

Vizualizace pravě jednosměrných skákajících konečných automatů

Jiří Drápal

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Jiří Drápal
Osobní číslo: A22003
Studijní program: B0613A140020 Softwarové inženýrství
Forma studia: Prezenční
Téma práce: Vizualizace pravě jednosměrných skákajících konečných automatů
Téma práce anglicky: The Visualisation of Right One-Way Jumping Finite Automata

Zásady pro vypracování

- Popište teorii týkající se pravě jednosměrných skákajících konečných automatů (dále PJSKA).
- Do popisu zahrňte PJSKA deterministické i nedeterministické, s epsilonovými přechody i bez epsilonových přechodů.
- Ve vhodném programovacím jazyce sestavte simulátor různých variant PJSKA.
- Simulátor musí umožňovat jednoduché zadávání PJSKA ve zvolené variantě, jejich editaci a názornou vizualizaci jejich činnosti.

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

Seznam doporučené literatury:

- BEIER Simon a HOLZER Markus. Properties of right one-way jumping finite automata. *Theoretical Computer Science*. 2019, sv. 798, s. 78–94. ISSN 0304-3975.
- BEIER Simon a HOLZER Markus. Nondeterministic right one-way jumping finite automata. *Information and Computation*. 2022, sv. 284, č. čl. 104687. ISSN 0890-5401.
- Černá, Ivana a kol. *Automaty a formální jazyky I*. Učební text FI MU. Fakulta informatiky, Masarykova univerzita, Brno: 2002. Dostupné z: http://is.muni.cz/elportal/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf. [cit. 2024-10-27]
- Chigahara Hiroyuki, Fazekas Szilárd Zsolt, a Yamamura Akihiro. One-way jumping finite automata. *International Journal of Foundations of Computer Science*. 2016, roč. 27, s. 391–405. ISSN 0129-0541.
- Meduna Alexander a Krivka Zbyněk. *Jumping Computation: Updating Automata and Grammars for Discontinuous Information Processing*. Boca Raton: CRC Press, 2024. ISBN 978-0-367-62093-6.
- Meduna Alexander a Zemek Petr. Jumping finite automata. *International Journal of Foundations of Computer Science*. 2012, roč. 23, s. 1555–1578. ISSN 0129-0541.
- PYTHON SOFTWARE FOUNDATION. *Python 3.13.0 documentation*. Online. 2024. Dostupné z: <https://docs.python.org>. [cit. 2024-10-25].
- QT GROUP. *Doc.qt.io*. Online. 1995, 14.08.2024. Dostupné z: <https://doc.qt.io>. [cit. 2024-10-25].
- Sipser Michael. *Introduction to the Theory of Computation*. 2. vyd. Boston: Thomson Course Technology, 2006. ISBN 0-534-95097-3.

Vedoucí bakalářské práce: **Ing. Pavel Martinek, Ph.D.**
Ústav matematiky

Datum zadání bakalářské práce: **27. října 2024**
Termín odevzdání bakalářské práce: **2. června 2025**

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



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

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- 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;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mě požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- 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 řádně citoval; v případě publikace výsledků budu uveden jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Pravě jednosměrné skákající konečné automaty jsou rozšířením klasických konečných automatů o možnost skokového pohybu čtecí hlavy při čtení vstupního slova. V teoretické části této práce jsou definovány deterministické i nedeterministické varianty těchto automatů. Je popsáno procházení slov a vztahy k formálním jazykům podle Chomského hierarchie. Praktická část práce je zaměřená na návrh a realizaci desktopové aplikace pro vizualizaci procházení vstupního slova pravě jednosměrnými konečnými automaty i automaty klasickými. Aplikace je naprogramována v jazyce Python s využitím frameworku PyQt6 a umožňuje uživateli prostřednictvím grafického rozhraní zadat konfiguraci automatu, která je následně vizualizována.

Klíčová slova: Konečný automat, Pravě jednosměrný skákající konečný automat, Vizualizace, Python

ABSTRACT

Right one-way jumping finite automata is an extension of standard finite automata with the possibility of jumping of the reading head over the input word. In the theoretical part of the thesis formulation of the automata and traversal of words, and their relation to the formal languages of the Chomsky hierarchy is described. The practical part deals with the design and creation of a desktop application for the visualization of traversal of the input word by both Right one-way jumping finite automata and classical finite automata. The application is created in the programming language Python with the framework PyQt6. The user can use the graphical interface to set desired configuration of the automata which will be visualized.

Keywords: Finite automata, Right one-way finite automata, Visualization, Python

Na tomto místě bych rád poděkoval vedoucímu své bakalářské práce, Ing. Pavlu Martinkovi, Ph.D., za odborné vedení, cenné rady, vstřícnost a podporu během celého zpracování této práce.

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST	12
1 ZÁKLADNÍ POJMY	13
1.1 DETERMINISTICKÝ KONEČNÝ AUTOMAT.....	14
1.2 NEDETERMINISTICKÝ KONEČNÝ AUTOMAT	15
2 PRAVĚ JEDNOSMĚRNÝ SKÁKAJÍCÍ KONEČNÝ AUTOMAT.....	17
2.1 DETERMINISTICKÝ PRAVĚ JEDNOSMĚRNÝ SKÁKAJÍCÍ AUTOMAT	17
2.2 NEDETERMINISTICKÝ PRAVĚ JEDNOSMĚRNÝ SKÁKAJÍCÍ KONEČNÝ AUTOMAT	18
2.3 SOUVISLOST S JAZYKY CHOMSKÉHO HIERARCHIE.....	20
II PRAKTICKÁ ČÁST.....	21
3 TVORBA APLIKACE.....	22
3.1 GUI.....	22
3.1.1 Hlavní panel	23
3.1.2 Nastavení konfigurace automatu.....	24
3.1.3 Vizualizace	25
4 POPIS APLIKACE	26
4.1 KNIHOVNA FINITE AUTOMATA PY	26
4.1.1 Činnost deterministického automatu	28
4.1.2 Činnost nedeterministického automatu	29
4.2 FUNKCE PRO PRÁCI S KONEČNÝM AUTOMATEM	33
4.2.1 GetItem.....	33
4.2.2 getNextSymbol.....	33
4.2.3 setUsed	34
4.3 APLIKACE.....	34
4.3.1 Vizualizace vstupní abecedy	36
5 NÁVOD K APLIKACI	38
5.1 INSTALACE A SPUŠTĚNÍ	38
5.2 NASTAVENÍ APLIKACE.....	39
5.3 ZADÁVÁNÍ AUTOMATU	39
5.4 VIZUALIZACE AUTOMATU	41
5.5 UKLÁDÁNÍ A NAČÍTÁNÍ SOUBORŮ	43
ZÁVĚR	45
SEZNAM POUŽITÉ LITERATURY.....	47
SEZNAM OBRÁZKŮ	48
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	49

SEZNAM PŘÍLOH.....50

ÚVOD

Tato bakalářská práce se zabývá právě jednosměrnými konečnými automaty (PJSKA) včetně jejich deterministických a nedeterministických variant (NPJSKA). Jedná se o rozšíření klasických konečných automatů, které ovlivňuje, jak způsob zpracování vstupních slov, tak i jazyky, které tyto automaty přijímají.

Teoretická část se zaměřuje na rozbor právě jednosměrných konečných skákajících automatů z pohledu teorie formálních jazyků a automatů. Úvodem budou představeny základní koncepty nezbytné pro pochopení problematiky, včetně formulace konečných automatů klasických a právě jednosměrně skákajících. Dále bude podrobně popsána formální definice PJSKA, jejich struktura, způsob zpracování vstupních řetězců a specifiky kroků výpočtu. Také budou popsány vztahy mezi právě jednosměrnými konečnými automaty a třídami jazyků Chomského hierarchie. Kromě teoretického popisu budou uvedeny konkrétní příklady těchto automatů, na kterých bude vysvětleno jejich chování při různých konfiguracích.

Praktická část se soustředí na tvorbu aplikace pro vizualizaci těchto automatů. Cílem je umožnit uživatelům vizualizovat činnost PJSKA a NPJSKA, což pomůže k lepšímu pochopení jejich chování a vlastností. Vývoj zahrnuje návrh uživatelského rozhraní a programování činnosti automatů. Součástí praktické části je také vytvoření návodu, kde budou vysvětleny jednotlivé možnosti aplikace a práce s ní.

I TEORETICKÁ ČÁST

1 ZÁKLADNÍ POJMY

Vstupní abeceda je neprázdná konečná množina symbolů, ze kterých se může skládat vstupní slovo. Značíme ji Σ . Například vstupní abeceda, která se skládá ze symbolů „a“, „b“ a „c“ může být formálně zapsána jako $\Sigma = \{a, b, c\}$. Abeceda tvoří základní prvek formální jazykové struktury a určuje možnosti konstrukce slov [1].

Vstupní slovo je definováno jako konečná posloupnost symbolů pocházejících z dané vstupní abecedy Σ . Formálně lze slovo w definovat jako:

$$w = a_1 a_2 a_3 \dots a_n, \text{ kde } a_i \in \Sigma \text{ pro } i = 1, 2, \dots, n. [2]$$

Délka vstupního slova w , označovaná jako $|w|$, je určena počtem symbolů, které slovo obsahuje. Formálně pak $|w| = n$. [3]

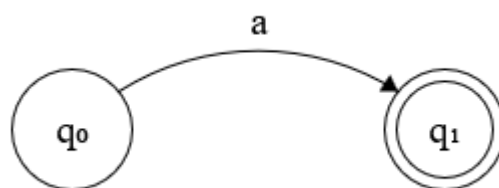
Například uvažujeme-li vstupní abecedu $\Sigma = \{0, 1\}$, pak slovo $w = 10101$ je platným slovem, neboť všechny jeho symboly jsou obsaženy v abecedě Σ . Jeho délka je $|w| = 5$, protože obsahuje pět symbolů.

Naopak slovo $w = 1201$ v rámci této vstupní abecedy není platné, jelikož obsahuje symbol „2“, který se v množině $\Sigma = \{0, 1\}$ nenachází. Délka tohoto slova je $|w| = 4$.

