VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

REGULOVANÉ JAZYKOVÉ OPERACE A JEJICH UŽITÍ

REGULATED LANGUAGE OPERATIONS AND THEIR USE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DAVID CHOCHOLATÝ**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

141140

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Chocholatý David**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Regulované jazykové operace a jejich užití**
Kategorie:	Teoretická informatika
Akademický rok:	2022/23

Zadání:

1. Seznamte se s novými jazykovými operacemi, které zavedla moderní teoretická informatika.
2. Dle pokynů vedoucího zaveďte a studujte nové jazykové operace se zaměřením na regulované změny řetězců a jazyků. Formalizujte tyto operace pomocí vhodných matematických modelů.
3. Obecně definujte modely z předchozího bodu.
4. Studujte využití těchto modelů pro definované operace. Zaměřte se na využití v mikrobiologii a textových editorech. Implementujte vhodný model a ověřte jeho činnost na sadě minimálně 10 příkladů.
5. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Maxime Crochemore and Wojciech Rytter: Text algorithms. New York ; Oxford : Oxford University Press, 1994
- Lila Kari: On insertion and deletion in formal languages. PhD Thesis, University of Turku, Finland, 1991

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3.

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

Vedoucí práce:	**Meduna Alexandr, prof. RNDr., CSc.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	28.10.2022

Abstrakt

Tato práce představuje a studuje *vymazávací systémy* jako alternativní formální jazykový model k *obecným skákajícím konečným automatům*. Významným rozdílem oproti daným automatům je využití řídícího regulárního jazyka namísto stavového řízení v podobě obecného konečného automatu. Vymazávací systémy ponechávají práci s řetězci na vstupní pásce, přičemž samotné regulární jazyky mohou být přijímány klasickými konečnými automaty. Zároveň se zavedením nového formálního systému práce prokazuje jeho vztahy se známými jazykovými rodinami, rodinou jazyků zamíchání, Dyckovými jazyky a uzávěrové vlastnosti. Na základě formální specifikace vymazávacího systému je uvedeno více aplikací v oblasti bioinformatiky pro molekulární biologii, textových editorů a kompozičního šachu, včetně návrhu algoritmů a prezentování implementačního řešení.

Abstract

This thesis introduces and studies *erasing systems* as an alternative formal language model to *general jumping finite automata*. A significant difference compared to the given automata is the use of a control regular language instead of state control in the form of a general finite automaton. Erasing systems leave the string work on the input tape, whereas regular languages themselves can be accepted by classical finite automata. At the same time, with the introduction of a new formal system, the thesis demonstrates its relations with well-known language families, the family of shuffle languages, Dyck languages and closure properties. Based on the formal specification of the erasing system, multiple applications in bioinformatics for molecular biology, text editors and compositional chess are shown, including designing algorithms and presenting the implementation solution.

Klíčová slova

vymazávací systém, skákající konečné automaty, regulované gramatiky a automaty, operace vymazání, srovnání s jinými formálními zařízeními definujícími jazyk, uzávěrové vlastnosti, aplikace formálních modelů, otevřené problémy, nové oblasti výzkumu

Keywords

erasing system, jumping finite automata, regulated grammars and automata, erasing operation, comparison with other language-defining formal devices, closure properties, application of formal models, open problems, new areas of research

Citace

CHOCHOLATÝ, David. *Regulované jazykové operace a jejich užití*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Alexander Meduna, CSc.

Regulované jazykové operace a jejich užití

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. RNDr. Alexandera Meduny, CSc. Další informace mi poskytla paní Ing. Ivana Burgetová, Ph.D. z oblasti molekulární biologie a pan Ing. Zbyněk Křivka, Ph.D. o aktuálním výzkumu formálních modelů, který také nadnesl několik otevřených problémů, jenž jsou explicitně označeny. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
David Chocholatý
3. května 2023

Poděkování

Tímto bych rád poděkoval panu prof. RNDr. Alexanderu Medunovi, CSc. za odborné vedení, vstřícný přístup, inspirativní konzultace a za jeho ochotu v průběhu vypracovávání této práce. Dále bych rád poděkoval paní Ing. Ivaně Burgetové, Ph.D. za poskytnuté informace z oblasti bioinformatiky a molekulární biologie, rady při aplikaci formálních systémů a za doporučení odborné literatury. Mé poděkování patří také panu Ing. Zbyňku Křivkovi, Ph.D. za jeho ochotu, cenné přispění otevřenými problémy a za poskytnuté materiály z oblasti aktuálního výzkumu formálních modelů.

Obsah

Kapitola 1

Úvod

Informační technologie se staly neodmyslitelnou součástí každodenního života. Se stále se rozšiřujícími znalostmi a dovednostmi lidstva nacházejí počítače a obecně veškeré výpočetní stroje své další uplatnění. Nové technologie, algoritmy a modely čím dál více nahrazují mnohdy monotónní lidskou práci a vytvářejí prostor pro kreativní činnosti. Samotné počítače ovšem v moderním použití pro provádění úloh byly nejprve popsány v teoretickém smyslu jako *Turingův stroj*. Tento matematický model výpočtu tvoří důležitou součást oblasti *teorie automatů* a obecně *teorie formálních jazyků*, která má velký vliv na samotné odvětví informačních technologií.

Teorie formálních jazyků studuje způsoby vyjádření jazyků matematicky spolu s jejich vlastnostmi. Nejběžnější formální jazykové modely používané k definici jazyků jsou *gramatiky* a *automaty*. Gramatiky slouží jako zařízení generující jazyk. Naopak automaty se využívají jako zařízení pro jeho rozpoznávání. Automat si lze jednoduše představit jako stroj, který čte vstupní řetězec symbol po symbolu zleva doprava a odpovídajícím způsobem mění svůj stav, přičemž jako řetězec se uvažuje sekvence symbolů. Pokud se automat dostane do koncového stavu zároveň s tím, že byl přečten celý vstupní řetězec, daný řetězec patří do jazyka, který příslušný automat rozpoznává.

V průběhu výzkumu v oblasti teorie automatů, jenž se věnuje jejich studiu, vznikly různé modifikace a rozšíření klasických konečných automatů, které obsahují konečný počet stavů. Jednu z těchto modifikací představují také *skákající konečné automaty*. Ty pracují jako klasické konečné automaty s významnou výjimkou, že nečtou vstupní řetězec zleva doprava, ovšem po přečtení symbolu a změně stavu může čtecí hlava přeskočit na libovolnou pozici ve zbývajícím vstupu. Skákající konečné automaty podléhají důkladnému studování a jedná se stále o aktuální téma výzkumu nejen v oblasti teorie automatů, ale i jejich aplikace na reálné problémy.

Samotné skákající konečné automaty řídí přijímání řetězce pomocí změny stavu, přičemž stav automatu je měněn na základě vstupu. Ovšem v jisté souvislosti k *regulovaným automatům*, jenž při přijímání řetězce mohou být řízeny současně stavem a také jazykem, lze přirozeně uvažovat nový formální systém, který by namísto stavů využíval pouze řídící jazyk.

Tato práce si klade za cíl zavedení nového formálního systému, nazývaného *vymazávací systém*, v oblasti formální teorie jazyků. Zároveň odpovídá na otázky typu, jaké jsou přijímací schopnosti a vlastnosti tohoto systému. Významnou úlohou je zkoumání jeho uplatnění na řešení reálných problémů v oblasti bioinformatiky pro molekulární biologii, v textových editorech či kompozičním šachu. Neméně důležitý účel této práce také zastupuje návrh al-

goritmu samotného vymazávacího systému a jeho aplikací, včetně vytvoření programového řešení demonstrujícího jejich funkcionalitu.

Celkově práce rozděluje její jednotlivé úlohy do osmi kapitol. Po úvodu uvedeném v této kapitole, kapitola 2 shrnuje základy teorie formálních jazyků, včetně nezbytných matematických pojmů a notace použité v této práci, za účelem eliminování případných nejasností. Následně kapitola 3 navazuje úvodem do teorie automatů a představuje fundamentální formální modely stěžejní pro pochopení navazujícího obsahu. Poté kapitola 4 představuje definice, příklady a samotné vztahy a vlastnosti skákajících konečných automatů. Kapitola 5 formálně zavádí vymazávací systémy do teorie formálních jazyků, včetně ukázkových příkladů, prokázání jejich přijímacích schopností, vymezení síly vůči některým jazykovým rodinám a uzávěrových vlastností. Na tuto kapitolu navazuje kapitola 6, která studuje a demonstruje jednotlivá užití nového formálního systému na reálné problémy a následně kapitola 7 vyobrazuje návrh algoritmů a popisuje programové řešení vymazávacího systému a jeho aplikací. Závěrečná kapitola, kapitola 8, shrnuje dosavadní prokázané vztahy a vlastnosti vymazávacích systémů, diskutuje o jejich užití a navrhuje nové otevřené problémy s nimi souvisejícími. S tím zároveň přednáší nové oblasti výzkumu v teorii formálních jazyků, a to především na vývoj nových formálních modelů inspirovaných molekulární biologií.

