

# Vizualizace regulárních výrazů

Regular Expression Visualization

Dominik Kundra

Bakalářská práce

Vedoucí práce: Ing. Jakub Beránek

Ostrava, 2024

# Zadání bakalářské práce

Student:

**Dominik Kundra**

Studijní program:

B0613A140014 Informatika

Téma:

Vizualizace regulárních výrazů  
Regular Expression Visualization

Jazyk vypracování:

čeština

Zásady pro vypracování:

Cílem práce je vytvořit nástroj sloužící pro vizualizaci a ladění regulárních výrazů. Nástroj by měl být schopný zpracovat zvolený regulární výraz, sestavit plán vykonávání daného výrazu dle zvolené implementace a poté umožnit programátorovi interaktivně krokově provádění regulárního výrazu. Nástroj by měl být vytvořen jako rozšíření do vývojového prostředí (např. do Visual Studio Code), aby šel jednoduše použít při vývoji programů. Výsledná aplikace by měla být řádně zdokumentována a při jejím vývoji by měl být využit verzovací systém (např. git).

1. Analyzujte a popište možnosti implementace regulárních výrazů.
2. Navrhnete architekturu rozšíření do vývojového prostředí, které bude schopné analyzovat regulární výrazy ze zvoleného zdrojového kódu.
3. Naimplementujete nástroj pro vizualizaci regulárního výrazu a integrujte jej do vývojového prostředí.
4. Otestujete vizualizaci nástroje na regulárních výrazech z reálných projektů.

Seznam doporučené odborné literatury:

[1] FRIELD, Jeffrey. Mastering Regular Expressions 3rd Edition. 2006. O'Reilly Media. ISBN: 978-0596528126

[2] SORVA, Juha. Visual program simulation in introductory programming education. Espoo: Aalto Univ. School of Science, 2012. ISBN 9789526046266. Dostupný také z WWW: <http://doi.acm.org/10.1145/2445196.2445368>.

[3] VANDERKAM, Dan. Effective TypeScript : 62 Specific Ways to Improve Your TypeScript. O'Reilly Media, 2019. ISBN 978-1492053699

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jakub Beránek**

Datum zadání: 01.09.2023

Datum odevzdání: 30.04.2024

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 09.11.2023 15:43:31

## **Abstrakt**

No Czech or Slovak abstract is given

## **Klíčová slova**

No Czech or Slovak keywords are given

## **Abstract**

No English abstract is given

## **Keywords**

No English keywords are given

# Obsah

<b>Seznam obrázků</b>	<b>5</b>
<b>Seznam tabulek</b>	<b>6</b>
<b>1 Úvod</b>	<b>7</b>
<b>2 Principy a historie regulárních výrazů</b>	<b>8</b>
2.1 Formální jazyk . . . . .	8
2.2 Konečný automat . . . . .	8
2.3 Původ, vznik a implementace . . . . .	10
<b>Literatura</b>	<b>12</b>
<b>Přílohy</b>	<b>12</b>

# Seznam obrázků

2.1	Příklad deterministického automatu přijímající slova obsahující písmena z abecedy {a, b} končící písmenem a . . . . .	9
2.2	Příklad nedeterministického automatu ekvivalentního k předchozímu deterministickému	9
2.3	Převedený prázdný výraz $\epsilon$ . . . . .	10
2.4	Převedený výraz <b>a</b> . . . . .	10
2.5	Převedený výraz <b>s t</b> . . . . .	11

# Seznam tabulek

# Kapitola 1

## Úvod

Vyhledávání v textu patří mezi základní problémy, se kterými se velice pravděpodobně potká skoro každý programátor. Tento problém se dá řešit mnoha způsoby, avšak ne všechna řešení lze použít univerzálně a každý ze způsobů má své výhody a nevýhody. Jedním ze přístupů je využití regulárních výrazů. Jedná se o sadu znaků, které nám umožňují nadefinovat výraz a ten je následně převedený na strukturu, nejčasteji ve formě konečných automatů. Téměř každý dnešní programovací jazyk je obsahuje, ale jejich implementace se mohou lišit.