Množina stavů, značená Q , představuje konečnou množinu všech možných stavů, ve kterých se může konečný automat nacházet v průběhu zpracování vstupního slova. Stavy jsou nejčastěji značeny $q_0, q_1, \dots, q_n$. Množinu stavů, která se skládá ze čtyř stavů q_0, q_1, q_2, q_3 , můžeme označit jako $Q = \{q_0, q_1, q_2, q_3\}$. Konkrétní stav, společně s aktuálně zpracovávaným slovem a pozicí čtecí hlavy, určuje jedinečnou konfiguraci automatu v daném okamžiku výpočtu.[1]

Zpracování vstupního slova probíhá na základě přechodové funkce δ , která popisuje, jakým způsobem automat přechází z jedné konfigurace do druhé v závislosti na aktuálním stavu a přečteném symbolu. Přechodová funkce δ se formálně definuje jako:

$\delta: Q \times \Sigma \rightarrow Q$ [3], kde první množina Q odpovídá aktuálnímu stavu, Σ reprezentuje symbol právě čtený vstupním automatem a výsledné Q je stav, do nějž se automat může následně přesunout.



Obrázek 1 Zobrazení přechodu

Na obrázku 3 můžeme vidět zobrazení stavů a přechodů pomocí takzvaného stavového diagramu. Tento diagram zobrazuje přechod ze stavu q_0 při čtení symbolu „a“ do stavu q_1 . Stav q_1 je koncovým stavem, což je zobrazeno dvojitém kruhem.

1.1 Deterministický konečný automat

Konečný automat lze formulovat jako pětici:

$$M = (Q, \Sigma, \delta, q_0, F),$$

kde Q je množina stavů, Σ je vstupní abeceda, δ je přechodová funkce, $q_0 \in Q$ je počáteční stav. $F \subseteq Q$ je množina koncových stavů.[3]

Konečný automat začíná svůj výpočet v počátečním stavu q_0 se vstupním slovem umístěným na pásce. Automat následně zpracovává vstupní symboly jeden po druhém podle pravidel definovaných přechodovou funkcí δ . Pro každý symbol načtený ze vstupního slova se automat, na základě aktuálního stavu a právě přečteného symbolu, přesune do nového stavu. Tato operace pokračuje, dokud nejsou přečteny všechny symboly vstupního slova. V případě že, pro aktuální symbol a stav není definována přechodová funkce, automat svou činnost ukončí a vstupní slovo není přijato. Automat pouze využívá právě čtený symbol a aktuální stav, žádná jiná data nejsou ukládána nebo čtena. [1]

Po dokončení zpracování vstupního slova automat vyhodnotí, zda stav, ve kterém automat skončil, patří do množiny koncových stavů. Pokud ano, tak je slovo přijato, a tedy náleží do jazyka rozpoznávaného daným automatem. V opačném případě je vstupní slovo zamítnuto a do rozpoznávaného jazyka nepatří [2].

Uvažujeme konečný automat pro detekci, zda slovo obsahuje lichý počet písmena „a“.

$M = (Q, \Sigma, \delta, q_0, F)$, kde $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $F = \{q_1, q_2\}$, $w = abaa$

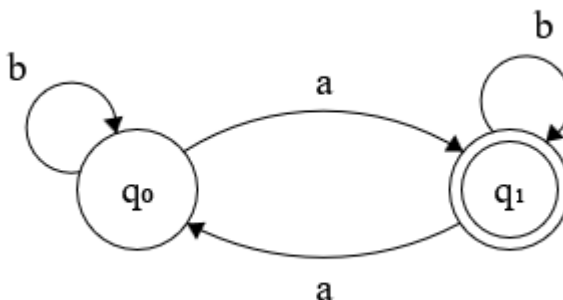
$$\delta(q_0, a) = q_1$$

$$\delta(q_0, b) = q_0$$

$$\delta(q_1, a) = q_0$$

$$\delta(q_1, b) = q_1$$

Tento automat můžeme zobrazit pomocí stavového diagramu jako:



Obrázek 2 Automat pro detekci, zda obsahuje slovo lichý počet písmen a

Jestliže automat začíná svou činnost se vstupním slovem $w = „abaa“$ na pásce, pak přečte první symbol z pásky a zkusí v přechodové funkci najít přechod, který obsahuje daný symbol. Stav q_0 (počáteční stav) a čtení „a“ odpovídá přechod $\delta(q_0, a) = q_1$. Automat proto přejde do stavu q_1 a čtecí hlava se posune na další symbol. Nyní automat čte symbol „b“ a nachází se ve stavu q_1 . Tomu odpovídá přechod $\delta(q_1, b) = q_1$. Na základě tohoto přechodu se automat přesune do stavu q_1 . Jelikož se, ale automat již ve stavu q_1 nachází, zůstane v tomto stavu. Další symbol na pásce je „a“ čemuž odpovídá přechod $\delta(q_1, a) = q_0$. Automat se přesune do stavu q_0 . Na pásce se nachází poslední symbol „a“, takže bude použit přechod $\delta(q_0, a) = q_1$. Automat se tedy přesune do stavu q_1 a zde skončí. Jelikož stav q_1 patří mezi stavy koncové, slovo je přijato. V tomto případě to znamená, že slovo obsahuje lichý počet písmen „a“.

1.2 Nedeterministický konečný automat

Nedeterministický konečný automat lze formálně definovat podobně jako deterministický konečný automat, avšak s významným rozdílem v přechodové funkci δ . Zatímco u deterministického automatu přechodová funkce dvojici stav a symbol přiřazuje stav, u nedeterministického automatu se jedná o funkci, která každé dvojici stavu a symbolu (včetně možnosti tzv. ε -symbolu) přiřazuje množinu možných stavů. Formálně lze tuto funkci zapsat jako:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$$

kde Q je množina stavů, Σ je vstupní abeceda, ε symbolizuje možnost přechodu bez čtení vstupního symbolu a 2^Q označuje množinu všech podmnožin množiny Q [3].

Významnou vlastností takto definovaných nedeterministických automatů je možnost tzv. ε -přechodů, což jsou speciální přechody, při kterých automat nečte žádný symbol ze vstupní pásky, ale pouze změní svůj aktuální stav. Při tomto přechodu tedy nedochází ke změně slova na vstupní pásce, ani ke změně pozice čtecí hlavy [1].

Pokud se automat ocitne v situaci, kdy přechodová funkce umožňuje z aktuálního stavu přejít do více možných stavů, automat nedeterministicky prochází všechny možné cesty najednou. Tuto situaci lze chápat jako rozvětvení činnosti automatu na více paralelních větví, kde každá větev reprezentuje automat, který se přesune do jednoho z možných stavů. Pokud alespoň jedna z těchto větví skončí po přečtení vstupního slova v některém z koncových stavů, vstupní slovo se považuje za akceptované, a tedy patří do jazyka rozpoznávaného nedeterministickým konečným automatem. V opačném případě, kdy žádná větev nedosáhne koncového stavu, je vstupní slovo zamítnuto a není součástí rozpoznávaného jazyka [2].

2 PRAVĚ JEDNOSMĚRNÝ SKÁKAJÍCÍ KONEČNÝ AUTOMAT

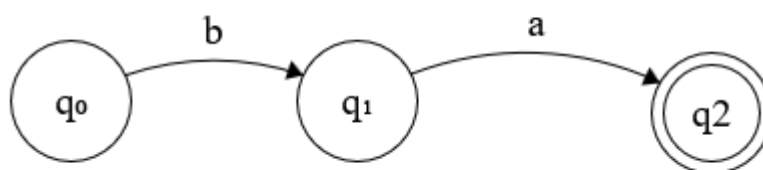
2.1 Deterministický pravě jednosměrný skákající automat

Jedná se o rozšíření deterministického konečného automatu, které mu umožňuje provádět přesun (skok) čtecí hlavy (doprava) bez změny aktuálního stavu. Tento skok nastává, když se automat nachází ve stavu, v němž nemůže pro aktuálně čtený symbol provést žádný přechod definovaný přechodovou funkcí.

$$M = (Q, \Sigma, \delta, q_0, F), \text{ kde } Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, F = \{q_2\}, w = ab$$

$$\delta(q_0, b) = q_1$$

$$\delta(q_1, a) = q_2$$



Obrázek 3 Automat pro ukázkou skoku

Příklad: Pokud bychom si představili deterministický konečný automat, který se nachází ve stavu q_0 , a právě čte symbol „a“, situace, kdy přechodová funkce neumožňuje žádný další přechod, by znamenala zastavení automatu v aktuálním stavu. V tomto případě by automat skončil ve stavu q_0 , aniž by přečetl celé vstupní slovo, což vede k zamítnutí tohoto vstupního slova.

Naopak, pokud použijeme pravě jednosměrný skákající konečný automat, situace se významně mění. Automat ve stavu q_0 se symbolem „b“ může provést skok čtecí hlavy na nejbližší symbol „b“, který ve vstupním slově následuje za aktuální pozicí. Automat tak může část vstupního slova přeskočit, a to i opakovaně. Po dosažení konce vstupního slova se automat vrací na začátek nezpracované (tj. přeskočené) části vstupního slova a pokračuje ve své činnosti, dokud není slovo kompletně přečteno. Po skoku čte tedy automat symbol „b“, který lze v aktuálním stavu zpracovat. Jakmile automat přečte tento

symbol, může podle přechodové funkce provést přesun do nového stavu, například q_1 . V dalším kroku může automat pokračovat ve čtení symbolů, a následně přejít do koncového stavu q_2 . V tomto případě by vstupní slovo bylo automatem přijato, neboť se automat úspěšně dostal do koncového stavu po přečtení celého vstupního slova.

Jednosměrné skákající automaty tedy přinášejí možnost efektivnějšího a flexibilnějšího zpracování vstupních slov oproti klasickým deterministickým automatům tím, že dovolují překonávat překážky dané absencí definovaných přechodů pro konkrétní symboly v určitých stavech [4].

2.2 Nedeterministický právě jednosměrný skákající konečný automat

Nedeterministický právě jednosměrně skákající konečný automat, dále jen NPJSKA, představuje rozšíření PJSKA o nedeterminizmus podobně jako u klasických konečných automatů s rozdílem v ε -přechodech.