Kapitola 2

Základy teorie formálních jazyků

Cílem této kapitoly je představit základní terminologii týkající se teorie formálních jazyků a vybrané podstatné části matematické teorie včetně použité notace. Všechny související definice jsou uvedeny v rozsahu potřebném pro tuto práci a lze je označit za klíčové pro plné pochopení celého textu. Čtenáři, který je již obeznámen s teorií formálních jazyků, lze doporučit, aby se ke kapitole 2 navracel v případě potřeby.

Tato kapitola je rozdělena do osmi sekcí, jenž každá zastupuje jednotný celek, ať již z oblasti matematické teorie, nebo teorie formálních jazyků. Sekce 2.1 prezentuje vybrané specifické partie z teorie množin. Sekce 2.2 poskytuje přehled definic základních prvků teorie formálního jazyka, především se zaměřením na řetězce, jazyky a operace s nimi. Po upevnění použité základní notace pro minimalizaci nebezpečí možné záměny sekce 2.3 představuje po řadě *Chomského gramatiky* a jejich rozdělení do tříd dle *Chomského hierarchie*. Následně sekce 2.4 představuje specifické partie oblasti formální teorie, které lze označit fundamentálními pro odlišné části této práce.

2.1 Vybrané partie matematické teorie

Tato sekce definuje vybrané pojmy z matematické teorie a předpokládá alespoň základní znalosti teorie množin. Pro další informace je čtenář odkázán například na souhrnné přehledy v [19, 20]. Pro více informací lze poté odkázat již na jednotlivé publikace [8, 12, 14, 18].

Definice 2.1.1. Nechť S je množina. Pro množinu S, $card(S)$ značí kardinalitu této množiny (počet prvků v S) [20].

Definice 2.1.2. Nechť Q_1 a Q_2 jsou dvě množiny. Pak *funkce* ψ z Q_1 do Q_2 je relace z Q_1 do Q_2, přičemž pro každé $x \in Q_1$,

$$card(\{y \mid y \in Q_2 \text{ a } (x, y) \in \psi\}) \leq 1$$

dle [20].

Definice 2.1.3. Nechť Q_1 a Q_2 jsou dvě množiny. Nechť ψ je funkce z Q_1 do Q_2. Pak *rozsah* (anglicky *range*) ψ, značen $range(\psi)$, je definován jako

$$range(\psi) = \{y \mid y \in Q_2 \text{ a } (x, y) \in \psi \text{ pro nějaké } x \in Q_1\}$$

dle [20].

2.2 Abeceda, řetězec a jazyk

Cílem této sekce je představení základních pojmů formální teorie se zaměřením především na řetězce a jazyky. Podsekce 2.2.1 předkládá základní stavební prvky formální teorie, a to abecedy a řetězce. Na definované pojmy navazuje podsekce 2.2.2 prezentováním jednotlivých operací s řetězci, které jsou dále v této práci využity. Pro rozšíření obecných pojmů ve stejné posloupnosti podsekce 2.2.3 a 2.2.4 zavádí po řadě definice formálních jazyků a následně možných operací s formálními jazyky.

2.2.1 Abecedy a řetězce

Definice 2.2.1. *Abeceda* je konečná, neprázdná množina elementů, které jsou nazývány *symboly* [19].

Definice 2.2.2. Nechť Σ je abeceda. Pokud $card(\Sigma) = 1$, pak Σ je *unární abeceda* [20].

Definice 2.2.3. Nechť Σ je abeceda. *Řetězec* nad Σ je jakákoliv konečná sekvence symbolů ze Σ. [20] V řetězci jsou dále vynechávány všechny oddělovací čárky. To znamená, že řetězec $a_1, a_2, ..., a_n$ pro nějaké $n \geq 1$, je zapisován jako $a_1a_2...a_n$. Řetězec neobsahující žádný symbol, značen ε, je nazýván *prázdným řetězcem.* [20]

Definice 2.2.4. Nechť x je řetězec nad abecedou Σ. *Délka* x, značena $|x|$, je definována následovně:

(i) Pokud $x = \varepsilon$, pak $|x| = 0$.

(ii) Pokud $x = a_1a_2...a_n$, kde $a_i \in \Sigma$ pro všechna $i = 1, ..., n$ pro nějaké $n \geq 1$, pak $|x| = n$ [19].

Definice 2.2.5. Nechť x je řetězec nad abecedou Σ. *Počet výskytů daného symbolu* $a \in \Sigma$ v x je značen $|x|_a$ [21].

Definice 2.2.6. Nechť x je řetězec nad abecedou Σ. *Abeceda* x, značena $alph(x)$, je definována následovně:

(i) Pokud $x = \varepsilon$, pak $alph(x) = \emptyset$.

(ii) Pokud $x = a_1a_2...a_n$, kde $a_i \in \Sigma$ pro všechna $i = 1, ..., n$ pro nějaké $n \geq 1$, pak $alph(x) = \{a_1, a_2, ..., a_n\}$ [20].

2.2.2 Operace s řetězci

Definice 2.2.7. Nechť x je řetězec nad abecedou Σ. *Reverzace* řetězce x, značena $rev(x)$, je definována následovně:

(i) Pokud $x = \varepsilon$, pak $rev(x) = \varepsilon$.

(ii) Pokud $x = a_1a_2...a_n$, kde $a_i \in \Sigma$ pro všechna $i = 1, ..., n$ pro nějaké $n \geq 1$, pak $rev(x) = a_na_{n-1}...a_2a_1$ [20].

Definice 2.2.8. Nechť x a y jsou dva řetězce nad abecedou Σ. *Konkatenace* x a y je řetězec xy. Pro konkatenaci řetězce x a prázdného řetězce ε nad abecedou Σ platí

$$x\varepsilon = \varepsilon x = x$$

dle [20].

Definice 2.2.9. Nechť x je řetězec nad abecedou Σ. Pro $n \geq 0$, *n-tá mocnina* řetězce x, značena x^n, je řetězec nad Σ, rekurzivně definován jako

$$x^0 = \varepsilon,$$
$$x^n = xx^{n-1}, n \geq 1$$

dle [19].

Definice 2.2.10. Nechť x a y jsou dva řetězce nad abecedou Σ. Pak x je *prefixem* y, pokud existuje řetězec z nad Σ, přičemž $xz = y$. Pokud $x \notin \{\varepsilon, y\}$, pak x je *vlastním prefixem* y.

Pro řetězec y, $prefix(y)$ značí množinu všech prefixů y, tudíž

$$prefix(y) = \{x \mid x \text{ je prefix } y\}$$

dle [19].

Definice 2.2.11. Nechť x a y jsou dva řetězce nad abecedou Σ. Pak x je *sufixem* y, pokud existuje řetězec z nad abecedou Σ, přičemž $zx = y$. Pokud $x \notin \{\varepsilon, y\}$, pak x je *vlastním sufixem* řetězce y.

Pro řetězec y, $suffix(y)$ značí množinu všech sufixů y, tudíž

$$suffix(y) = \{x \mid x \text{ je sufix } y\}$$

dle [19].

Definice 2.2.12. Nechť x a y jsou dva řetězce nad abecedou Σ. Pak x je *podřetězcem* y, pokud existují dva řetězce z a z' nad abecedou Σ, přičemž $zxz' = y$. Pokud $x \notin \{\varepsilon, y\}$, pak x je *vlastním podřetězcem* řetězce y.

Pro řetězec y, $sub(y)$ značí množinu všech podřetězců y, tudíž

$$sub(y) = \{x \mid x \text{ je podřetězec } y\}.$$

Navíc pro každý řetězec y platí následující tři vlastnosti:

(i) $prefix(y) \subseteq sub(y)$,

(ii) $suffix(y) \subseteq sub(y)$,

(iii) $\{\varepsilon, y\} \subseteq prefix(y) \cap suffix(y) \cap sub(y)$ [19].

Definice 2.2.13. Nechť x a y jsou dva řetězce nad abecedou Σ. *Zamíchání* (anglicky *shuffle*) x a y, značeno $\sqcup\!\sqcup$, je definováno jako

$$x \sqcup\!\sqcup y = \{x_1y_1x_2y_2...x_ny_n \mid u = x_1x_2...x_n, v = y_1y_2...y_n, x_i, y_i \in \Sigma^*, 1 \leq i \leq n, n \geq 1\}$$

dle [11].

2.2.3 Formální jazyky

Definice 2.2.14. Nechť Σ je abeceda. Nechť Σ^* značí množinu všech řetězců nad Σ. Množina

$$\Sigma^+ = \Sigma^* - \{\varepsilon\}$$

značí množinu všech neprázdných řetězců nad Σ [19].

Definice 2.2.15. Nechť Σ je abeceda. *Jazyk* L nad abecedou Σ je každá podmnožina Σ^*, $L \subseteq \Sigma^*$. Množina Σ^* je nazývána *univerzálním jazykem*, protože se skládá ze všech řetězců nad Σ [20].