Cílem této práce je naimplementovat nástroj, který bude schopný procházet regulární výrazy a následně vizualizovat tyto průchody, jako rozšíření do vývojového prostředí.

Při vývoji programů, je programátor často obeznámen s regulárními výrazy, jedná se totiž o poměrně rychlé řešení pro vyhledávání v textu. Můžeme se s nimi setkat v podstatě skoro ve všech částech softwaru<sup>1</sup>, např. validace formulářů, vyhledávání v textu nebo třeba v příkazovém řádku. Tyto výrazy se však brzy mohou stát hůře čitelnými, jelikož neumožňují v podstatě žádné formátování<sup>2</sup>. Taktéž mohou být pro mnoho lidí matoucí, či nepřehledné. Z tohoto důvodu se hodí mít nástroj, který potencionálně usnadní práci programátorům, tak aby si mohli zobrazit průchod zadaným výrazem. Dále pro lidi, kteří například vidí tyto výrazy poprvé v životě může být snazší jim porozumět, je-li jim ukázáno jak fungují v jednotlivých krocích. Sice již existují řešení tohoto problému a to v různých formách [1, 2, 3], ale pro zvolené vývojové prostředí mnoho přístupů neexistuje. Tato situace je motivací, zabývat se problémem do hloubky a nabídnout originální řešení v daném směru, které by mohlo být přínosem pro ostatní lidi.

---

<sup>1</sup>počítačový program, aplikace

<sup>2</sup>upravení vzhledu, tvaru

## Kapitola 2

# Principy a historie regulárních výrazů

Abychom mohli porozumět, jakým problémem se v této práci vlastně zabýváme, tak si musíme zadefinovat co jsou regulární výrazy, jak fungují a jak se jednotlivé implementace mohu lišit. Následně si vysvětlíme, jak lze zjednodušit pochopení těchto výrazů, popřípadě jak rychle najít chybu ve vlastním výrazu.

Předtím než si vysvětlíme blíže jak fungují samotné regulární výrazy, je potřeba si první objasnit význam několika pojmů z teoretické informatiky<sup>1</sup>.

### 2.1 Formální jazyk

Formální jazyk je libovolná množina konečných slov nad určitou abecedou. Slova chápeme jako řetěze znaků, která jsou přijímaná zadaným jazykem. Tato slova musí být sice konečná ale množina těchto slov může být nekonečná. Tyto jazyky mohou být definovány regulárními výrazy, formální gramatikou, konečnými automaty a dalšími. Regulární jazyky jsou tak jednou z možných definic formálních jazyků.

### 2.2 Konečný automat

Ve spojení s regulárními výrazy se často pojí konečné automaty, jedná se o další oblast v teoretické informatice. Proto abychom pochopili proč je tato oblast pro nás důležitá, musíme si vysvětlit co vlastně jsou.

Konečný automat je model jednoduchého počítače, který má určitý počet stavů a přechodů [4].

Stavy jsou typicky zakreslovány jako kružnice, a každý automat musí obsahovat alespoň jeden počáteční stav, ale může jich také obsahovat více. To samé platí pro konečný/é stav/y. Konečné stavy se vyznačují jako kruh z dvojitou čarou a počáteční stavy jsou, označovány jako stav do kterého vede šipka, která ale nevychází z jiného stavu. Prechody jsou pak šipky vedoucí z jednoho stavu do