Dle článku [4] existuje více implementací skoků a ε -přechodů pro NPJSKA:

1. skok je proveden pouze v případě, kdy není možné provést ε -přechod,
2. automat vybere, zda provede ε -přechod nebo skok, ale pokud provede skok, skočí (stejně jako PJSKA) pouze na pozici nejbližšího symbolu, který umí v aktuálním stavu zpracovat,
3. automat vybere, zda provede ε -přechod, nebo skok, při kterém zůstane v aktuálním stavu a přeskočí jediný symbol.

Uvažujme následující NPJSKA

$M = (Q, \Sigma, \delta, q_0, F)$, kde $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$, $F = \{q_4\}$, $w = bba$

$$\delta(q_0, a) = \{q_1\}$$

$$\delta(q_1, b) = \{q_1\}$$

$$\delta(q_1, a) = \{q_1\}$$

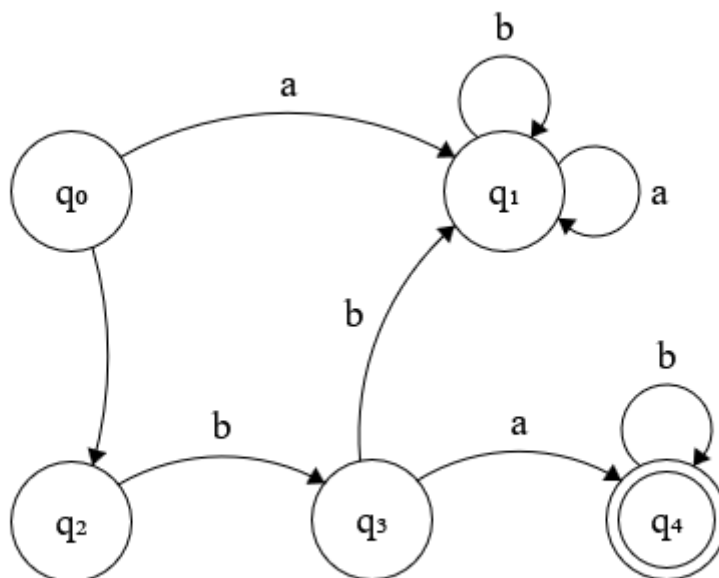
$$\delta(q_2, b) = \{q_3\}$$

$$\delta(q_3, b) = \{q_1\}$$

$$\delta(q_3, a) = \{q_4\}$$

$$\delta(q_0, \varepsilon) = \{q_2\}$$

$$\delta(q_4, b) = \{q_4\}$$



Obrázek 4 NPJKA s různou implementací ϵ -přechodů

Pokud použijeme implementaci 1, tak automat ve stavu q_0 provede ϵ -přechod do stavu q_2 , ve stavu q_2 přečte symbol „b“, přesune se do q_3 , kde přečte „b“ a přesune se do q_1 , kde po přečtení „a“ zůstane. Tedy automat skončí ve stavu q_1 , který nepatří mezi koncové stavy, proto je slovo odmítnuto.

Implementace dvě provede skok na symbol „a“ a následně se přesune do stavu q_1 , kde po přečtení symbolu „b“ zůstane. Další symbol je také „b“, takže zůstane ve stavu q_1 . Také je možná větev kdy automat ve stavu q_0 provede ϵ -přechod do stavu q_2 . Tato větev je projita stejně jako u implementace jedna a skončí ve stavu q_1 . Jelikož ani jedna z větví nekončí v koncovém stavu, je slovo zamítnuto.

V implementaci tři automat posune svou čtecí hlavu na symbol „b“ a následně provede ϵ -přechod do stavu q_2 , kde přečte „b“ a přesune se do stavu q_3 . Ve stavu q_3 přečte „a“ a přesune se do stavu q_4 , který patří mezi koncové stavy. Slovo je proto přijato. I v této implementaci je větev kdy automat provede ϵ -přechod do stavu q_2 a skončí v nekonečném stavu. Pro přijetí vstupního slova ale stačí, aby pouze jedna z větví slovo přijala.

Jelikož různé implementace ϵ -přechodů vedou k různým slovům přijímaných automatem, je nutné specifikovat, kterou implementaci používáme. V praktické části mé bakalářské práce využívám implementaci číslo 1.

2.3 Souvislost s jazyky Chomského hierarchie

Za jazyky Chomského hierarchie považujeme jazyky

- regulární
- bezkontextové
- kontextové
- rekurzivně vyčíslitelné

Jazyky regulární jsou přijímané konečnými automaty, bezkontextové jsou přijímané zásobníkovými automaty, kontextové jsou přijímané lineárně omezenými automaty a rekurzivně vyčíslitelné jsou přijímané Turingovými stroji.

Vztahy mezi jazyky Chomského hierarchie a jazyky přijímanými právě jednosměrnými skákajícími konečnými automaty jsou popsány následujícími větami.

- Třída jazyků přijímaných PJSKA je vlastní podtřídou třídy kontextových jazyků. [5]
- Třída regulárních jazyků je vlastní podtřídou třídy jazyků přijímaných PJSKA.[6]
- Třídy jazyků přijímaných PJSKA a bezkontextových jazyků jsou nesrovnatelné.[6]

Regulární jazyky jsou přijímané standardními konečnými automaty. Z druhého z uvedených vztahů tak plyne, že PJSKA mají větší výpočetní sílu než konečné automaty. Jazyky přijímané PJSKA a bezkontextové jazyky jsou nesrovnatelné, což znamená, že existuje bezkontextový jazyk, který není přijímaný žádným PJSKA, a naopak.

II PRAKTICKÁ ČÁST

3 TVORBA APLIKACE

Aplikace byla programována v jazyce Python, specificky ve verzi 3.13. Jako vývojové prostředí byl zvolen editor Visual Studio Code, který nabízí širokou škálu rozšíření, pohodlnou správu projektů a integrovaný terminál, což výrazně usnadňuje vývoj.

Volbu jazyka motivovaly následující faktory.

1. Platformní nezávislost – Python je multiplatformní jazyk, což umožňuje provozovat výslednou aplikaci na různých operačních systémech (Windows, Linux, macOS) bez potřeby výraznějších úprav zdrojového kódu a bez nutnosti vydávat více verzí aplikace. [7]
2. Srozumitelnost kódu – Python je znám svou přehlednou syntaxí, což je výhodné nejen při samotném vývoji aplikace, ale i při dalším možném rozšíření aplikace v budoucnu.
3. Dostupnost knihoven – Python disponuje velkým množstvím otevřených knihoven, které pokrývají širokou škálu oblastí, od vědeckých výpočtů až po grafické uživatelské rozhraní.

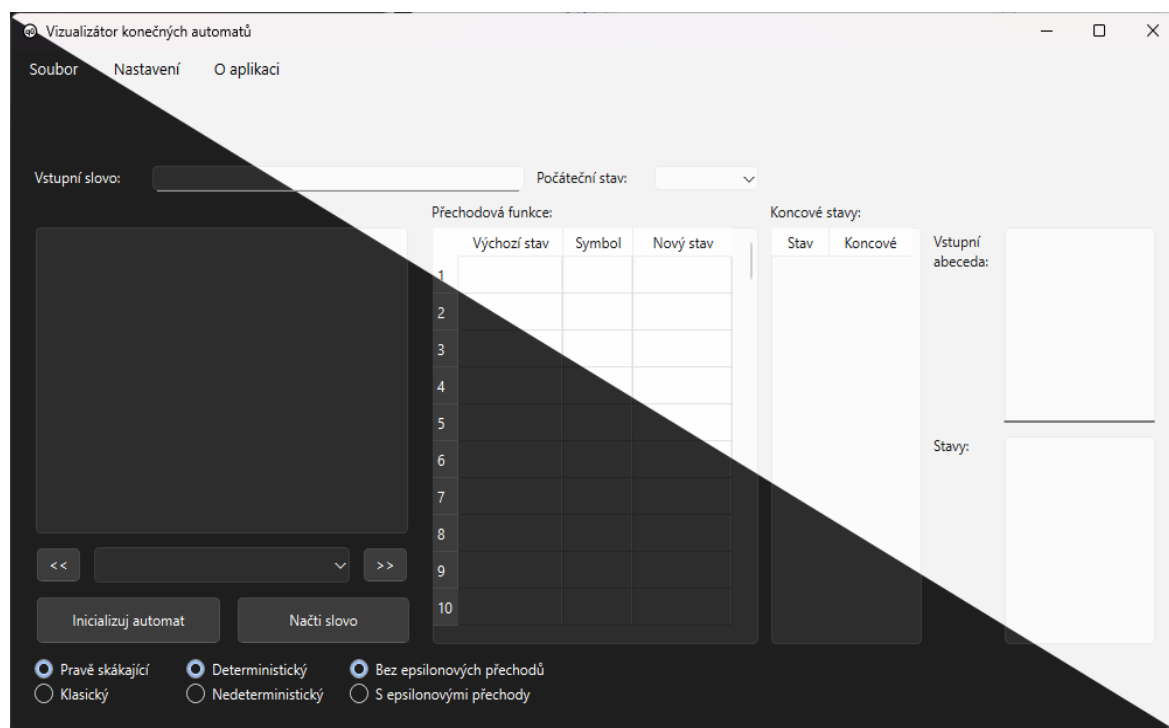
Grafická část programu je tvořena ve Frameworku PyQt6, což je Python vazba na multiplatformní knihovnu Qt. Ta byla využita pro její možnost tvorby grafického rozhraní nejen s pomocí předefinovaných objektů, ale také možnosti kreslení vlastní grafiky. Návrh samotného uživatelského rozhraní byl vytvořen pomocí nástroje Qt Designer, který umožňuje vizuální návrh GUI pomocí přetahování prvků. Vytvořený návrh je uložen ve formátu .ui, ze kterého je následně pomocí PyQt převeden do Pythonu.

3.1 GUI

Při vytváření grafického rozhraní byly brány v potaz veškeré funkce, které musí automat podporovat. Funkce lze rozdělit do tří kategorií, a to nastavení aplikace, zadávání konfigurace automatu a výpočet automatu. Bylo zvoleno rozložení aplikace, kdy nastavení samotné aplikace a méně používané funkce jsou schované v menu.

