázek [5.3]
- **make run rv="((0*|1*)|(2*|3*)).7'"** generuje automata viz výpis [5.4] automata.pdf viz obrázek [5.4]

```
digraph {
          fake [style=invisible]
          S4 [root=true]
          fake -> S4 [style=bold]
          F1 [shape=doublecircle]
          S1
          S2
          S3
          S1 -> F1 [label=b]
          S2 -> F1 [label=c]
          S3 -> S1 [label="ϵ"]
          S3 -> S2 [label="ϵ"]
          S4 -> S3 [label=a]
}
```

Výpis 5.1: automata k "a.b|c"

```
digraph {
	fake [style=invisible]
	S3 [root=true]
	fake -> S3 [style=bold]
	F2 [shape=doublecircle]
	S1
	S2
	F1
	S1 -> F1 [label=a]
	S2 -> F1 [label=b]
	S3 -> S1 [label="ϵ"]
	S3 -> S2 [label="ϵ"]
	S3 -> F2 [label="ϵ"]
	F1 -> S3 [label="ϵ"]
	F1 -> F2 [label="ϵ"]
}
```

Výpis 5.2: automata k "(a|b)*"

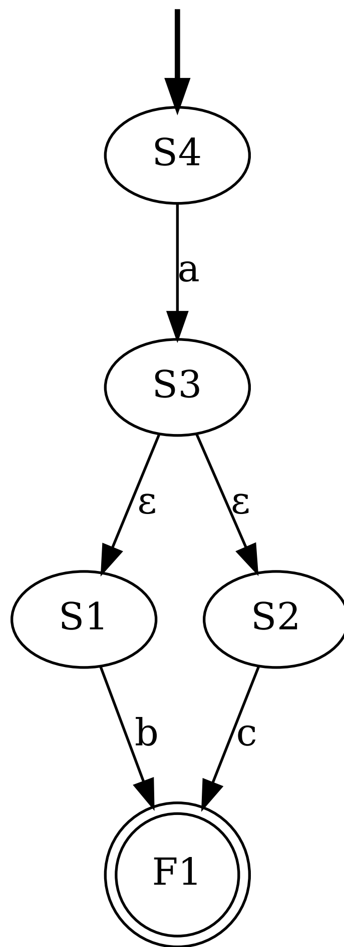
```
digraph {
	fake [style=invisible]
	S6 [root=true]
	fake -> S6 [style=bold]
	F2 [shape=doublecircle]
	S1
	S2
	S3
	F1
	S4
	S5
	S1 -> F1 [label=a]
	S2 -> F1 [label=b]
	S3 -> S1 [label="ϵ"]
	S3 -> S2 [label="ϵ"]
	S3 -> F2 [label="ϵ"]
	F1 -> S3 [label="ϵ"]
	F1 -> F2 [label="ϵ"]
	S4 -> S5 [label=c]
	S5 -> F2 [label=a]
	S6 -> S3 [label="ϵ"]
	S6 -> S4 [label="ϵ"]
}
```

Výpis 5.3: automata k "((a|b)*|(c.a)"