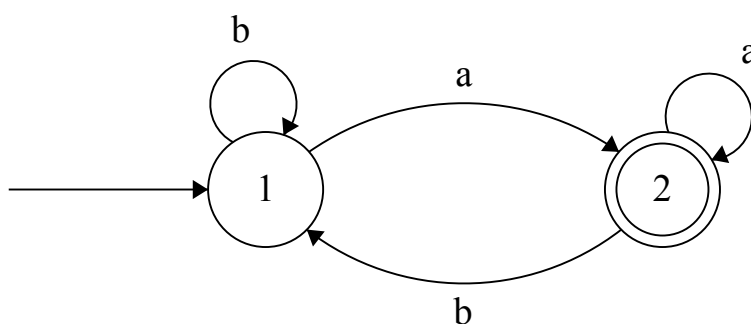
---

<sup>1</sup>vědní obor na pomezí mezi informatikou a matematikou

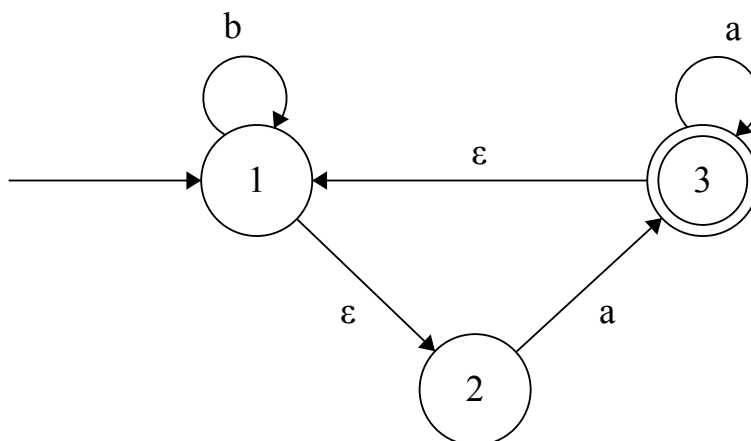


druhého, jsou označeny přechodovým symbolem. Tyto šipky nám říkají že pokud chceme přejít z jednoho stavu do druhého, tak musíme v přijímaném slově se posunout o daný symbol. Pokud to není možné, tak nemůžeme přejít do tohoto stavu.

Tyto automaty dělíme na deterministické a nedeterministické, zkráceně DKA (deterministický konečný automat) a NKA (nedeterministický konečný automat). DKA mohou mít v daném stavu pro každý znak abecedy **maximálně** jeden přechod, dále **nemohou obsahovat tzv. prázdný znak** často označovaný řeckým písmenem epsilon  $\epsilon$ . NKA naopak obojí umožňují, prázdné znaky nám umožňují změnu stavu bez změny aktuální pozice v hledaném slově. Každý NKA lze převést na ekvivalentní DKA.



Obrázek 2.1: Příklad deterministického automatu přijímající slova obsahující písmena z abecedy {a, b} končící písmenem a



Obrázek 2.2: Příklad nedeterministického automatu ekvivalentního k předchozímu deterministickému

## 2.3 Původ, vznik a implementace

Regulární výrazy byly poprvé nadefinovány Americkým matematikem **Stephan Cole Kleene**, jako regulární jazyky. Dále se aplikovali v teoretické informatice, jako podkategorie **teorie automatů** a součást **formálních jazyků**. Ačkoliv byli nadefinovány začátkem padesátých let, tak jejich využití v počítačích nastalo až na konci šedesátých let a to v jedním z nejznámějších operačních systémů UNIX.

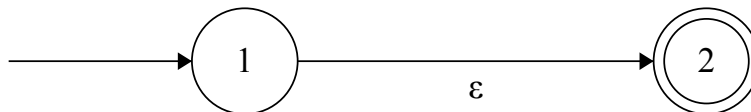
### Thompsonovo sestrojení

**Ken Thompson** byl člověkem kdo navrhnul první implementaci převodu regulárního výrazu na NKA využívanou v počítačích, která se používá do dnes. Algoritmus se pojmenoval **Thompson's construction** (Thompsonovo sestrojení), který převádí textovou reprezentaci výrazu na ekvivalentní nedeterministický automat. Toto sestrojení je využito v této práci, proto si musíme vysvětlit jak funguje.

Používá se často implementace NKA, jelikož je poměrně jednoduchá na implementaci a také dovoluje oproti DKA využití zpětného krokování (backtracking) a rozhlédnutí se kolem sebe (look-around). DKA mají výhodu že jsou rychlejší oproti NKA, ale jsou typicky větší než jejich ekvivalentní NKA a neumožňují již zmíněné funkce. Někdy se ale využívá například kombinace DKA i NKA, kdy DKA kvůli vyšší rychlosti se použije na vyhledání daného slova a pokud bylo nalezeno, tak se využije NKA pro jejich rozšířené možnosti.

Výsledný NKA má právě jeden vstupní a výstupní stav. Aby došlo ke správnému sestrojení NKA z Regulárního výrazu, musíme se řídit několika pravidly, pro ukázkou si pár těchto pravidel ukážeme.