Pro grafické zobrazení bylo zvoleno nastavení barev dle operačního systému a možnosti změny motivu v nastavení. Pokud je systém nastaven tak, aby používal tmavý mód, aplikace se také zobrazí v tmavém módu. Toto chování lze vypnout v nastavení aplikace. Kromě barev jsou takto automaticky zobrazeny prvky aplikace, aby odpovídaly

vzhledu systému. Například ve Windows 10 jsou využity prvky s ostrými hranami a ve Windows 11 zakulacená tlačítka. Některé operační systémy nejsou kompatibilní se všemi motivy. Nejčastěji se problémy s kompatibilitou vyskytují na operačních systémech na jádře Linux, kde často nefunguje automatická detekce barvy, ale přepínání funguje. Na starších verzích operačního systému Windows se vyskytuje často opačný problém, motiv se nastaví korektně, ale není ho možné změnit.



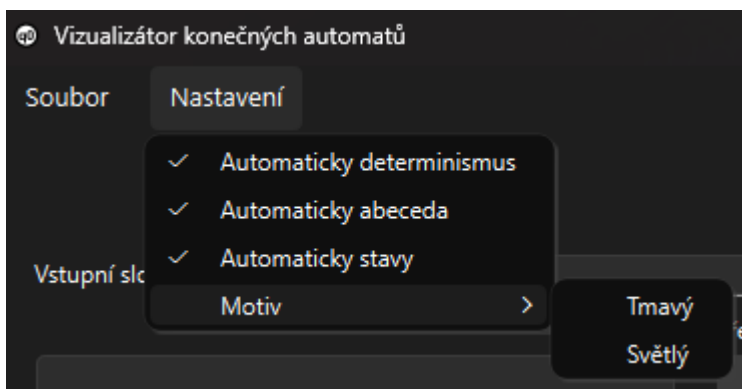
Obrázek 5 Porovnání tmavého a světlého módu aplikace

3.1.1 Hlavní panel

Horní panel slouží k nastavení chování aplikace a přístupu k málo používaným funkcím. Záložka soubor slouží k ukládání a načítání automatu ze souboru. Záložka nastavení slouží k nastavení chování aplikace. Tyto nastavení jsou:

- Automaticky determinismus – při zaškrtnutí této možnosti aplikace automaticky vybere v dolním menu správné nastavení determinismu.
- Automaticky abeceda – při zaškrtnutí této možnosti se do vstupní abecedy automaticky přidají veškeré symboly, které se vyskytují v přechodové funkci a neoznačují stavy.

- Automaticky stavy – při zaškrtnutí automaticky přidá všechny stavy z přechodové funkce do množiny stavů.
- Motiv – slouží k nastavení motivu aplikace, je na výběr mezi tmavým, světlým a automatickým motivem. Při nastavení motivu na automatický se vybere tmavý nebo světlý motiv, dle nastavení operačního systému.



Obrázek 6 Nastavení aplikace

Panel je typu QMenuBar, kde jednotlivé podmenu jsou typu QAction. Pole v nabídce jsou QAction. U prvků, které se dají zaškrťávat, je nastavena možnost checkable, která nastavuje zaškrťávání.

3.1.2 Nastavení konfigurace automatu

Pro nastavení automatu bylo nutné zvolit více různých vstupních prvků pro zadávání jednotlivých částí konfigurace automatu. Vstupní slovo je zadáváno QLineEdit, jedná se o prvek, který slouží k zadávání jednořádkové textové informace. Vstupní abeceda a stavy jsou zadávány do QPlainTextEdit. Tento prvek má zapnuté automatické zalamování textu v případě přesáhnutí velikosti řádku.

Pro přechodovou funkci byla použita tabulka, tedy QTableWidgetItem. V tabulce jsou tři sloupce: Výchozí stav, Symbol, Nový stav. Tabulka má v základu 99 řádků. Pro koncové stavy je také využita tabulka, avšak místo textu obsahuje sloupec pouze zaškrťovací políčka pro zvolení, zda se jedná o koncový stav nebo ne. Tabulka má v základu 0 řádků, ale při přidání stavů do textového pole stavů se automaticky doplní a objeví se každý stav. Název stavu ve sloupci stav se v této tabulce nedá upravovat, pro úpravu je nutné provést změny v poli stavy.

Následuje výběr typu automatu, který se pro procházení bude používat. Může se vybírat mezi:

- právě skákajícím nebo klasickým
- deterministickým nebo nedeterministickým
- s nebo bez ε -přechodů

Toto nastavení je řešeno formou QRadioButton. Tento prvek má možnost „exclusive“, díky které je možné označit pouze jedno z tlačítek ve skupině. Dvě možnosti (Deterministický a nedeterministický, Právě skákající a klasický), jsou společně ve skupině. Není tedy možné vybrat zároveň deterministický a nedeterministický, zároveň ale tato skupina nemá vliv na žádnou z ostatních skupin. Je tedy možné vybrat nedeterministický a klasický. Skupiny jsou řešeny pomocí QButtonGroup.

3.1.3 Vizualizace

Zbývajíc část je určena pro zobrazení vizualizace. Dělí se na dvě části. První část se skládá z textového pole QTextEdit, u kterého je zamknuto vpisování textu, a slouží pouze k výpisu informací. Informace jsou vypisovány po řádcích, kdy jednotlivé řádky se skládají z názvu informace a informace samotné.

Druhá část se skládá ze zobrazení abecedy. Skládá se z písmen ve čtvercovém ohraničení, které mění barvu v závislosti na stavu procházení. Tyto prvky jsou dynamicky generovány pomocí kódu, proto budou popsány v následující kapitole.

Další z prvků pro vizualizace je ovládací panel. Ten slouží k pohybu ve vizualizaci o krok dopředu a zpět. V případě nedeterministického automatu také umožňuje výběr větve, která bude vizualizována. Pro pohyb jsou využity dvě tlačítka typu QPushButton, které jsou označeny symboly „>“ pro posun dopředu a „<“ pro posun zpět. Mezi tlačítky leží rozbalovací seznam QComboBox, který slouží k výběru větve. Prvky jsou do toho seznamu automaticky doplňovány.

4 POPIS APLIKACE

Aplikace je rozdělena na dvě hlavní části, a to knihovnu FiniteAutomata a vlastní aplikaci, která se skládá z grafického rozhraní, které bylo popsáno v předchozí kapitole, a ze souboru main, který se stará o spojení vstupu v GUI s funkcionalitou v knihovně FiniteAutomata.

4.1 Knihovna finite automata py

Knihovna FiniteAutomata obsahuje veškeré funkce potřebné pro práci s automaty, je proto možné použít tuto knihovnu samostatně. Lze ji jednoduše integrovat do jiných projektů, nebo ji použít v samostatných skriptech. To je výhodné zejména v případech, kdy je potřeba rychle otestovat větší počet automatů, bez nutnosti jednotlivé automaty ručně zadávat do gui.

V aplikaci je využíván datový typ slovník zapsán jako `slovník = {'klíč1': 'hodnota1', 'klíč2': 'hodnota2'}`. Ten slouží k namapování klíče na jinou hodnotu. V příkladu výše je klíč „klíč1“ namapován na hodnotu „hodnota2“. Pokud tedy zavoláme `slovník['klíč1']` bude navraceno „hodnota1“. Pro kopírování slovníku je použita knihovna copy, která se používá při větvení nedeterministického automatu.

Při využívání knihovny je nutné nejprve vytvořit instanci objektu FiniteAutomaton. Tedy konkrétní objekt vytvořený podle šablony třídy, se kterým lze dále pracovat. K tomu je využita metoda `init`.

```
def __init__(self, states, alphabet, transition_function, initial_state,
accepting_states, deterministic, jumping=True):
    self.states = states
    self.alphabet = alphabet
    self.transition_function = transition_function
    self.initial_state = initial_state
    self.accepting_states = accepting_states
    self.jumping = jumping
    self.deterministic = deterministic
    self.visited_states = []
```

Obrázek 7 Metoda init

Tato funkce má argumenty:

- States – pole možných stavů např.: ['q0', 'q1', 'q2']
- Alphabet – pole možných znaků vstupní abecedy ['a', 'b', 'c']

- Transition_function – přechodová funkce je v datovém typu slovník. Jako klíč slovníku slouží dvojice stavu a symbolu nebo pouze stavu, v případě ϵ -přechodu. Tyto klíče jsou napojeny na množinu stavu, do kterých může automat přejít př.: $\{('q0', 'a'): ['q1', 'q1'], ('q1'): ['q2']\}$
- Initial_state – počáteční stav automatu ve formátu string př. „q0“
- Accepting_states – pole koncových stavů např.: ['q0', 'q3']
- Deterministic – zda je automat deterministický (proměnná True/False)
- Jumping – zda se jedná o skákající automat (proměnná True/False)

Při vytvoření instance automatu se nastaví dané hodnoty konfigurace a navštívené stavy se nastaví na prázdný slovník tedy {}.