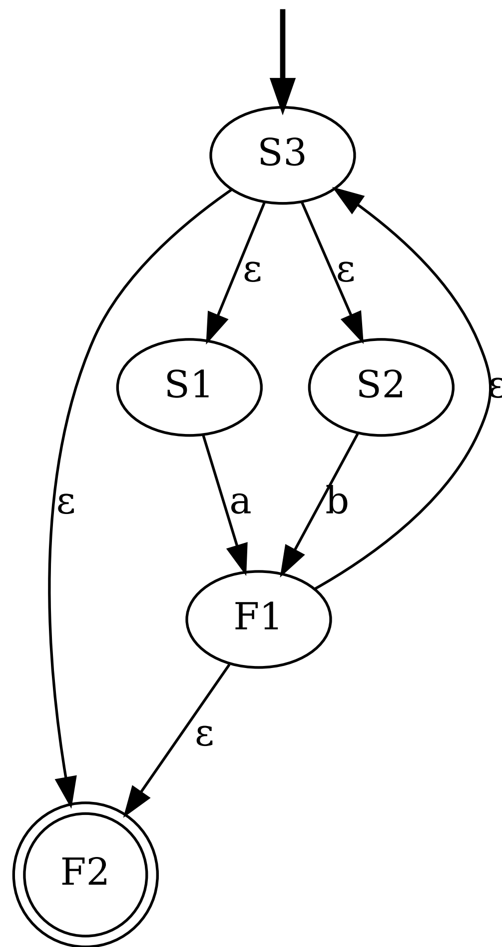
```
digraph {
	fake [style=invisible]
	S7 [root=true]
	fake -> S7 [style=bold]
	F9 [shape=doublecircle]
```

```
        S1
        F1
        F2
        S2
        F3
        S3
        S4
        F5
        S5
        F7
        S6
        S1 -> F1 [label=0]
        S1 -> F2 [label="ϵ"]
        F1 -> S1 [label="ϵ"]
        F1 -> F2 [label="ϵ"]
        S2 -> F3 [label=1]
        S2 -> F2 [label="ϵ"]
        F3 -> S2 [label="ϵ"]
        F3 -> F2 [label="ϵ"]
        S3 -> S1 [label="ϵ"]
        S3 -> S2 [label="ϵ"]
        S4 -> F5 [label=2]
        S4 -> F2 [label="ϵ"]
        F5 -> S4 [label="ϵ"]
        F5 -> F2 [label="ϵ"]
        S5 -> F7 [label=4]
        S5 -> F2 [label="ϵ"]
        F7 -> S5 [label="ϵ"]
        F7 -> F2 [label="ϵ"]
        S6 -> S4 [label="ϵ"]
        S6 -> S5 [label="ϵ"]
        S7 -> S3 [label="ϵ"]
        S7 -> S6 [label="ϵ"]
        F2 -> F9 [label=7]
}
```

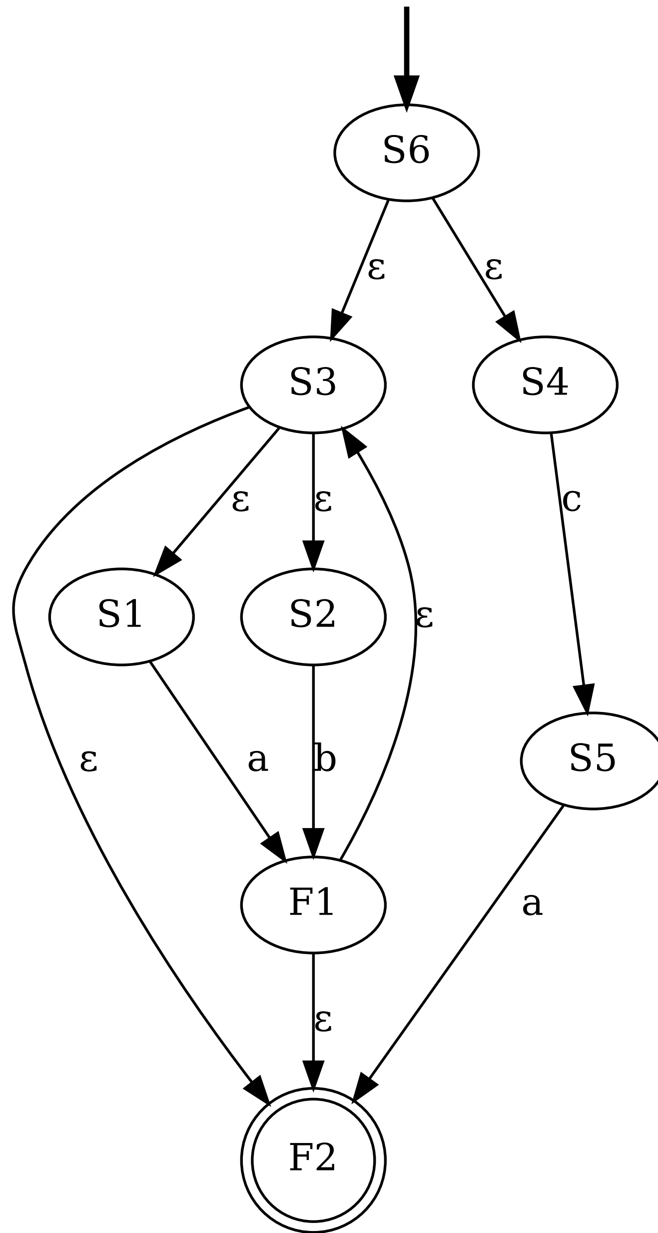
Výpis 5.4: automata k "((0*|1*)|(2*|3*)).7"