Definice 2.2.16. Nechť L je jazyk nad abecedou Σ. Pokud L je konečná množina, pak se jedná o *konečný jazyk*. V opačném případě jazyk L je nazýván *jazykem nekonečným*. Množina všech konečných jazyků nad Σ je značena $fin(\Sigma)$. Pokud $card(\Sigma) = 1$, pak L je *unární jazyk*. *Prázdný jazyk* je značen $\emptyset$ [20].

Definice 2.2.17. Nechť L je jazyk nad abecedou Σ. Pro $L \in fin(\Sigma)$, $max(L)$ značí délku nejdelšího řetězce v L. Dále je stanoveno $max(\emptyset) = 0$ [20].

Definice 2.2.18. Nechť L je jazyk nad abecedou Σ. *Abeceda jazyka* L, značena $alph(L)$, je definována jako

$$alph(L) = \bigcup_{x \in L} alph(x)$$

dle [20].

2.2.4 Operace s formálními jazyky

Definice 2.2.19. Nechť L je jazyk nad abecedou Σ. *Reverzace jazyka* L, značena $rev(L)$, je definována jako

$$rev(L) = \{rev(x) \mid x \in L\}$$

dle [20].

Definice 2.2.20. Nechť L_1 a L_2 jsou dva jazyky nad abecedou Σ. Jelikož jsou jazyky množiny, lze definovat množinové operace jako *sjednocení*, *průnik*, *rozdíl* a *doplněk*:

$$\begin{aligned}
&L_1 \cup L_2 = \{x \mid x \in L_1 \text{ nebo } x \in L_2\},\\
&L_1 \cap L_1 = \{x \mid x \in L_1 \text{ a } x \in L_2\},\\
&L_1 - L_2 = \{x \mid x \in L_1 \text{ a } x \notin L_2\},\\
&\overline{L} = \Sigma^* - L,
\end{aligned}$$

kde L je jazyk nad abecedou Σ [20].

Definice 2.2.21. Nechť L_1 a L_2 jsou dva jazyky nad abecedou Σ. L_1 a L_2 jsou *ekvivalentní*, zapsáno $L_1 = L_2$, pokud $L_1 \cup \{\varepsilon\}$ a $L_2 \cup \{\varepsilon\}$ jsou identické. Pro jazyky L_1 a L_2 platí, že $L_1 \subseteq L_2$ pokud jazyk $L_1 \cup \{\varepsilon\}$ je podmnožinou jazyka $L_2 \cup \{\varepsilon\}$ [20].

Definice 2.2.22. Nechť L_1 a L_2 jsou dva jazyky nad abecedou Σ. *Konkatenace* L_1 a L_2, značena L_1L_2, je definována jako

$$L_1L_2 = \{xy \,|\, x \in L_1 \; a \; y \in L_2\}$$

dle [19]. Konkatenace jazyků je asociativní, protože konkatenace řetězců je asociativní [16].

Definice 2.2.23. Nechť L je jazyk nad abecedou Σ. Jelikož konkatenace jazyků je asociativní, lze definovat jazyk L^n pro $n \geq 1$, který je jazykem získaným konkatenací n kopií L, nazývaný jako *n-tá mocnina jazyka* L, rekurzivně definována jako

$$L^0 = \{\varepsilon\},$$
$$L^n = LL^{n-1}, n \geq 1$$

dle [16].

Definice 2.2.24. Nechť L je jazyk nad abecedou Σ. Jazyk L^0 je jazyk $\{\varepsilon\}$ obsahující pouze prázdný řetězec ε. Pro jakýkoliv jazyk L,

$$L\emptyset = \emptyset L = \emptyset,$$
$$L\{\varepsilon\} = \{\varepsilon\}L = L$$

dle [16].

Definice 2.2.25. Nechť L je jazyk nad abecedou Σ. *Iterace* jazyka L, značena L^*, kde $*$ je *Kleeneho hvězdička* [21], je definována jako sjednocení všech mocnin jazyka L,

$$L^* = \bigcup_{i=0}^{\infty} L^i,$$

dle [16].

Definice 2.2.26. Nechť L je jazyk nad abecedou Σ. *Pozitivní iterace* jazyka L, značena L^+, kde $+$ je *Kleeneho plus* [21], je definována jako sjednocení všech kladných mocnin jazyka L,

$$L^+ = \bigcup_{i=1}^{\infty} L^i.$$

Z toho vyplývá, že L je ekvivaletní L^* nebo $L^* - \{\varepsilon\}$ v závislosti na tom, zda $\varepsilon \in L$ nebo $\varepsilon \notin L$ [16].

Definice 2.2.27. Nechť L_1 a L_2 jsou dva jazyky nad abecedou Σ. *Levý kvocient* L_1 podle L_2 je definován jako

$$L_2 \backslash L_1 = \{y \,|\, xy \in L_1 \text{ pro nějaké } x \in L_2\}$$

dle [16].

Definice 2.2.28. Nechť L_1 a L_2 jsou dva jazyky nad abecedou Σ. *Pravý kvocient* L_1 podle L_2 je definován jako

$$L_1/L_2 = \{y \mid yx \in L_1 \text{ pro nějaké } x \in L_2\}$$

dle [16].

Definice 2.2.29. Nechť L_1 a L_2 jsou dva jazyky nad abecedou Σ. Operace *zamíchání* (anglicky *shuffle*) je definována jako

$$L_1 \sqcup\!\sqcup L_2 = \bigcup_{\substack{u \in L_1 \\ v \in L_2}} (u \sqcup\!\sqcup v)$$

pro $u, v \in \Sigma^*$ a $L_1, L_2 \subseteq \Sigma^*$ [11].

Definice 2.2.30. Pro $L \subseteq \Sigma^*$, *opakované zamíchání* (anglicky *iterated shuffle*) L je definované jako

$$L^{\sqcup\!\sqcup,*} = \bigcup_{n=0}^{\infty} L^{\sqcup\!\sqcup,n},$$

kde $L^{\sqcup\!\sqcup,0} = \{\varepsilon\}$ a $L^{\sqcup\!\sqcup,i} = L^{\sqcup\!\sqcup,i-1} \sqcup\!\sqcup L$ pro $i \geq 1$ [11].

Definice 2.2.31. Nechť Σ a Γ jsou dvě abecedy. Totální funkce σ ze Σ^* na 2^{Γ^*}, přičemž $\sigma(xy) = \sigma(x)\sigma(y)$ pro každé $x, y \in \Sigma^*$, je *substituce* [20]. Pokud σ je definována ze Σ^* na 2^{Γ^+}, tedy $\varepsilon \notin \sigma(a)$ pro každé $a \in \Sigma$ [21], pak je substituce nazývána *substitucí bez* ε [20]. Pokud $card(\sigma(a))$ je konečná pro každé $a \in \Sigma$, pak substituce je nazývána *konečnou substitucí* [21]. Dle definice $\sigma(\varepsilon) = \{\varepsilon\}$ a $\sigma(a_1a_2...a_n) = \sigma(a_1)\sigma(a_2)...\sigma(a_n)$, kde $n \geq 1$ a $a_i \in \Sigma$ pro všechna $i = 1, ..., n$. Tudíž σ je plně specifikována definováním $\sigma(a)$ pro každé $a \in \Sigma$. Pro $L \subseteq \Sigma^*$, je definice σ rozšířena na

$$\sigma(L) = \bigcup_{x \in L} \sigma(x)$$

dle [20].

Definice 2.2.32. Nechť Σ a Γ jsou dvě abecedy. Totální funkce φ ze Σ^* na Γ^*, přičemž $\varphi(xy) = \varphi(x)\varphi(y)$ pro každé $x, y \in \Sigma^*$, je *homomorfismus*, nebo synonymně *morfismus*. Jelikož každý homomorfismus je speciální případ konečné substituce, je φ specifikováno analogicky dle specifikace σ [20]. Tudíž φ je *homomorfismem bez* ε, pokud $\varepsilon \notin \varphi(a)$ pro každé $a \in \Sigma$ [21]. Pro jazyk $L \subseteq \Sigma^*$, je definováno

$$\varphi(L) = \{\varphi(x) \mid x \in L\}$$

dle [20].

Definice 2.2.33. Nechť φ je homomorfismus. Pak φ^{-1} značí *inverzní homomorfismus*, definovaný jako

$$\varphi^{-1}(w) = \{x \in \Sigma^* \mid \varphi(x) = w\}$$

dle [21].

2.3 Chomského gramatiky a hierarchie

Cílem této sekce je představit základní zařízení pro generování jazyků využívané v oblasti formální teorie, a to gramatiky. Následně popisuje jejich rozdělení do hierarchických tříd.

Jako první podsekce 2.3.1 představuje jednotlivé typy *Chomského gramatik* na základě jejich generativní síly a podsekce 2.3.2 rozřazuje definované formální gramatiky do tříd podle *Chomského hierarchie*.

2.3.1 Chomského gramatiky

Tato podsekce definuje zařízení, která v teorii formálních jazyků hrají důležitou roli jako jazykové modely pro generování jazyků, obecně nazývané *gramatiky*. Dále se definice zaměřují pouze na gramatiky známé jako *Chomského gramatiky*, přičemž postupně jsou představeny jejich jednotlivé základní typy se snižující se generativní silou.