Po nastavení konfigurace automatu je nutné načíst vstupní slovo. Při načítání je kontrolováno, zda vstupní slovo obsahuje pouze symboly ze vstupní abecedy. Pokud slovo obsahuje symboly, které ve vstupní abecedě nejsou, skončí načítání slova s chybou "Input symbol not in alphabet". Při načtení slova jsou nastaveny navštívené stavy na prázdné pole. Vstupní slovo je ukládáno ve slovníku, kdy každý symbol ze slova je uložen v tomto formátu: {'symbol': 'a', 'used': False}, kde symbol je daný symbol a used je proměnná true/false, která určuje, zda byl již symbol přečten.

```
def load_input(self, input_string):
    self.position=0
    self.word=[]
    for input in input_string:
        if input not in self.alphabet:
            raise Exception("Vstupní slovo obsahuje symbol, který není v abecedě automatu")
        self.word.append({'symbol':input, 'used':False})
    self.current_state = self.initial_state
    self.visited_states = []
```

Obrázek 8 Kód metody pro načítání slov

Pro spuštění automatu použijeme metodu run. Tato metoda je volána na instanci třídy finiteAutomata. Průběh krokování automatu je rozdílný v případě deterministického a nedeterministického automatu. V případě deterministického automatu je spuštěna smyčka while True, ve které probíhá celé krokování automatu. Nejprve je nastaven předchozí stav na současný stav, poté vytvořena kopie slovníku slova a lokální proměnná pozice, ve které je uchována pozice čtecí hlavy. Nyní proběhne samotné krokování automatu pomocí metody

step, která bude popsána v následující kapitole. V případě nedeterministického automatu se pouze zavolá metoda `step_nondeterministic`.

```
def run(self):
    if self.deterministic:
        output=[]
        while True:
            previousState=self.current_state
            position=self.position
            step=self.step()
            wordcopy=copy.deepcopy(self.word)
            output.append({'action': step, 'state': previousState, 'word': wordcopy, "headPosition": position})
            if step == "Konec":
                output[-1]['headPosition']-=1
                output[-1]['accepting']=self.current_state in self.accepting_states
                break
            if step == "Zacyklení" or step == "Žádný možný přechod":
                output[-1]['accepting']=False
                break
        else:
            output = [{'action': "Start", 'state': self.current_state, 'word': self.word, 'headPosition': 0,
            'accepting': None, 'new_state':
            'initeAutomaton.step_nondeterministic(self.word,0,self.transition_function,self.current_state,self.accepting_state
            ], self.jumping)}]
            output[0]['accepting']=any(child['accepting'] for child in output[0]['new_state'])
            return output
```

Obrázek 9 Kód metody run

4.1.1 Činnost deterministického automatu

Metoda `step` nejprve zkontroluje, zda nakonfigurovaný automat je deterministický či nikoliv. Pokud se jedná o nedeterministický automat, ohlásí chybu, poté vyzkouší, zda aktuálně čtený symbol je již přečten, pokud ano, tak automat skončí a metoda vrátí akci „end“.

Nejprve je nutné zkontrolovat, zda automat již byl ve stejném stavu ve slově se stejnými přečtenými znaky a stejnou pozicí čtecí hlavy, je tedy nutné činnost automatu ukončit, protože další procházení by vedlo k zacyklení.

V případě, že by nebylo zacyklení zabráněno, automat by nikdy neukončil své procházení, a program by neustále zapisoval do proměnné `output` cestu automatu, což by mělo za následek běh aplikace až do okamžiku, kdy by došla paměť na zápis cesty a aplikace by selhala. Pro zabránění zacyklení je činnost automatu ukončena s akcí: „Zacyklení“. Tedy je nekonečná smyčka ukončena a projitá cesta je odeslána volající funkcí.

Pokud aktuální dvojice stavu a symbolu nebyla navštívena, přidám ji do slovníku a nastavím ji jako navštívenou.

Nový stav získám tak, že vytvoříme dvojici aktuálního stavu a aktuálně čteného symbolu. Tuto dvojici předáme metodě „get“, pomocí které získáme pole možných stavů z přechodové funkce. Protože se jedná o deterministický automat, je v poli pouze jeden nebo žádný stav. V případě, že pole nových stavů je prázdné a automat je nakonfigurován jako

skákající, automat provede skok posunem na další symbol pomocí metody getNextSymbol, která bude popsána v příští kapitole. Metoda vrátí jumping, pokud provede skok. Metodu step lze využívat i bez metody run, například k procházení automatu z nějakého stavu nebo k procházení postupně pouze o jeden krok. Při použití metody run je vrácen list přechodů automatu.

```
def step(self):
    if not self.deterministic:
        raise Exception("Cannot step through a nondeterministic automaton")
    if len(self.word) == 0:
        return "Konec"
    if self.word[self.position]['used']:
        return "Konec"
    state_word_pair = (self.current_state, FiniteAutomaton.getUsed(self.word), self.position)
    if state_word_pair in self.visited_states:
        return "Zacyklení"
    symbol = self.word[self.position]['symbol']
    self.visited_states.append(state_word_pair)
    new_state = self.transition_function.get((self.current_state, symbol))
    if new_state is None:
        if self.jumping:
            self.position = FiniteAutomaton.getNextSymbol(self.word, self.position)
            return "Skok"
        return "Žádný možný přechod"
    self.current_state = new_state[0]
    self.setUsed()
    return f"Přechod do {self.current_state} při čtení {symbol}"
```

Obrázek 10 Kód metody step

4.1.2 Činnost nedeterministického automatu

Činnost nedeterministického automatu se odlišuje od deterministického tím, že automat prochází více větví najednou, není tedy možné automat krokovat postupně. Metoda step_nondeterministic po spuštění vrátí celý průchod automatu. Výstup je strom v tomto formátu {'action': '', 'state': '', 'word': word, 'headPosition': position, 'accepting': None, 'new_state': []}, kde v new_state je pole možných akcí (přechodů nebo skoků čtecí hlavy). Při každém kroku je tedy rekurzivně volána metoda step_non_deterministic. Jediný případ, kdy není metoda rekurzivně volána, je případ, kdy je celé slovo přečteno nebo při zabrání zacyklení.

Funkce není volána na instanci Finiteautomaton, proto je nutné předat potřebná data pomocí argumentu:

- Word – vstupní slovo
- Position – pozice čtecí hlavy
- Transition_function – přechodová funkce

- Current_state – název aktuálního stavu ve formátu string
- Accepting_states – pole koncových stavů
- Visited_states – list navštívených stavů

```
def step_nondeterministic(word, position, transition_function, current_state, accepting_states=[],
visited_states=[], jumping=True):
    output = []
    state_word_pair = (current_state, FiniteAutomaton.getUsed(word), position)
    if len(word) == 0:
        output.append({'action': "Konec", 'state': current_state, 'word': word, 'headPosition': position,
'accepting': current_state in accepting_states, 'new_state': None})
        return output
    if word[position]['used']:
        output.append({'action': "Konec", 'state': current_state, 'word': word, 'headPosition': position,
'accepting': current_state in accepting_states, 'new_state': None})
        return output

    if state_word_pair in visited_states:
        output.append({'action': "Zacyklení", 'state': current_state, 'word': word, 'headPosition': position,
'accepting': False, 'new_state': None})
        return output
    visited_states.append(state_word_pair)

    symbol = word[position]['symbol']
    new_states = transition_function.get((current_state, symbol))
    epsilon_states = transition_function.get(current_state)
    if new_states is None and epsilon_states is None:
        if jumping:
            stateInfo = {'action': "Skok", 'state': current_state, 'word': word, 'headPosition': position,
'accepting': None, 'new_state': FiniteAutomaton.step_nondeterministic(word, FiniteAutomaton.getNextSymbol(word,
position), transition_function, current_state, accepting_states, visited_states, jumping)}
            stateInfo['accepting'] = any(child['accepting'] for child in stateInfo['new_state'])
            output.append(stateInfo)
        else:
            output.append({'action': "Žádný možný přechod", 'state': current_state, 'word': word,
'headPosition': position, 'accepting': False, 'new_state': None})
    else:
        if new_states is not None:
            for state in new_states:
                stateWord = copy.deepcopy(word)
                stateWord[position]['used'] = True
                stateInfo = {'action': f"Přechod do {state} při čtení {symbol}", 'state': current_state, 'word':
stateWord, 'headPosition': position, 'accepting': None, 'new_state':
FiniteAutomaton.step_nondeterministic(stateWord, FiniteAutomaton.getNextSymbol(stateWord, position),
transition_function, state, accepting_states, visited_states, jumping)}
                stateInfo['accepting'] = any(child['accepting'] for child in stateInfo['new_state'])
                output.append(stateInfo)
            if epsilon_states is not None:
                for state in epsilon_states:
                    stateInfo = {'action': f"Epsilon Přechod do {state}", 'state': current_state, 'word': word,
'headPosition': position, 'accepting': None, 'new_state': FiniteAutomaton.step_nondeterministic(word, position,
transition_function, state, accepting_states, visited_states, jumping)}
                    stateInfo['accepting'] = any(child['accepting'] for child in stateInfo['new_state'])
                    output.append(stateInfo)
        return output
```

Obrázek 11 Kód metody step_non_deterministic

Automat začne svou činnost nastavením výstupu output na prázdné pole a vygenerováním dvojice state word pair, stejně jako v deterministickém automatu. Následně provede kontrolu, zda již byl současný stav, v kombinaci se slovem, navštíven. Pokud byl, tak zde ukončí automat svou činnost se slovy „Zacyklení“, a slovo je tedy nepřijato. Pokud navštíven nebyl, přidá se mezi navštívené, a pokračuje v procházení.

Nyní se automat pokusí najít, zda existuje v přechodové funkci přechod s aktuálním stavem a symbolem. Použitím funkce „get“ získá množinu cílových stavů. Automat se

pokusí získat ε -přechody z daného stavu, pokud existují. V případě, že nejsou žádné možné přechody, automat provede skok. Pomocí metody getNextSymbol automat získá další, ještě nepřečtený symbol ve vstupním slovu, a rekurzivně opět zavolá metodu step_nondeterministic pro automat ve stavu po skoku. V případě změny využívané implementace ε -přechodů, jak je popsáno v článku [4], by bylo nutné provést změnu zde v rozhodování, zda automat provede skok a popřípadě ve způsobu získávání symbolu, na kterém automat skončí v případě skoku.

Pokud existuje alespoň jedna akce, kterou automat může vykonat (přechod nebo skok), automat pro každou z akcí rekurzivně zavolá metodu step_nondeterministic. V případě, že existuje více než jedna možná akce, dojde k rozvětvení programu.