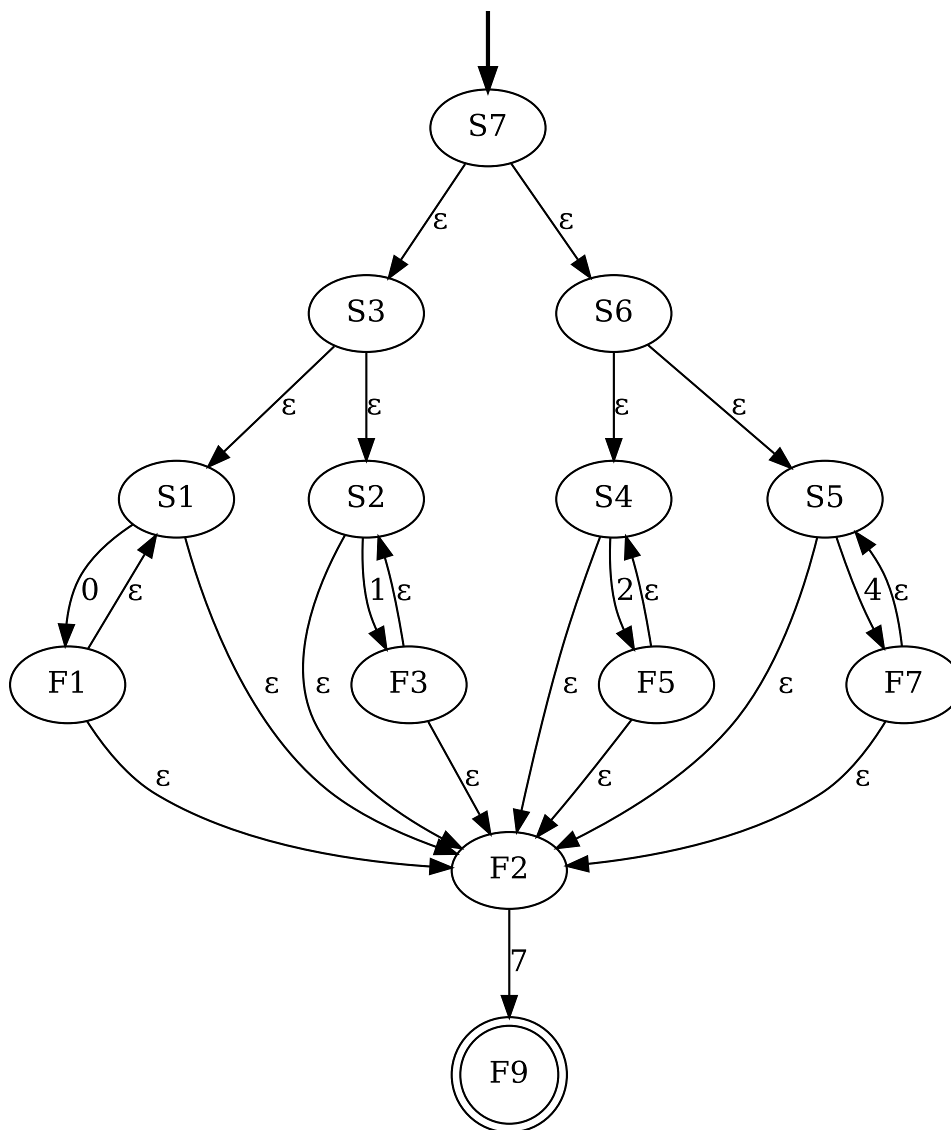
Obrázek 5.1: KA reprezentující příkaz: python3 gen_ka.py "a.b|c"