Definice 2.3.1. *Frázová gramatika* (anglicky *phrase-structure grammar*), zkráceně *PSG*, je čtveřice

$$G = (N, T, P, S),$$

kde

- N je abeceda *neterminálů*,
- T je abeceda *terminálů*, přičemž $N \cap T = \emptyset$,
- P je konečná množina *pravidel* tvaru $x \to y$, kde $x \in (N \cup T)^* N (N \cup T)^*$, $y \in (N \cup T)^*$,
- $S \in N$ je *počáteční symbol* [24].

Definice 2.3.2. Nechť $G = (N, T, P, S)$ je PSG a $V_G = N \cup T$. Pro $x, y \in V_G^*$, *relace přímé derivace*, značena $\Rightarrow_G$, je definována jako

$$x \Rightarrow_G y$$

tehdy a jen tehdy, když $x = x_1 u x_2$, $y = x_1 v x_2$ pro nějaké $x_1, x_2 \in V^*$ a $u \to v \in P$. Pokud G je pochopeno, je zapisováno $x \Rightarrow y$, přičemž je řečeno, že x *přímo derivuje* y (vzhledem ke G) [24]. Jelikož $\Rightarrow$ je relace, $\Rightarrow^n$ je n-tá mocnina $\Rightarrow$ pro $n \geq 0$, $\Rightarrow^+$ tranzitivní uzávěr $\Rightarrow$ a $\Rightarrow^*$ reflexivně-tranzitivní uzávěr $\Rightarrow$ [20].

Jazyk generovaný G, značený $L(G)$, je definován jako

$$L(G) = \{x \in T^* \mid S \Rightarrow^* x\}.$$

Dvě gramatiky G_1 a G_2 jsou *ekvivalentní*, pokud $L(G_1) = L(G_2)$ [24].

Definice 2.3.3. *Kontextová gramatika* (anglicky *context-sensitive grammar*), zkráceně *CSG*, je PSG

$$G = (N, T, P, S),$$

přičemž každé $u \to v \in P$ je tvaru

$$u = u_1 A u_2, v = u_1 x u_2$$

pro $u_1, u_2 \in V_G^*$, $A \in N$ a $x \in V_G^+$, kde $V_G = N \cup T$ [20].

Definice 2.3.4. *Bezkontextová gramatika* (anglicky *context-free grammar*), zkráceně *CFG*, je PSG

$$G = (N, T, P, S),$$

přičemž každé pravidlo v P je tvaru

$$A \to x$$

pro $A \in N$ a $x \in V_G^*$, kde $V_G = N \cup T$ [20].

Definice 2.3.5. *Lineární gramatika* (anglicky *linear grammar*), zkráceně *LG*, je PSG

$$G = (N, T, P, S),$$

přičemž každé pravidlo v P je tvaru

$$A \to xBy \text{ nebo } A \to x,$$

kde $A, B \in N$ a $x, y \in T^*$ [20].

Definice 2.3.6. *Regulární gramatika* (anglicky *regular grammar*), zkráceně *RG*, je PSG

$$G = (N, T, P, S),$$

přičemž každé pravidlo v P je tvaru

$$A \to aB \textit{ nebo } A \to a \textit{ nebo } A \to \varepsilon,$$

kde $A, B \in N$ a $a \in T$ [24].

2.3.2 Chomského hierarchie

Cílem této podkapitoly je prezentování klasifikace jednotlivých gramatik do tříd, které tvoří prvky celkové struktury označované jako *Chomského hierarchie*.

Chomského hierarchie je hierarchie tříd formálních gramatik generujících formální jazyky. Hierarchie řadí jednotlivé typy gramatik na základě jejich obecnosti a následně jejich generativní síly, přičemž platí, že čím vyšší je číslo typu gramatiky, tím je menší generativní síla.

Frázová gramatika, kontextová gramatika, bezkontextová gramatika a regulární gramatika jsou také příslušně nazývány gramatikami typu 0, typu 1, typu 2 a typu 3.

RE, **CS**, **CF**, **LIN** a **REG** značí rodiny jazyků generované příslušnými libovolnými kontextovými, bezkontextovými, lineárními a regulárními gramatikami (**RE** značí rodinu rekurzivně spočetných jazyků) [24].

Platí následující striktní inkluze:

$$\mathbf{REG} \subset \mathbf{LIN} \subset \mathbf{CF} \subset \mathbf{CS} \subset \mathbf{RE},$$

přičemž se jedná o Chomského hierarchii [24] (viz [4, 5]). Uzávěrové vlastnosti vyjmenovaných jazykových rodin jsou uvedeny v příloze B.

Dle [20] je také uvažována rodina konečných jazyků, značena **FIN**, přičemž

$$\mathbf{FIN} \subset \mathbf{REG}.$$

Pro důkazy v této práci se dále neuvažuje rodina jazyků, která je generována lineárními gramatikami (**LIN**).

2.4 Specifické oblasti formální teorie jazyků

Po představení základních a stěžejních pojmů z oblasti formální teorie jazyků přidává následující sekce k zavedeným definicím ještě určité specifické formalismy, které jsou využívány v této práci.

Celkem se jedná o čtyři oblasti. Nejprve podsekce 2.4.1 představuje bezkontextové jazyky známé jako *Dyckovy* a *polo-Dyckovy jazyky*. Poté podsekce 2.4.2 zavádí jazykové operace se zaměřením na vkládání jako inverzní operaci k vymazávání. Nakonec podsekce 2.4.3 a 2.4.4 postupně prezentují dva typy výrazů, a to *regulární výrazy* a *výrazy zamíchání*.

2.4.1 Dyckovy jazyky

Tato podsekce představuje *Dyckovy* a *polo-Dyckovy jazyky*, které jsou důležitými podrodinami rodiny bezkontextových jazyků.

V oblasti formálních jazyků jsou Dyckovy jazyky bezkontextové jazyky, které obsahují řetězce s vyváženým počtem správně vnořených závorek. V této sekci jsou nejprve představeny samotné Dyckovy jazyky a následně takzvané polo-Dyckovy jazyky, které obsahují řetězce se stejným počtem odpovídajících symbolů (může se jednat opět o závorky), ovšem na rozdíl od Dyckových jazyků nezajišťují korektní uzávorkování.

Definice 2.4.1. *Dyckův jazyk* (anglicky *Dyck language*), značen $\mathcal{D}_n$, nad

$$T_n = \{a_1, a'_1, a_2, a'_2, ..., a_n, a'_n\},$$

kde $n \geq 1$, je bezkontextový jazyk generovaný gramatikou

$$G = (\{S\}, T_n, S, \{S \to \varepsilon, S \to SS\} \cup \{S \to a_i S a'_i \mid 1 \leq i \leq n\}, S).$$

Dvojice (a_i, a'_i), kde $1 \leq i \leq n$, mohou být uvažovány jako závorky, levá a pravá, různých druhů. Pak $\mathcal{D}_n$ se skládá ze všech řetězců správně vnořených závorek [21].

Definice 2.4.2. *Polo-Dyckův jazyk* (anglicky *semi-Dyck language*), značen $\mathcal{S}_n$, nad

$$T_n = \{a_1, a'_1, a_2, a'_2, ..., a_n, a'_n\},$$

kde $n \geq 1$, je bezkontextový jazyk generovaný gramatikou

$$G = (\{S\}, T_n, S, \{S \to \varepsilon, S \to SS\} \cup \{S \to a_i S a'_i, S \to a'_i S a_i \mid 1 \leq i \leq n\}, S).$$

Polo-Dyckovy jazyky generují všechny řetězce stejného počtu odpovídajících symbolů [2].

Nechť $\boldsymbol{\mathcal{D}}$ a $\boldsymbol{\mathcal{S}}$ označují rodiny bezkontextových jazyků generované příslušnými libovolnými gramatikami pro Dyckovy jazyky a polo-Dyckovy jazyky.

2.4.2 Vložení a kompozice

Hlavním cílem této podkapitoly je definování specifických jazykových operací a samotných typů jazyků s důrazem na operaci vkládání, které jsou využity v této práci pro některé vybrané důkazy. Následující definice představuje operaci *vložení*, kterou využívá takzvaná *kompozice*.

Definice 2.4.3. Nechť $K, L \subseteq \Sigma^*$ jsou jazyky. *Vložení* K do L je definováno jako

$$L \leftarrow K = \{u_1 v u_2 \mid u_1 u_2 \in L, v \in K\}.$$

Obecněji pro každé $k \geq 1$ označujeme

$$L \leftarrow^k K = (L \leftarrow^{k-1} K) \leftarrow K,$$
$$L \leftarrow^* K = \bigcup_{i \geq 0} L \leftarrow^i K,$$

kde $L \leftarrow^0 K$ znamená L. Ve výrazech s $\leftarrow$ a $\leftarrow^*$, jednoprvková množina $\{w\}$ může být nahrazena w. Sekvence $L_1 \leftarrow L_2 \leftarrow ... \leftarrow L_d$ vložení je vyhodnocena zleva. Například $L_1 \leftarrow L_2 \leftarrow L_3$ znamená $(L_1 \leftarrow L_2) \leftarrow L_3$. V souladu s [9], $L \subseteq \Sigma^*$ je *jednotkový jazyk* (anglicky *unitary language*), pokud $L = w \leftarrow^* K$ pro $w \in \Sigma^*$ a konečný jazyk $K \subseteq \Sigma^*$ [27].