Na konci běhu metoda vrátí výsledek procházení pro aktuální stav, do kterého se vloží výsledky procházení pro jednotlivé rekurzivně volané větve programu. Výsledná struktura je tedy typu strom, kde jednotlivé stavy mají:

- Action – akce, kterou právě automat vykonal (přechod do x, skok)
- State – aktuální stav, ve kterém se automat nachází
- Word – stav aktuálně procházeného slova
- HeadPosition – aktuální pozice čtecí hlavy
- Accepting – True nebo False dle toho, zda aktuální větev může skončit akceptováním slova
- New_state – množina všech možných akcí, které je možné vykonat z aktuálního stavu (zde dochází k rozvětvení programu)

```

{
  "action": "Start",
  "state": "q0",
  "word": [ ],
  "headPosition": 0,
  "accepting": true,
  "new_state": [
    {
      "action": "Přechod do q1 při čtení a",
      "state": "q0",
      "word": [ ],
      "headPosition": 0,
      "accepting": true,
      "new_state": [
        {
          "action": "Přechod do q2 při čtení b",
          "state": "q1",
          "word": [
            {
              "symbol": "a",
              "used": true
            },
            {
              "symbol": "b",
              "used": true
            },
            {
              "symbol": "a",
              "used": false
            },
            {
              "symbol": "a",
              "used": false
            }
          ],
          "headPosition": 1,
          "accepting": true,
          "new_state": [ ]
        }
      ],
      "headPosition": 1,
      "accepting": true,
      "new_state": [ ]
    }
  ],
  "accepting": true,
  "new_state": [
    {
      "action": "Přechod do q0 při čtení a",
      "state": "q0",
      "word": [ ],
      "headPosition": 0,
      "accepting": true,
      "new_state": [
        {
          "action": "Skok",
          "state": "q0",
          "word": [ ],
          "headPosition": 1,
          "accepting": true,
          "new_state": [
            {
              "action": "Přechod do q1 při čtení a",
              "state": "q0",
              "word": [ ],
              "headPosition": 0,
              "accepting": true,
              "new_state": [
                {
                  "action": "Skok",
                  "state": "q0",
                  "word": [ ],
                  "headPosition": 1,
                  "accepting": true,
                  "new_state": [ ]
                }
              ]
            }
          ]
        }
      ],
      "headPosition": 1,
      "accepting": true,
      "new_state": [ ]
    }
  ],
  "accepting": true,
  "new_state": [ ]
}

```

Obrázek 12 Ukázka struktury strom (pro přehlednost jsou některé větve skryty)

Metodu `step_nondeterministic` není možné použít k procházení pouze o krok, jako u deterministického automatu. Při spuštění automaticky proběhne celá činnost automatu ze zadaného stavu až do konce nebo zacyklení všech výpočetních větví.

4.2 Funkce pro práci s konečným automatem

4.2.1 getItem

Pro získávání jednotlivých kroků automatu ze stromu používáme metodu `getItem`. Jedná se o jednoduchou metodu, která umožňuje pomocí souřadnic získat jednotlivé prvky stromu. Souřadnice jsou ukládány ve formátu `[0,5,1]`, kde začínáme v kořenu stromu a následně každé číslo je výběr větve. Například pokud je použit strom viz Obrázek 12 a index `[0,1]`, bude vybrána větev s indexem 0 tedy větev s akcí start. Následně bude vybrána z `new_state` větev s indexem 1, tedy „přechod do `q0` při čtení „a“ a tato větev bude vrácena.

```
def getItem(tree, index):  
    if len(index) == 1:  
        return tree[index[0]]  
    return FiniteAutomaton.getItem(tree[index[0]]['new_state'], index[1:])
```

Obrázek 13 Kód získání prvků

4.2.2 getNextSymbol

Metoda slouží k získání příštího nepřechteného symbolu ze vstupní abecedy. V případě, že již byly všechny symboly přechteny, vrátí chybovou hlášku. Metoda není volána na instanci vstupního slova z důvodu využití v nedeterministickém automatu. Vrátí tedy pouze číslo nové pozice čtecí hlavy.

```
def getNextSymbol(word, position):  
    i = position + 1  
    if i >= len(word):  
        i = 0  
    while word[i]['used']:  
        if i == position:  
            return position  
        i += 1  
        if i >= len(word):  
            i = 0  
    return i
```

Obrázek 14 Kód získání symbolu

4.2.3 setUsed

Metoda označí symbol na pozici čtecí hlavy jako přečtený. Po nastavení symbolu jako přečtený, zavolá výše popsanou metodu getNextSymbol, pro získání nové pozice čtecí hlavy.

```
def setUsed(self):  
    if self.word[self.position]['used']:  
        raise Exception("Symbol již byl použit")  
    self.word[self.position]['used']=True  
  
    self.position=FiniteAutomaton.getNextSymbol(self.word,self.position)  
    return True
```

Obrázek 15 Kód metody pro nastavení symbolu jako použitého

4.3 Aplikace

Soubor main.py napojuje uživatelské rozhraní navrhnuté v předchozím kroku na knihovnu FiniteAutomata.py.

Main využívá tyto knihovny:

- Os – získání cesty k souboru s definicí GUI
- Sys – získání argumentů a ukončování GUI
- Json – ukládání a načítání konfigurace ve formátu json
- PyQt6– práce s GUI

Při spuštění aplikace proběhne následující kód, který spustí grafické rozhraní aplikace v knihovně PyQt6. Podmínka zkontroluje, zda se aplikace spouští přímo, a ne například pomocí importu některé metody pro využití jinou aplikací, nebo v případě použití multiprocessingu, o který může být aplikace rozšířena v budoucnu. Pokud se aplikace spouští přímo, spustíme aplikaci. Tedy specificky se vytvoří nová instance QApplication [8], které se předají argumenty sys.argv, které jsou v současném případě prázdné. Poté se vytvoří instance hlavního okna, které se předá cesta k šabloně a vytvoří se tak nový widget. Ten využije hlavní okno vytvořené v předchozím kroku. Nyní lze zobrazit okno aplikace a nastavit mu fixní velikost, tedy výšku 555 pixelů a šířku 1031 pixelů. V try catch je část kódu, který ukončí aplikaci v případě, kdy o to aplikace požádá, například kliknutí na křížek.

V případě, že aplikace je ukončena nečekaně, například pád aplikace nebo ukončení přes správce úloh, aplikace se ukončí a do terminálu vypíše hlášku „Exiting“ a chybový kód, který konec aplikace zavinil.

```
if __name__ == '__main__':  
    app = QApplication(sys.argv)  
    mainwindow = MainWindow()  
    widget = QtWidgets.QStackedWidget()  
    widget.addWidget(mainwindow)  
    widget.show()  
    widget.setWindowIcon(QIcon(os.path.join(os.path.dirname(__file__), "icon.png")))  
    widget.setWindowTitle("Vizualizátor konečných automatů")  
    widget.setFixedHeight(555)  
    widget.setFixedWidth(1031)  
    try:  
        sys.exit(app.exec())  
    except Exception as e:  
        print("Exiting"+ str(e))
```

Obrázek 16 Kód pro zapnutí aplikace

Nyní proběhne část aplikace v části `init`, která slouží k inicializaci aplikace. Proběhne tedy získání cesty k souboru šablony a následné načtení uživatelského rozhraní ze šablony. Následuje propojení tlačítek a akcí z uživatelského rozhraní s metodami, které se starají o obsluhu těchto situací.

Nastavení „clicked“ se stará o kliknutí na tlačítka nastavení, „triggered“ se stará o kliknutí na položky z horní nabídky. `FocusOutEvent` je automaticky volán v případě, že uživatel pracuje s jedním ovládacím prvkem a následně se překlikne na jiný prvek.[8] Následuje nastavení šířek jednotlivých sloupců. Toto nastavení není možné udělat přímo v šabloně, proto je provedeno zde. Jako poslední se volá metoda `clearAutomaton`, která slouží k nastavení proměnných automatu na prázdné hodnoty, slouží tedy k inicializaci a nastavení prázdných hodnot nebo ke smazání současné konfigurace automatu.

```

def __init__(self):
    super(MainWindow, self).__init__()
    ui_file = os.path.join(os.path.dirname(__file__), 'MainWindow.ui')
    loadUi(ui_file, self)
    self.Btn_step.clicked.connect(self.StepButton)
    self.Btn_stepBack.clicked.connect(self.StepBackButton)
    self.Btn_init.clicked.connect(self.initButton)
    self.Act_save.triggered.connect(self.saveButton)
    self.Act_load.triggered.connect(self.loadButton)
    self.Btn_loadWord.clicked.connect(self.loadWord)
    self.txtBx_states.focusOutEvent=self.statesFocusOut
    self.Act_dark.triggered.connect(self.setDarkTheme)
    self.Act_light.triggered.connect(self.setLightTheme)
    self.RadBtn_deterministic.toggled.connect(self.onDeterminismChanged)
    self.RadBtn_nondeterministic.toggled.connect(self.onDeterminismChanged)
    self.Act_autoAlphabet.triggered.connect(self.AutoAlphabet)
    self.Act_autoStates.triggered.connect(self.AutomaticStates)
    # Table transitions
    self.Tbl_transitionFunction.setColumnWidth(0, 90)
    self.Tbl_transitionFunction.setColumnWidth(1, 60)
    self.Tbl_transitionFunction.setColumnWidth(2, 85)
    # Table States
    self.Tbl_acceptingStates.setColumnWidth(0, 50)
    self.Tbl_acceptingStates.setColumnWidth(1, 70)
    self.Tbl_transitionFunction.focusOutEvent = self.updateAutomaticStates
    self.clearAutomaton()