Obrázek 5.2: KA reprezentující příkaz: python3 gen_ka.py "(a|b)*"



Obrázek 5.3: KA reprezentující příkaz: python3 gen_ka.py "(a|b)*|(c.a)"



Obrázek 5.4: KA reprezentující příkaz: python3 gen_ka.py "((0*|1*)|(2*|3*)).7"

Kapitola 6

Závěr

Cílem této práce bylo seznámit se s matematickými modely formálních jazyků se zaměřením na konečné automaty a regulární výrazy a jejich vlastnosti. Dále nastudovat transformace z regulárního výrazu na konečný automat a algoritmy s tím spojené. Dále bylo nezbytné nastudovat všechny veličiny, ze kterých se automaty skládají od symbolu přes abecedu až k jazyku. Důležitým krokem bylo pochopení a důkladné studium zavedených algoritmů pro tvorbu konečného automatu z regulárního výrazu, která je jádrem této práce. Poslední částí studia bylo studium matematických důkazů spojených s ekvivalencí matematických modelů popisujících formální jazyky, které nebylo součástí zadání, ale ukázalo se jako nezbytné pro relevantní tvorbu na poli transformací jazykových modelů. Následně navrhnout alternativní transformaci regulárního výrazu na konečný automat se zaměřením na ekonomickou stránku v oblasti stavů a přechodů konečného automatu. Všechny cíle práce byly splněné.

Na základě znalosti algoritmů pro zavedené transformace jazykových modelů bylo možné navrhnout alternativní transformace jazykových modelů, které měly za cíl snížit počet stavů a přechodů, které byly pro jednoduchost algoritmu na úkor složitosti algoritmu pro tvorbu konečného automatu, který tímto procesem vznikl. Pro smysluplnost práce bylo poté důležité dokázat, že tyto nově zavedené algoritmy nejsou holým výmyslem, ale validně zavedeným procesem, což vedlo k nezbytnosti důkazu validnosti jednotlivých částí transformace a tím i transformace jako takové. K tomu byla zvolena varianta matematického důkazu pomocí matematické indukce, která validnost ověřuje a tím i dokazuje. Podle vytvořených algoritmů byl naimplementován program, který tuto transformaci provede. Konečný automat lze zapsat několika způsoby. Pro potřeby tohoto programu se ukázala jako nejvhodnější grafická podoba, která umožňuje jednoduše a přehledně ukázat vytvořené konečné automaty. Regulární výraz je nejprve převeden a poté zapsán v postfixovovém tvaru a následně převeden dle navržených algoritmů na výčet stavů, kde jsou označeny počáteční a koncové stavy a výčet pravidel. Ty jsou generovány do grafické podoby, která je potom poskytnuta uživateli. Program je tedy schopen nejen transformace z regulárního výrazu na konečný automat, ale také kontroly validnosti regulárního výrazu.

Algoritmus jako takový nebylo problém navrhnout, ovšem při jeho návrhu bylo nezbytné vyladit detaily, které se zdály nepatrné, ovšem tyto detaily byly rozdílovým faktorem mezi algoritmem bez potenciálu a smyslu a algoritmem, který fungoval a vytvářel konečné algoritmy, které dávaly smysl. Toto vyhlazování algoritmu probíhalo až do tvorby matematického důkazu díky kterému byly všechny nedostatky vyřešeny a algoritmy vyhlazeny. Zajímavé bylo navrhovat algoritmy se zaměřením na výstup algoritmu než na algoritmus samotný, což práci přineslo zajímavou dynamiku. Při implementaci bylo významné uvědomit