Definice 2.4.4. Jazyk $K \subseteq \Sigma^*$ je *kompozice*, pokud $K = \{\varepsilon\}$ nebo

$$K = \varepsilon \leftarrow u_d \leftarrow u_{d-1} \leftarrow ... \leftarrow u_2 \leftarrow u_1$$

pro nějaké $u_1, u_2, ..., u_d \in \Sigma^*, d \geq 1$. Kompozice K je *stupně* $n \geq 0$, pokud $K = \{\varepsilon\}$ nebo $|u_i| \leq n$ pro každé $i \in \{1, ..., d\}$. Pro každé $n \geq 0$, nechť $\mathbf{UC}_n$ označuje třídu jazyků L, které mohou být zapsány jako

$$L = \bigcup_{K \in \mathcal{C}} K,$$

kde $\mathcal{C}$ je jakákoliv (může být i nekonečná) množina kompozic stupně n. Také je značeno $\mathbf{UC} = \bigcup_{n \geq 0} \mathbf{UC}_n$. Zkratka $\mathbf{UC}$ znamená *sjednocení kompozic* (anglicky *union of compositions*) [27].

2.4.3 Regulární výrazy

Za účelem stručného a srozumitelného značení regulárních jazyků v sekvenční formě se používají takzvané *regulární výrazy*. Obecně vzato se jedná o výrazy s operátory „·“, „+“ a „∗“, které značí v tomto pořadí konkatenaci, sjednocení a iteraci.

Definice 2.4.5. Nechť Σ je abeceda. *Regulární výrazy* nad Σ a jazyky, které tyto výrazy značí, jsou rekurzivně definovány následovně:

(i) $\emptyset$ je regulární výraz značící prázdnou množinu.

(ii) ε je regulární výraz značící $\{\varepsilon\}$.

(iii) a, kde $a \in \Sigma$, je regulární výraz značící $\{a\}$.

(iv) Pokud r a s jsou regulární výrazy značící po řadě jazyky L_r a L_s, potom:

  (a) $(r \cdot s)$ je regulární výraz značící jazyk $L = L_r L_s$.
  (b) $(r + s)$ je regulární výraz značící jazyk $L = L_r \cup L_s$.
  (c) (r^*) je regulární výraz značící jazyk $L = L_r^*$ [19].

Plně uzávorkované regulární výrazy, které byly zavedeny definicí 2.4.5, mohou být zjednodušeny. Pro redukci počtu závorek ve výrazech předpokládejme, že „$*$“ má vyšší prioritu než „$\cdot$“ a „$\cdot$“ má vyšší prioritu než „$+$“. Mimoto výrazy lze zkrátit vynecháním symbolu „$\cdot$“. Navíc výraz rr^* je obvykle pro stručnost zapisován jako r^+. Tato zjednodušení činí regulární výrazy stručnějšími. Například výraz $((a)^*(b \cdot (b)^*))$ může být zapsán jako a^*b^+ [19]. Pro regulární výraz r, $L(r)$ označuje jazyk značený r.

Definice 2.4.6. Nechť L je jazyk nad abecedou Σ. L je *regulární jazyk* nad Σ, pokud $L = L(r)$ pro regulární výraz r nad Σ [19].

Definice 2.4.7. Dva regulární výrazy r a s jsou *ekvivalentní* tehdy a jen tehdy, když $L(r) = L(s)$ [19].

2.4.4 Výrazy zamíchání

Tato podsekce představuje *výrazy zamíchání* využívající stejnojmenné jazykové operace, včetně jejich sémantiky. Tyto výrazy jsou důležité pro vybrané důkazy následně uvedené v této práci.

Definice 2.4.8. Symboly $\emptyset$, ε a každé $w \in \Sigma^+$ jsou (atomické) *výrazy zamíchání* (anglicky *shuffle expressions*), dále označované jako *SHUF výrazy*. Pokud S_1 a S_2 jsou SHUF výrazy, potom

$$(S_1 + S_2),$$
$$(S_1 \sqcup\!\sqcup S_2),$$
$$(S_1)^{\sqcup\!\sqcup,*}$$

jsou SHUF výrazy [11].

Definice 2.4.9. Sémantika SHUF výrazů je definována následovně:

$$L(\emptyset) = \emptyset,$$
$$L(\varepsilon) = \{\varepsilon\},$$
$$L(w) = \{w\},$$

kde $w \in \Sigma^+$. Dále nechť S_1 a S_2 jsou SHUF výrazy. Potom platí

$$L(S_1 + S_2) = L(S_1) \cup L(S_2),$$
$$L(S_1 \sqcup\!\sqcup S_2) = L(S_1) \sqcup\!\sqcup L(S_2),$$
$$L(S_1^{\sqcup\!\sqcup,*}) = L(S_1)^{\sqcup\!\sqcup,*}$$

dle [11].

Dále *SHUF jazykem* rozumíme jazyk generovaný libovolným SHUF výrazem. Nechť **SHUF** značí rodinu jazyků generovaných SHUF výrazy.

Kapitola 3

Automaty a převodníky

Zatímco gramatiky slouží pro generování jazyků v oblasti formální teorie, *automaty* se využívají pro jejich rozpoznávání. Obecně pro vstupní řetězec automat rozhodne, zda daný řetězec náleží do příslušného jazyka či nikoliv. Pokud automaty nepracují pouze se vstupem a obsahují zároveň i výstup, neslouží pouze pro rozpoznávání jazyka, ale i pro převody mezi jednotlivými jazyky, přičemž jsou příslušně nazývány jako *převodníky*.

Nejprve sekce 3.1 definuje *konečné automaty* včetně jejich jednotlivých variant, konfigurací a přechodů. Pro názorné prezentování příslušná sekce zároveň obsahuje příklad, který důkladně popisuje jejich funkcionalitu. Ve stejném sledu sekce 3.2 přidává k zavedeným konečným automatům výstup a definuje jednotlivé varianty *převodníků*.

3.1 Definice a příklady konečných automatů

Cílem této sekce je definování a seznámení se s konečnými automaty jako jazykovými modely pro rozpoznávání jazyků. Postupně jsou představeny jednotlivé varianty konečných automatů počínaje těmi nejobecnějšími. Následně zavádí také klíčovou terminologii včetně konfigurace či přechodu a prezentuje funkcionalitu jednoduchého konečného automatu v názorném příkladu.

Pokud není uvedeno jinak, veškeré definice jsou převzaty z [20].

Definice 3.1.1. *Obecný konečný automat* (anglicky *general finite automaton*), zkráceně *GFA*, je pětice

$$M = (Q, \Sigma, R, s, F),$$

kde

- Q je konečná množina *stavů*,
- Σ je *vstupní abeceda*,
- $R \subseteq Q \times \Sigma^* \times Q$ je konečná relace nazývaná taktéž konečnou množinou *pravidel*,
- $s \in Q$ je *počáteční stav*,
- $F \subseteq Q$ je množina *koncových stavů*.

Namísto $(p, y, q) \in R$ je dále zapisováno $py \to q \in R$.

Definice 3.1.2. Nechť $M = (Q, \Sigma, R, s, F)$ je GFA. *Konfigurace* M je jakýkoliv řetězec $\chi \in Q\Sigma^*$. Relace *přechodu*, symbolicky značena $\vdash_M$, je definována nad $Q\Sigma^*$ jako

$$pyx \vdash_M qx,$$

tehdy a jen tehdy, když $pyx, qx \in Q\Sigma^*$ a $py \to q \in R$. Pokud M je pochopeno, je zapisováno $pyx \vdash qx$. Pokud $y = \varepsilon$, není ze vstupní pásky přečten žádný symbol.

Nechť $\vdash^n$, $\vdash^+$ a $\vdash^*$ značí příslušně *n-tou mocninu* $\vdash$ pro nějaké $n \geq 0$, tranzitivní uzávěr $\vdash$ a reflexivně-tranzitivní uzávěr $\vdash$.

Definice 3.1.3. Nechť $M = (Q, \Sigma, R, s, F)$ je GFA. *Jazyk přijímaný* M, značen $L(M)$, je definován jako

$$L(M) = \{w \mid w \in \Sigma^*, sw \vdash^* f, f \in F\}.$$

Definice 3.1.4. Nechť $M = (Q, \Sigma, R, s, F)$ je GFA. M je *GFA bez ε-přechodů*, pokud pro každé $py \to q \in R$ platí, že $y \neq \varepsilon$.

Definice 3.1.5. Nechť $M = (Q, \Sigma, R, s, F)$ je GFA. M je *konečný automat* (anglicky *finite automaton*), zkráceně *FA*, tehdy a jen tehdy, když $py \to q \in R$ implikuje, že $|y| \leq 1$.