```

Obrázek 17 Kód inicializace

4.3.1 Vizualizace vstupní abecedy

Nejprve se zavolá původní implementace metody `paintEvent`, aby se zajistilo, že kreslení okna proběhne správně. Metoda `paintEvent` následně zkontroluje, zda jsou načteny kroky procházení, které je možné vizualizovat. Pokud je automat deterministický, aktuální, krok je přímo vybrán podle indexu `self.Curentstep`. U nedeterministického automatu se aktuální krok získá pomocí metody `getItem`, která byla vysvětlena v předchozí kapitole, na základě cesty uložené v `self.nonDeterministicStep`. Vytvoří se instance `QPainter` pro kreslení na okno. Nastaví se počáteční souřadnice `x, y` a velikost čtverců `square_size`. Pole `self.word_positions` se vyčistí, aby se mohly uložit pozice jednotlivých symbolů. Pro každý symbol ve slově `step` ['word'] se vykreslí čtverec a text symbolu. Barva pera `pen_color` se nastaví podle toho, zda je aktivován tmavý režim `is_dark_mode` a zda byl symbol použit `letter['used']`. Souřadnice každého symbolu se uloží do `self.word_positions`. Pokud existují pozice symbolů, vykreslí se šipka nad aktuální pozicí čtecí hlavy (`step['headPosition']`). Šipka se skládá z čáry a dvou šikmých čar, které tvoří hrot šipky.

```
def paintEvent(self, event):
    super().paintEvent(event)
    if self.steps is not None:
        step=None
        if automaton.deterministic:
            step=self.steps[self.Curentstep]
        else:
            step = FiniteAutomaton.getItem(self.steps, self.nonDeterministicStep)
        painter = QPainter(self)
        x = 15
        y = 30
        square_size = 30
        self.word_positions = []
        for letter in step['word']:
            pen_color = Qt.GlobalColor.white if self.is_dark_mode() else Qt.GlobalColor.black
            pen = QPen(Qt.GlobalColor.red if letter['used'] else pen_color, 2, Qt.PenStyle.SolidLine)
            painter.setPen(pen)
            painter.drawRect(x, y, square_size, square_size)
            painter.drawText(x + 10, y + 20, letter['symbol'])
            self.word_positions.append((x, y))
            x += square_size + 10
        if self.word_positions:
            arrow_x, arrow_y = self.word_positions[step['headPosition']]
            painter.setPen(QPen(pen_color, 2, Qt.PenStyle.SolidLine))
            painter.drawLine(arrow_x + square_size // 2, arrow_y + square_size + 20, arrow_x + square_size // 2,
            arrow_y + square_size)
            painter.drawLine(arrow_x + square_size // 2, arrow_y + square_size, arrow_x + square_size // 2 - 5,
            arrow_y + square_size + 5)
            painter.drawLine(arrow_x + square_size // 2, arrow_y + square_size, arrow_x + square_size // 2 + 5,
            arrow_y + square_size + 5)
```

Obrázek 18 Kód metody paint

5 NÁVOD K APLIKACI

5.1 Instalace a spuštění

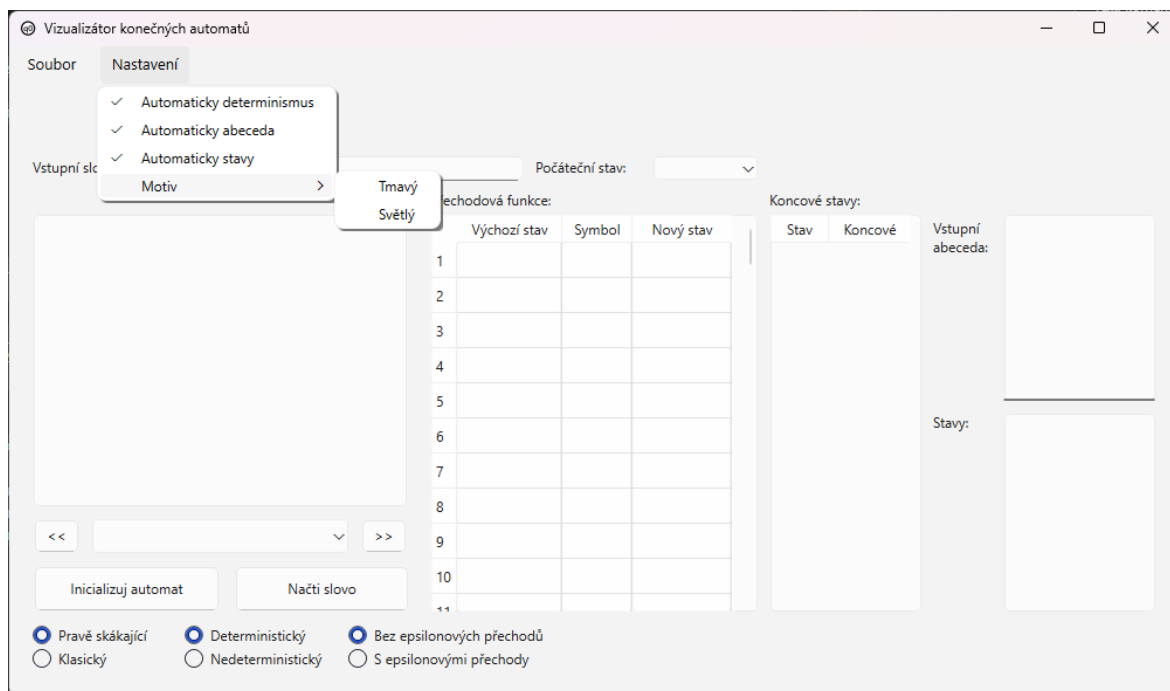
Ke spuštění aplikace je nutné mít nainstalovaný Python verzi 3.10 nebo novější a mít staženou knihovnu PyQt6. Instalaci Pythonu je možné provést prostřednictvím oficiálních webových stránek projektu Python, které se nacházejí na adrese <https://www.python.org/downloads/>. Na stránce je nabídnuta aktuální stabilní verze Pythonu. Po stažení instalačního souboru je třeba jej spustit a v úvodním kroku instalace zaškrtnout volbu „Add Python to PATH“, která zajistí, že bude možné Python spouštět z příkazového řádku bez nutnosti zadávat celou cestu k jeho umístění. Následně se pokračuje standardní instalací volbou „Install Now“, čímž dojde k instalaci Pythonu včetně nástroje pip, který slouží pro správu balíčků. Na systémech MacOS a Linux bývá často Python předinstalovaný, je proto možné tento skok přeskočit a nejdříve ověřit, zda je Python ve verzi novější než 3.10 nainstalován za pomoci příkazu v následujícím odstavci a vrátit se k instalaci v případě že požadovaný Python nainstalován není.

Po dokončení instalace je vhodné ověřit její úspěšnost spuštěním příkazového řádku na systému Windows nebo terminálu na systému MacOS a zadáním příkazu `Python --version`. Tento příkaz by měl vrátit nainstalovanou verzi Pythonu, čímž je potvrzeno, že byl Python úspěšně nainstalován a správně přidán do systémové cesty. V případě, že příkaz nevrátí nainstalovanou verzi Python, je nejspíše chyba v nezaškrtnutí možnosti „Add Python to PATH“.

Pro správné spuštění aplikace je dále nutné mít nainstalovanou knihovnu PyQt6, která slouží pro zobrazení grafického rozhraní. Tuto knihovnu lze nainstalovat prostřednictvím nástroje pip, konkrétně pomocí příkazu `„pip install PyQt6`. Instalace tímto způsobem zajistí, že budou staženy a nainstalovány všechny potřebné komponenty knihovny, včetně závislostí. Je možné, že při prvním použití bude potřeba spustit příkaz `„python -m pip install --upgrade pip“`, aby bylo zajištěno, že nástroj pip je aktuální a schopný korektně pracovat s balíčky.

Po úspěšném dokončení všech kroků je systém připraven ke spuštění samotné aplikace. Aplikaci je možné spustit kliknutím na soubor `main.pyw` v případě, že po kliknutí se aplikace nespustí, je možné aplikaci spustit rozkliknutím `run.bat`.

5.2 Nastavení aplikace



Obrázek 19 Aplikace s rozbalenou nabídkou nastavení

V nastavení aplikace lze nastavit, jakým způsobem bude automat zadáván. Je možné zvolit, které informace budou zadávány ručně, a které se mají automaticky doplnit. Automaticky je možné doplnit vstupní abecedu, stavy, determinismus a ε -přechody. Pokud chci zvolit některou z možností, stačí na ni kliknout a po jejím zvolení se na řádku před ní zobrazí fajfka. Je zde také možnost volby motivu aplikace, na výběr je světlý a tmavý. Je možné, že po kliknutí na změnu motivu aplikace, nezmění motiv, což je nejčastěji zaviněno nekompatibilním zpracováním motivů ze strany operačního systému.

5.3 Zadávání automatu

Zadávání automatu je rozděleno na dvě části. Nastavení konfigurace automatu a zadání vstupního slova.

Pro nastavení konfigurace automatu je nutné zadat vstupní abecedu a stavy, které se budou v automatu používat. Vstupní abecedu je možné zadat do textového pole s označením Abeceda. Skládá se ze skupiny symbolů (znaků). Jednotlivé znaky jsou od sebe odděleny mezerou. Každý symbol abecedy se musí skládat pouze z jednoho znaku. Jako stavy je možné použít libovolnou skupinu znaků. Nejčastěji se používá označení $q_0, q_1, q_2 \dots q_n$, jednotlivé označení stavů jsou odděleny mezerou. V názvu stavu, tedy nelze použít

mezeru. Pokud je automat nastaven tak, aby používal automatické stavy nebo automatickou abecedu bude textové pole pro zadání abecedy nebo stavů zamčeno pro zápis tento krok tedy bude přeskočen.

Přechodová funkce se do tabulky zadává pomocí trojice výchozího stavu, symbolu a nového stavu v případě, že je nastaveno, že je automat nedeterministický, je možné zadávat více přechodů se stejným výchozím stavem a symbolem. V tomto případě se každý přechod zadává zvlášť na řádek. Pokud jsou povoleny ε -přechody, je možné zadat ε -přechod tak, že se při zadávání přechodu nechá symbol prázdný, bude tedy zadán pouze výchozí stav a nový stav. V tabulce je možné smazat zadané hodnoty pomocí označení buněk tabulky a následné stisknutí klávesy delete nebo backspace.

Přechodová funkce:

	Výchozí stav	Symbol	Nový stav
1	q0	a	q1
2	q1	b	q2
3	q2		q3
4			
5			
6			
7			
8			
9			
10			
11			

Koncové stavy:

Stav	Koncové
1 q0	☒
2 q1	☐
3 q2	☐
4 q3	☐

Vstupní abeceda:

Stavy:

Obrázek 20 Ukázka zadání přechodové funkce

Dále je nutné vybrat koncové stavy, tyto stavy je možné vybrat tak, že v tabulce u každého stavu se zaškrtně políčko. Pokud je zaškrtnuto, jedná se o koncový stav, a pokud automat přečte všechny symboly ze slova a skončí ve stavu, který je označen jako koncový, bude slovo přijato.