si důležitost konverze infixové notace v regulárním výrazu na postfixovou, což významně usnadnilo další postup implementací. Práce byla v teoretické části náročná při návrhu algoritmů a obzvláště potom při dokazování validnosti algoritmů pomocí matematických důkazů, což vyžadovalo matematické znalosti na vysoké úrovni a schopnost aplikovat teorii matematických důkazů pro praktické příklady. Díky vynaloženému úsilí při návrhu a kvalitnímu popisu algoritmů samotná implementace nebyla komplikovaná, protože vše nezbytné pro plně funkční implementaci bylo do detailu rozebráno a popsáno v teoretické části práce. Pro implementaci se volba jazyka a knihoven ukázala jako perfektní díky jednoduchému a přímému užití, které přesně plnilo potřeby pro implementaci. Pomocí matematického důkazu a programu vytvořeného v rámci zdrojového kódu, který je součástí této práce se podařilo splnit cíle zadání.

Tato práce reprezentuje jiný druh zaměření při transformaci regulárního výrazu na konečný automat, kde cílem není algoritmus, ale konečný automat. Navržené algoritmy lze využít pro potřeby ve všech odvětvích informačních technologií, kde je zapotřebí převádět mezi jazykovými modely, díky širokému spektru teoretické informatiky, protože konečné automaty jsou základním stavebním kamenem tohoto odvětví a tedy i informatiky jako takové. Téma konečných algoritmů má široký dosah a pokud přímo není využito s algoritmy vytvořenými v této práci, tak připomíná, že některé věci nejsou pevně dané a lze se zamyslet, jestli přetvořením základních algoritmů nelze dosáhnout lepších výsledků. Tato práce otevírá nové možnosti, protože matematickými modely nejsou pouze regulární výrazy a konečné automaty. Také jedinou ve světě informatiky využívanou transformací není transformace z regulárního výrazu na konečný automat, tak lze na tuto práci jednoduše navázat při jiných matematických modelech s jinými transformacemi. Dále je možno navázat návrhem algoritmů, jejich matematickým důkazem a implementací dalších operací, kterých je značná řada v regulárních výrazech a všechny lze využít při převodu na konečný automat. Další variantou návaznosti jsou transformace pro paralelní konečné automaty, která je k datu vypracování práce významným tématem, ve kterém se odehrávají významné objevy a posuny vpřed na poli konečných automatů.

Literatura

[1] GeeksforGeeks. *Mealy and Moore Machines in TOC* [online]. GeeksforGeeks, listopad 2019 [cit. 2021-04-20]. Dostupné z: https://www.geeksforgeeks.org/mealy-and-moore-machines-in-toc/.

[2] Ivana Černá, A. K. *Automaty a formální jazyky I* [online]. 2002 [cit. 2021-04-01]. Dostupné z: https://is.muni.cz/elportal/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf.

[3] Meduna, A. *Formal Languages and Computation: Models and Their Applications*. 1. vyd. CRC Press, 2014. ISBN 9781466513457,1466513454.

[4] Meduna, A. *Formální jazyky a překladače* [online]. 2020 [cit. 2020-12-26]. Dostupné z: https://www.fit.vutbr.cz/study/courses/IFJ/public/materials/.

[5] Pukančík, M. *Analýza a transformace kódu založená na převodnících*. Brno, CZ, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22798/.

[6] Vais, V. *Teoretická informatika 1. část* [online]. 2020 [cit. 2021-01-07]. Dostupné z: http://home.zcu.cz/~vais/Vais%20-%20KA%20a%20RG.pdf.

[7] Vavrečková Šárka. *Teorie jazyků a automatů I, Základy teoretické informatiky I* [online]. 2016 [cit. 2021-04-01]. Dostupné z: http://vavreckova.zam.slu.cz/obsahy/tja/skripta1/tja1.pdf.