Definice 3.1.6. Nechť $M = (Q, \Sigma, R, s, F)$ je FA. M je *deterministický FA* (anglicky *deterministic finite automaton*), zkráceně *DFA*, tehdy a jen tehdy, když $py \to q \in R$ implikuje, že $|y| = 1$ a $py \to q_1, py \to q_2 \in R$ implikuje, že $q_1 = q_2$ pro všechna $p, q, q_1, q_2 \in Q$ a $y \in \Sigma^*$.

Definice 3.1.7. Nechť $M = (Q, \Sigma, R, s, F)$ je DFA. M je *úplný FA* (anglicky *complete finite automaton*), zkráceně *CFA*, tehdy a jen tehdy, když pro všechna $p \in Q$ a pro všechna $a \in \Sigma$, $pa \to q \in R$ pro nějaké $q \in Q$.

Za účelem stručnějšího zápisu definic je někdy pravidlo $pa \to q$ zapisováno včetně unikátního označení r, přičemž $r : pa \to q$, jenž je formalizováno následující definicí.

Definice 3.1.8. Nechť $M = (Q, \Sigma, R, s, F)$ je FA. Nechť Ψ je abeceda *označení pravidel* (anglicky *rule labels*), přičemž $card(\Psi) = card(R)$ a ψ je bijekce z R do Ψ. Pro zjednodušení za účelem vyjádření, že ψ přiřazuje r pravidlo $pa \to q \in R$, kde $r \in \Psi$, je zapisováno $r : pa \to q \in R$. Tudíž $r : pa \to q$ odpovídá $\psi(pa \to q) = r$.

Definice 3.1.9. Nechť $M = (Q, \Sigma, R, s, F)$ je FA. Pro každé $y \in \Sigma^*$ a $r : pa \to q \in R$, M provede *přechod* z konfigurace pay do konfigurace qy podle r, zapsáno jako

$$pay \vdash qy\,[r].$$

Definice 3.1.10. Nechť $M = (Q, \Sigma, R, s, F)$ je FA. Nechť χ je jakákoliv konfigurace M. M provede *nula přechodů* z χ do χ podle ε, symbolicky zapsáno jako

$$\chi \vdash^0 \chi\,[\varepsilon].$$

Nechť existuje sekvence konfigurací $\chi_0, \chi_1, ..., \chi_n$ pro nějaké $n \geq 1$, přičemž

$$\chi_{i-1} \vdash \chi_i[r_i],$$

kde $r_i \in \Psi$ pro $i = 1, ..., n$. Pak M provede n *přechodů* z χ_0 do χ_n podle $r_1 ... r_n$, symbolicky zapsáno jako

$$\chi_0 \vdash^n \chi_n [r_1 ... r_n].$$

Dále $\vdash^+$ a $\vdash^*$ jsou definovány standardním způsobem.

Definice 3.1.11. Nechť $M = (Q, \Sigma, R, s, F)$ je FA a nechť χ a χ' jsou dvě konfigurace M. Lze předpokládat, že pro nějaké $n \geq 0$, $\chi \vdash^n \chi' [\rho]$ v M, kde $\rho = r_1 ... r_n$, přičemž $r_i \in R$ pro všechna $i = 1, ..., n$ ($\rho = \varepsilon$ pokud $n = 0$). Pak ρ, nazývané jako *řetězec pravidel* odpovídající $\chi \vdash^n \chi'$ v M, specifikuje sekvenci n pravidel, podle kterých M provádí $\chi \vdash^n \chi'$. Pokud ρ reprezentuje nadbytečnou informaci, $\chi \vdash^n \chi' [\rho]$ je zjednodušeno na $\chi \vdash^n \chi'$ [19].

Definice 3.1.12. Nechť $M = (Q, \Sigma, R, s, F)$ je FA a nechť χ a χ' jsou dvě konfigurace M.

(i) Pokud existuje $n \geq 1$, přičemž $\chi \vdash^n \chi' [\rho]$ v M, pak $\chi \vdash^+ \chi' [\rho]$.

(ii) Pokud existuje $n \geq 0$, přičemž $\chi \vdash^n \chi' [\rho]$ v M, pak $\chi \vdash^* \chi' [\rho]$.

Analogicky k $\vdash^n$ je možné zjednodušit $\vdash^+$ a $\vdash^*$ zápisem $\chi \vdash^+ \chi'$ namísto $\chi \vdash^+ \chi' [\rho]$ a zápisem $\chi \vdash^* \chi'$ namísto $\chi \vdash^* \chi' [\rho]$ [19].

Teorém 3.1.1. *Pro každý obecný konečný automat M existuje CFA M', přičemž $L(M') = L(M)$, viz [28].*

Teorém 3.1.2. *Jazyk K je regulární tehdy a jen tehdy, když existuje CFA M, přičemž $K = L(M)$, viz [28].*

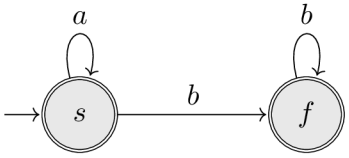
Tudíž *konečné automaty přijímají přesně rodinu regulárních jazyků.*

Příklad 3.1.1. Uvažujme FA

$$M = (\{s, f\}, \Sigma, R, s, \{s, f\}),$$

kde $\Sigma = \{a, b\}$ a

$$R = \{sa \to s, sb \to f, fb \to f\}.$$



Obrázek 3.1: Stavový diagram konečného automatu M.

Z počátečního stavu s může M nejprve přečíst libovolný počet symbolů a. Jakmile je pomocí M zpracován první symbol b, dále se již v řetězci přečteném ze vstupní pásky může vyskytovat libovolný počet pouze symbolů b, přičemž vstupní řetězec je zpracováván sekvenčně zleva doprava. Na základě skutečnosti, že počáteční stav s je zároveň stavem koncovým, je M přijat i prázdný řetězec ε a řetězec obsahující pouze symboly a. Tudíž přijímaným jazykem je jazyk

$$L(M) = \{a\}^* \{b\}^*.$$

Stavový diagram FA M je uveden na obrázku 3.1.

3.2 Definice a příklady převodníků

Jak již bylo uvedeno, konečné automaty mohou být rozšířeny o výstup, přičemž každý přechod obsahuje pro příslušný vstupní řetězec přiřazený řetězec výstupní. Tím umožňují převádění řetězců mezi jednotlivými jazyky. Souhrnně jsou příslušné varianty konečných automatů nazývány jako převodníky. Tato sekce kromě samotného definování převodníků zavádí také konečný převod a prezentuje v příkladu funkcionalitu vybrané varianty převodníku.

Pokud není uvedeno jinak, veškeré definice jsou převzaty z [24].

Definice 3.2.1. *Konečný převodník* (anglicky *finite transducer*), zkráceně *FT*, je šestice

$$T = (Q, \Sigma, \Delta, \sigma, s, F),$$

kde

- Q je konečná množina *stavů*,
- Σ je *vstupní abeceda*,
- Δ je *výstupní abeceda*,
- σ je přechodová a výstupní funkce z konečné podmnožiny z $Q \times \Sigma^*$ do konečných podmnožin z $Q \times \Delta^*$,
- $s \in Q$ je *počáteční stav*,
- $F \subseteq Q$ je množina *koncových stavů*.

Pro daný řetězec $u \in \Sigma^*$ je řečeno, že $v \in \Delta^*$ je výstup T pro u, pokud existuje sekvence přechodů T, $(q_1, v_1) \in \sigma(s, u_1)$, $(q_2, v_2) \in \sigma(q_1, u_2)$, ..., $(q_n, v_n) \in \sigma(q_{n-1}, u_n)$ a $q_n \in F$, to znamená

$$s \xrightarrow{u_1/v_1} q_1 \xrightarrow{u_2/v_2} ... \xrightarrow{u_n/v_n} q_n \in F,$$

přičemž $u = u_1u_2...u_n$, kde $u_1, u_2, ..., u_n \in \Sigma^*$ a $v = v_1v_2...v_n$, kde $v_1, v_2, ..., v_n \in \Delta^*$. Je zapisováno, že $v \in T(u)$, kde $T(u)$ značí množinu všech výstupů T pro vstupní řetězec u. Dále $s \in F$ implikuje, že $\varepsilon \in T(\varepsilon)$.

Definice 3.2.2. Nechť $T = (Q, \Sigma, \Delta, \sigma, s, F)$ je FT. T je *jednohodnotový FT*, pokud pro každý vstupní řetězec u, T obsahuje nanejvýš jeden odlišný výstup vzhledem k u, to znamená, že $|T(u)| \leq 1$ pro každé $u \in \Sigma^*$.

Definice 3.2.3. Nechť $T = (Q, \Sigma, \Delta, \sigma, s, F)$ je FT. T je *zobecněný sekvenční stroj* (anglicky *generalized sequential machine*), zkráceně *GSM*, pokud σ je funkcí z $Q \times \Sigma$ do konečných podmnožin z $Q \times \Delta^*$, to znamená, že T přečte právě jeden symbol při každém přechodu.