Počáteční stav lze vybrat z rozbalovacího seznamu stavů. Je možné vybrat pouze jeden počáteční stav. Automaticky je první označen jako počáteční stav s nejnižším číslem nebo písmenem. Například ze stavů q0, q1, q2 bude vybráno q0. Při úpravě přechodové funkce bude zachován aktuálně zvolený počáteční stav.

Nyní je možné inicializovat automat pomocí tlačítka inicializuj. Pokud je nějaká chyba v konfiguraci nebo nastavení bude tato chyba ohlášena ve výstupním okénku.

Posledním krokem je zadání vstupního slova. Slovo je zadáváno do pole, po zadání vstupního slova je možné začít procházení automatu pomocí tlačítka načíst slovo. Je možné nechat slovo prázdné a vyzkoušet tak činnost automatu na prázdném vstupním slově. Pro přehledné zobrazení vizualizace je stanovena maximální délka vstupního slova na 24 znaků.

Pokud chci využít stejnou konfiguraci na jiné vstupní slovo, stačí pouze zadat nové vstupní slovo a tlačítkem „načti slovo“ ho načíst. V případě, kdy měním konfiguraci automatu, je nutné znovu inicializovat automat a poté znovu načíst slovo.

5.4 Vizualizace automatu

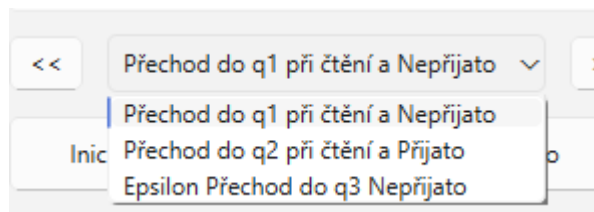
Po načtení vstupního slova automaticky začne vizualizace. V horní části vizualizace se zobrazuje vstupní slovo, každý symbol je zvlášť ve čtverci, barva symbolu označuje stav projití. Pokud je symbol červený, znamená to, že symbol byl již přečten. Pokud je bílý, přečten ještě nebyl. Šipka symbolizuje čtecí hlavu.

Pod vizualizací je textové pole výstupů, kde jsou napsány informace o aktuálním stavu procházení. Aktuální stav označuje stav, ve kterém se konečný automat nachází v aktuálním kroku. Akce je akce, kterou vyková v aktuálním stavu, například „přesun do x při čtení y“. Dále pole obsahuje informaci, zda bylo dané slovo přijato.



Obrázek 21 Vizualizační část aplikace

Pro procházení stavů se dole nachází ovládací položky, je možné posouvat se v procházení o krok dopředu a o krok zpět.

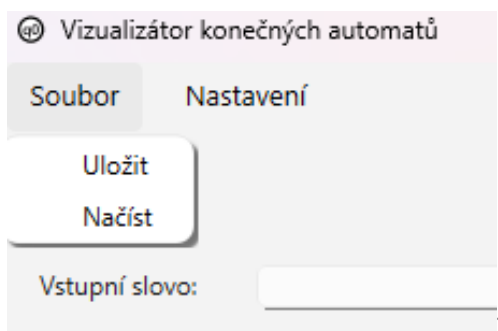


Obrázek 22 Ovládací prvek pro výběr větve

Při procházení nedeterministického automatu je GUI rozšířeno o možnost vybírat z akcí, které automat vykonal, pokud jich je více. Pokud je z možný výběr z více akcí bude informace o tom zobrazena ve výstupním poli Automaticky je vždy vybrána akce nebo jedna z akcí, která končí přijetím vstupního slova. V případě, že žádné z možných stavů nemůže skončit přijetím vstupního slova, je vybrána libovolná z ostatních akcí.

5.5 Ukládání a načítání souborů

Program umožňuje ukládání současné konfigurace pro budoucí použití, při uložení se uloží konfigurace automatu (nastavení a přechodová funkce) do souboru ve formátu json. Soubor je pak možné opětovně načíst. Jelikož je konfigurace ukládána ve formátu json, je možné využít uloženou konfiguraci dle vlastní potřeby v jiné aplikaci. Json obsahuje veškeré informace o konfiguraci automatu kromě vstupního slova to není zahrnuto, aby si po načtení bylo možné vyzkoušet automat na libovolném slově.



Obrázek 23 Rozbalovací nabídka soubor

```
{
  "transitions": [
    {
      "state": "q0",
      "new_state": "q1",
      "symbol": "a"
    },
    {
      "state": "q1",
      "new_state": "q0",
      "symbol": "b"
    },
    {
      "state": "q2",
      "new_state": "q3",
      "symbol": ""
    }
  ],
  "states": [
    {
      "state": "q0",
      "accepting": false
    },
    {
      "state": "q1",
      "accepting": true
    },
    {
      "state": "q2",
      "accepting": true
    },
    {
      "state": "q3",
      "accepting": false
    }
  ],
  "alphabet": [
    "a",
    "b"
  ],
  "initialState": "q2",
  "RightOnewayJumping": true,
  "deterministic": false,
  "epsilonTransitions": true
}
```

Obrázek 24 Ukázka struktury souboru json

ZÁVĚR

Tato práce se zabývala právě jednosměrnými konečnými skákajícími automaty a jejich deterministickou a nedeterministickou variantou.

V teoretické části bylo popsáno fungování automatů standardních i právě jednosměrně skákajících. Také byla popsána jejich formální definice, možnosti přechodů, skoků a rozdílné možnosti implementace ε -přechodů, jejich vztah k jazykům Chomského hierarchie. Bylo popsáno, že PJSKA mají větší výpočetní sílu než klasické konečné automaty, avšak jsou nesrovnatelné s bezkontextovými jazyky.

Praktická část práce se zaměřila na návrh a tvorbu desktopové aplikace, která umožňuje vizualizaci a krokování automatů klasických, právě jednosměrně skákajících, deterministických nebo nedeterministických s nebo bez ε -přechodů. Aplikace byla vytvořena v jazyce Python s grafickou částí ve frameworku PyQt6. Funkcionalitu a použití aplikace popisuje návod k použití, který byl vytvořen jako součást této práce.

V budoucnu je možné, díky rozdělení aplikace na GUI a knihovnu FiniteAutomaton, rozšířit aplikaci o další typy automatů nebo o možnost si u NPJSKA vybrat, která z implementací ε -přechodů a skoků bude využita. Také je možné zvážit možnost rozšířit možnosti exportu dat z aplikace nejen na konfiguraci automatu, ale také o výstup procházení.

SEZNAM POUŽITÉ LITERATURY

- [1] ČERNÁ, Ivana, Mojmir KŘETÍNSKÝ a Antonín KUČERA. *Automaty a formální jazyky I*. Brno, 2002. Dostupné také z: <http://is.muni.cz/elportal/estud/fi/js06/ib005/>. Učební text FI MU. Fakulta informatiky, Masarykova univerzita. [cit 2024-10-25].
- [2] MARTINEK, Pavel. *Základy teoretické informatiky. Učební text PřF UP*. Olomouc: Univerzita Palackého, 2005.
- [3] SIPSER, Michael. *Introduction to the theory of computation*. 2nd ed. Boston: Course Technology, c2006. ISBN 978-0-534-95097-2.
- [4] BEIER, Simon a Markus HOLZER. Nondeterministic right one-way jumping finite automata. *Information and Computation*. 2022, (284). ISSN 0890-5401.
- [5] BEIER, Simon a Markus HOLZER. Properties of right one-way jumping finite automata. *Theoretical Computer Science* [online]. 2019, **2019**(798), 78-94 [cit. 2025-04-15]. ISSN 0304-3975,. Dostupné z: doi:j.tcs.2019.03.044.
- [6] CHIGAHARA, Hiroyuki, Szilárd Zsolt FAZEKAS a Akihiro YAMAMURA. One-Way Jumping Finite Automata Send mail to Chigahara H.; *International Journal of Foundations of Computer Science*. 2016, **27**(3), 391 - 405. ISSN 01290541.
- [7] PYTHON SOFTWARE FOUNDATION. *Python 3.13.0 documentation* [online]. 2024 [cit. 2024-10-25]. Dostupné z: docs.python.org
- [8] QT GROUP. *Doc.qt.io* [online]. 1995, 14.08.2024 [cit. 2024-10-25]. Dostupné z: doc.qt.io

SEZNAM OBRÁZKŮ

Obrázek 1 Zobrazení přechodu	14
Obrázek 2 Automat pro detekci, zda obsahuje slovo lichý počet písmen a.....	15
Obrázek 3 Automat pro ukázkou skoku.....	17
Obrázek 4 NPJSKA s různou implementací ε -přechodů.....	19
Obrázek 5 Porovnání tmavého a světlého módu aplikace	23
Obrázek 6 Nastavení aplikace.....	24
Obrázek 7 Metoda init.....	26
Obrázek 8 Kód metody pro načítání slov.....	27
Obrázek 9 Kód metody run.....	28
Obrázek 10 Kód metody step.....	29
Obrázek 11 Kód metody step_non_deterministic	30
Obrázek 12 Ukázka struktury strom (pro přehlednost jsou některé větve skryty).....	32
Obrázek 13 Kód získání prvků.....	33
Obrázek 14 Kód získání symbolu	33
Obrázek 15 Kód metody pro nastavení symbolu jako použitého	34
Obrázek 16 Kód pro zapnutí aplikace.....	35
Obrázek 17 Kód inicializace	36
Obrázek 18 Kód metody paint	37
Obrázek 19 Aplikace s rozbalenou nabídkou nastavení	39
Obrázek 20 Ukázka zadání přechodové funkce	40
Obrázek 21 Vizualizační část aplikace	42
Obrázek 22 Ovládací prvek pro výběr větve	42
Obrázek 24 Rozbalovací nabídka soubor.....	43
Obrázek 23 Ukázka struktury souboru json.....	44

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

PJSKA	pravě jednosměrný skákající konečný automat
NPJSKA	nedeterministický pravě jednosměrný skákající konečný automat
GUI	grafické uživatelské rozhraní

SEZNAM PŘÍLOH

Příloha P I: Aplikace Vizualizátor konečných automatů

PŘÍLOHA P I: APLIKACE VIZUALIZÁTOR KONEČNÝCH AUTOMATŮ

Soubor .zip obsahuje aplikaci pro Vizualizace právě jednosměrných skákajících konečných automatů.