Prázdný výraz  $\epsilon$ , je převedený na vstupní stav přechod  $\epsilon$  a konečný stav.



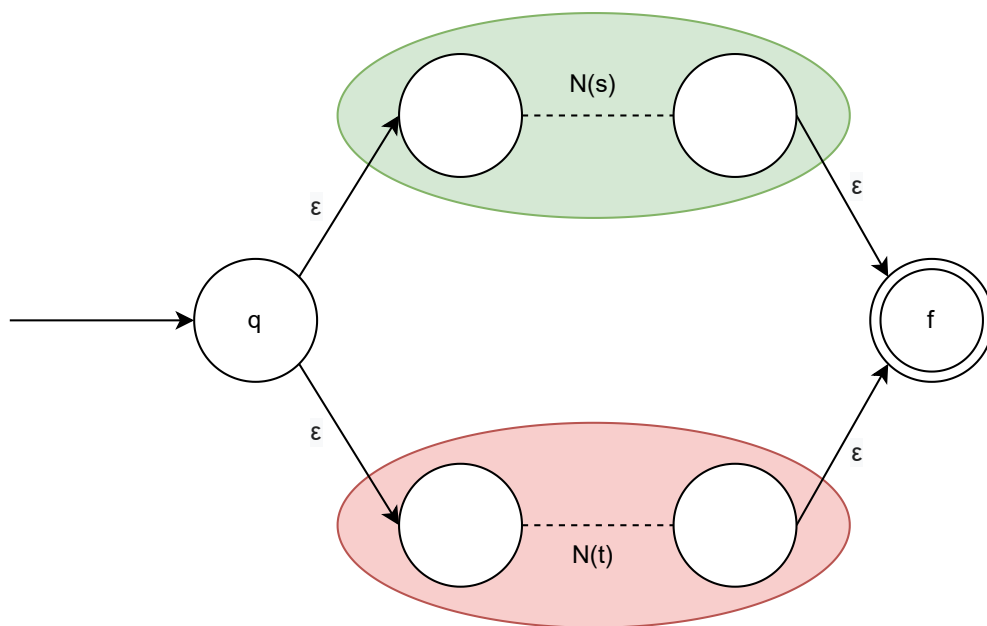
Obrázek 2.3: Převedený prázdný výraz  $\epsilon$

Výraz  $a$ , je převedený podobně jako prázdný výraz, ale s rozdílem přechodu  $a$  místo  $\epsilon$ .



Obrázek 2.4: Převedený výraz **a**

Pro zadaný výraz  $s|t$ , kdy  $s$  je levá strana varianty a  $t$  je pravá strana varianty, platí že ze stavu  $q$  vedou dva přechody  $\epsilon$  na počáteční stavy variant  $s$  a  $t$ . Z těchto počátečních stavů dále pokračuje sekvence stavů  $N(s)$  pro  $s$  a  $N(t)$  pro  $t$ . Konce variant  $s$  a  $t$  mají jediný přechod  $\epsilon$  na konečný stav.



Obrázek 2.5: Převedený výraz  $s|t$

Další pravidla pro sestrojení lze například najít na anglické wikipedia stránce pro thompsonovo sestrojení [5].

# Literatura

1. DIB, Firas. *Build, test, and debug regex* [online]. [B.r.]. [cit. 2024-01-25]. Dostupné z: <https://regex101.com/>.
2. AVALLONE, Jeff [online]. [B.r.]. [cit. 2024-01-25]. Dostupné z: <https://regexper.com/>.
3. [online]. [B.r.]. [cit. 2024-01-25]. Dostupné z: <https://regexr.com/>.
4. HAVRLANT, Lukáš. *Konečný Automat* [online]. [B.r.]. [cit. 2024-02-06]. Dostupné z: <https://www.matweb.cz/konecny-automat/>.
5. [online]. Wikimedia Foundation, 2023-07 [cit. 2024-02-17]. Dostupné z: [https://en.wikipedia.org/wiki/Thompson%27s\\_construction](https://en.wikipedia.org/wiki/Thompson%27s_construction).