Definice 3.2.4. Nechť $T = (Q, \Sigma, \Delta, \sigma, s, F)$ je GSM. T je *deterministický GSM* (anglicky *deterministic generalized sequential machine*), zkráceně *DGSM*, pokud jeho základní konečný automat (to jest T bez výstupu) je DFA. Tudíž σ je částečná funkce z $Q \times \Sigma$ do $Q \times \Delta^*$.

Každý konečný převodník $T = (Q, \Sigma, \Delta, \sigma, s, F)$ definuje *konečný převod* $T : \Sigma^* \to 2^{\Delta^*}$. Pro vstupní řetězec $w \in \Sigma^*$, konečný převod $T(w)$, což je množina všech výstupních řetězců vzhledem k w, může být konečná nebo nekonečná. $T(w) = \emptyset$, pokud T nemůže dosáhnout koncového stavu čtením w. Lze poznamenat, že T je používáno pro označení jak konečného převodníku, tak i samotného konečného převodu z důvodu, že nelze pochybovat o významu. Pro jazyk $L \subseteq \Sigma^*$, je definováno

$$T(L) = \bigcup_{w \in L} T(w).$$

Konečný převod T může být také považován za relaci $R_T \subseteq \Sigma^* \times \Delta^*$ definovanou jako

$$R_T = \{(u, v) \mid v \in T(u)\}.$$

Definice 3.2.5. Nechť $T = (Q, \Sigma, \Delta, \sigma, s, F)$ je FT. T je *FT ve standardním tvaru*, pokud σ je funkcí z $Q \times (\Sigma \cup \{\varepsilon\})$ do $2^{Q \times (\Delta \cup \{\varepsilon\})}$. To znamená, že standardní tvar omezuje vstup a výstup pro každý přechod pouze na jeden symbol nebo ε.

Pro ustálení konvence zápisu FT s FA je dále přechodová a výstupní funkce FT, σ, uvedena jako množina *pravidel*. Tudíž namísto $(q, v) \in \sigma(p, u)$ je zapisováno $pu \to qv$.

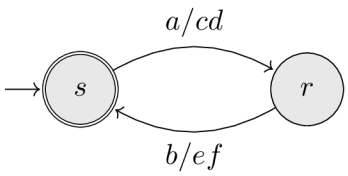
V celém textu je nadále využíván konečný převodník ve tvaru dle definice 3.2.1 z [24]. Za účelem odstranění pochybností lze upozornit na skutečnost, že se tato definice může lišit oproti některým publikacím, viz například [19], kde je uvažováno, že $\sigma \subseteq Q(\Sigma \cup \{\varepsilon\}) \times Q\Delta^*$.

Příklad 3.2.1. Uvažujme DGSM

$$T = (Q, \Sigma, \Delta, \sigma, s, F),$$

kde

$$\begin{aligned} Q &= \{s, r\}, \\ \Sigma &= \{a, b\}, \\ \Delta &= \{c, d, e, f\}, \\ \sigma &= \{sa \to rcd, \\ &\qquad rb \to sef\}, \\ F &= \{s\}. \end{aligned}$$



Obrázek 3.2: Stavový diagram deterministického zobecněného sekvenčního stroje T.

Z počátečního stavu s, který je zároveň koncovým stavem, může T přečíst symbol a, přičemž tento symbol je převeden na řetězec cd. Poté T musí přečíst symbol b, který je

převeden na řetězec ef, aby se opět začínalo od stavu s. Dochází tedy k převodu symbolu a na cd a symbolu b na ef, přičemž symboly a a b se musí ve vstupním řetězci nacházet přímo za sebou ve stejném pořadí a počet symbolů ve vstupním řetězci musí být sudý $(0, 2, ...)$. Vstupní řetězce T náleží do jazyka definovaného jako

$$L = \{ab\}^*.$$

Vyplývá, že

$$T(L) = \{cdef\}^*.$$

Stavový diagram DGSM T je uveden na obrázku 3.2

Kapitola 4

Skákající konečné automaty

V průběhu historie teorie automatů byl klasický konečný automat modifikován mnoha různými způsoby. Jednou z těchto modifikací, která se vyznačuje tím, že vstup není čten kontinuálním způsobem, jsou také *skákající konečné automaty*.

Tato kapitola představuje samotné skákající konečné automaty, které pracují jako klasické konečné automaty s významnou výjimkou, že nečtou vstupní řetězec zleva doprava po symbolech, ovšem po přečtení symbolu a přechodu do nového stavu může čtecí hlava přeskočit na libovolnou pozici ve zbývajícím vstupu. Jakmile je jednou symbol přečten ze vstupní pásky, nemůže být přečten znovu.

Kapitola poskytuje všechny potřebné znalosti týkající se skákajících konečných automatů pro pochopení další části textu a je uspořádána do čtyř sekcí následovně. Nejprve sekce 4.1 zavádí samotné skákající konečné automaty včetně jejich variant. Poté sekce 4.2 porovnává jejich sílu s dobře známými, jazyk definujícími formálními zařízeními, přičemž obsahuje pouze výčet vztahů. Následně sekce 4.3 prezentuje výsledky výzkumu uzávěrových vlastností rodin skákajících konečných automatů. V poslední řadě sekce 4.4 představuje charakteristiku, a to z hlediska operace zamíchání, rodiny jazyků přijímaných nejobecnější verzí skákajících konečných automatů, *obecnými skákajícími konečnými automaty*.

Obsah sekcí vychází z [20], kromě poslední sekce 4.4, která vychází z [11]. Veškeré definice jsou převzaty z [20].

4.1 Definice a příklady

V této sekci jsou definovány samotné skákající konečné automaty včetně jednotlivých variant, počínaje nejobecnější, a to obecným skákajícím konečným automatem. Mimo jiné tato sekce zavádí terminologie jako *konfiguraci* či relaci *skoku*. Uvedené formalismy následně využívá ilustrační příklad.

Veškeré definice jsou převzaty z [20].

Definice 4.1.1. *Obecný skákající konečný automat* (anglicky *general jumping finite automaton*), zkráceně *GJFA*, je pětice

$$M = (Q, \Sigma, R, s, F),$$

kde

- Q je konečná množina *stavů*,

- Σ je *vstupní abeceda*, $\Sigma \cap Q = \emptyset$,
- $R \subseteq Q \times \Sigma^* \times Q$ je konečná relace,
- $s \in Q$ je *počáteční stav*,
- $F \subseteq Q$ je množina *koncových stavů*.

Členy R jsou označovány jako *pravidla* M a namísto $(p, y, q) \in R$ je zapisováno $py \to q \in R$.

Definice 4.1.2. Nechť $M = (Q, \Sigma, R, s, F)$ je GJFA. *Konfigurace* M je libovolný řetězec v $\Sigma^* Q \Sigma^*$. Binární *relace skoku*, symbolicky značena $\curvearrowright_M$, nad $\Sigma^* Q \Sigma^*$, je definována následovně. Nechť $x, z, x', z' \in \Sigma^*$, přičemž $xz = x'z'$ a $py \to q \in R$. Pak M provede *skok* z $xpyz$ do $x'qz'$, zapsáno jako

$$xpyz \curvearrowright_M x'qz'.$$

Pokud M je pochopeno, je zapisováno $xpyz \curvearrowright x'qz'$. Standardním způsobem lze rozšířit relaci $\curvearrowright$ na $\curvearrowright^n$, kde $n \geq 0$. Na základě $\curvearrowright^n$ je možné definovat $\curvearrowright^+$ a $\curvearrowright^*$ označující tranzitivní uzávěr $\curvearrowright$ a reflexivně-tranzitivní uzávěr $\curvearrowright$.

Definice 4.1.3. Nechť $M = (Q, \Sigma, R, s, F)$ je GJFA. *Jazyk přijímaný* M, značen $L(M)$, je definován jako

$$L(M) = \{uv \mid u, v \in \Sigma^*, usv \curvearrowright^* f, f \in F\}.$$

Nechť $w \in \Sigma^*$. M *přijímá* w tehdy a jen tehdy, když $w \in L(M)$. M *nepřijímá* w tehdy a jen tehdy, když $w \in \Sigma^* - L(M)$. Dva GJFA M a M' jsou *ekvivalentní* tehdy a jen tehdy, když $L(M) = L(M')$.

Definice 4.1.4. Nechť $M = (Q, \Sigma, R, s, F)$ je GJFA. M je *GJFA bez ε-přechodů*, pokud $py \to q \in R$ implikuje, že $|y| \geq 1$. M je *stupně* n, kde $n \geq 0$, pokud $py \to q \in R$ implikuje, že $|y| \leq n$. M je *skákající konečný automat* (anglicky *jumping finite automaton*), zkráceně *JFA*, pokud se jedná o automat stupně 1.

Definice 4.1.5. Nechť $M = (Q, \Sigma, R, s, F)$ je JFA. M je *JFA bez ε-přechodů*, pokud $py \to q \in R$ implikuje, že $|y| = 1$. M je *deterministický JFA* (anglicky *deterministic JFA*), zkráceně *DJFA*, pokud M je JFA bez ε-přechodů a pro každé $p \in Q$ a každé $a \in \Sigma$ neexistuje více než jedno $q \in Q$ takové, že $pa \to q \in R$. M je *úplný JFA* (anglicky *complete JFA*), zkráceně *CJFA*, pokud M je DJFA a pro každé $p \in Q$ a každé $a \in \Sigma$ existuje právě jedno $q \in Q$ takové, že $pa \to q \in R$.

Definice 4.1.6. Nechť $M = (Q, \Sigma, R, s, F)$ je GJFA. *Přechodový graf* M, značen $\Delta(M)$, je multigraf, kde uzly jsou stavy z Q a existuje právě jedna hrana z p do q označena y tehdy a jen tehdy, když $py \to q \in R$. Stav $q \in Q$ je *dosažitelný*, pokud existuje cesta z s do q v $\Delta(M)$. Stav q je *ukončující*, pokud existuje cesta z q do nějakého $f \in F$. Pokud existuje cesta z p do q, $p = q_1, q_2, ..., q_n = q$ pro nějaké $n \geq 2$, kde $q_i y_i \to q_{i+1} \in R$ pro všechna $i = 1, ..., n-1$, lze zapsat

$$py_1y_2...y_n \rightsquigarrow q.$$

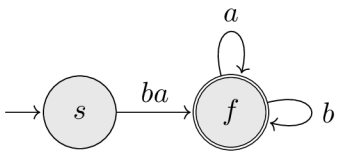
Dále jsou předchozí definice ilustrovány na ukázkovém příkladu.

Příklad 4.1.1. Uvažujme GJFA

$$M = (\{s, f\}, \{a, b\}, R, s, \{f\}),$$

kde

$$R = \{sba \to f, fa \to f, fb \to f\}.$$



Obrázek 4.1: Stavový diagram obecného skákajícího konečného automatu M.

Z počátečního stavu s musí M přečíst řetězec ba, který se může vyskytovat kdekoliv ve vstupním řetězci. Poté může M přečíst libovolný počet symbolů a a b, ale také již nemusí přečíst žádný symbol. Tudíž přijímaným jazykem je jazyk

$$L(M) = \{a, b\}^*\{ba\}\{a, b\}^*.$$

Stavový diagram GJFA M je uveden na obrázku 4.1.

Nechť **GJFA**, **GJFA**$^{-\varepsilon}$, **JFA**, **JFA**$^{-\varepsilon}$ a **DJFA** značí rodiny jazyků přijímaných po řadě obecnými skákajícími konečnými automaty, obecnými skákajícími konečnými automaty bez ε-přechodů, skákajícími konečnými automaty, skákajícími konečnými automaty bez ε-přechodů a deterministickými skákajícími konečnými automaty.

4.2 Vztahy se známými jazykovými rodinami

Cílem této sekce je definovat a vyobrazit vztahy rodin jazyků přijímaných obecnými skákajícími konečnými automaty (dále jen **GJFA**) a klasickými skákajícími konečnými automaty (dále jen **JFA**) s jazykovými rodinami dle Chomského hierarchie (viz sekce 2.3).

Pro konkrétní důkazy jednotlivých teorémů lze odkázat na [20].

Teorém 4.2.1. **FIN** $\subset$ **GJFA**.

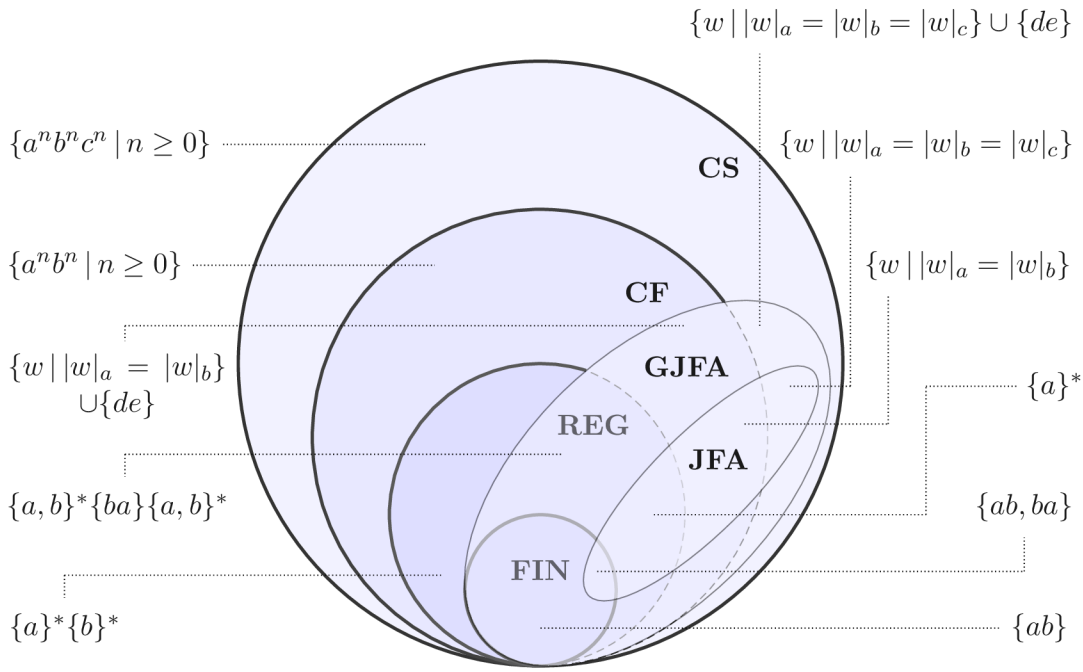
Teorém 4.2.2. **REG** *a* **GJFA** *jsou neporovnatelné.*

Teorém 4.2.3. **CF** *a* **GJFA** *jsou neporovnatelné.*

Teorém 4.2.4. **GJFA** $\subset$ **CS**.

Teorém 4.2.5. **FIN** *a* **JFA** *jsou neporovnatelné.*

Vztahy **GJFA** a **JFA** s jednotlivými uvedenými jazykovými rodinami dle Chomského hierarchie jsou vyobrazeny na obrázku 4.2.



Obrázek 4.2: Vztahy rodin jazyků přijímaných obecnými skákajícími konečnými automaty (**GJFA**) a klasickými skákajícími konečnými automaty (**JFA**) s jazykovými rodinami dle Chomského hierarchie. **FIN** značí rodinu konečných jazyků, **REG** značí rodinu jazyků generovaných regulárními gramatikami, **CF** značí rodinu jazyků generovaných bezkontextovými gramatikami a **CS** značí rodinu jazyků generovaných kontextovými gramatikami.

4.3 Uzávěrové vlastnosti

Tato sekce představuje uzávěrové vlastnosti jazykových rodin přijímaných obecnými skákajícími konečnými automaty (dále jen **GJFA**) a klasickými skákajícími konečnými automaty (dále jen **JFA**). Pro jednotlivé konkrétní důkazy uzávěrových vlastností je čtenář odkázán na [20]. Na tuto uvedenou publikaci navazuje [27], kde jsou zodpovězeny následující otevřené problémy:

- Je **GJFA** uzavřená vůči zamíchání?
- Je **GJFA** uzavřená vůči Kleeneho hvězdičce?
- Je **GJFA** uzavřená vůči Kleeneho plus?
- Je **GJFA** uzavřená vůči reverzaci?

A jsou upraveny důkazy pro uzávěrové vlastnosti **GJFA** vůči

(i) konečné substituci,

(ii) homomorfismu,

(iii) homomorfismu bez ε,

(iv) inverznímu homomorfismu.

Celkové shrnutí uzávěrových vlastností je uvedeno v tabulce 4.1.

	GJFA	JFA	REG
Značení konce	−	−	+
Konkatenace	−	−	+
Doplněk	−	+	+
Sjednocení	+	+	+
Průnik	−	+	+
Průnik s regulárními jazyky	−	−	+
Substituce	−	−	+
Regulární substituce	−	−	+
Konečná substituce	−	−	+
Homomorfismus	−	−	+
Homomorfismus bez ε	−	−	+
Inverzní homomorfismus	−	+	+
Kleeneho hvězdička	−	−	+
Kleeneho plus	−	−	+
Zamíchání	−	+	+
Reverzace	+	+	+

Tabulka 4.1: Uzávěrové vlastnosti jazykových rodin přijímaných obecnými skákajícími konečnými automaty (**GJFA**) a klasickými skákajícími konečnými automaty (**JFA**). **REG** značí rodinu regulárních jazyků. Symbol „+“ značí, že rodina jazyků je uzavřena vůči operaci. Symbol „−“ naopak značí, že rodina jazyků není uzavřena vůči operaci.

4.4 Operace zamíchání

Hlavním cílem této sekce je představení vztahu jazykové rodiny přijímané obecnými skákajícími konečnými automaty (dále jen **GJFA**) s rodinou jazyků generovaných SHUF výrazy, dále označované jako **SHUF**.

Jednotlivé vztahy jazykové rodiny přijímané klasickými skákajícími konečnými automaty (dále jen **JFA**) a **GJFA** s **SHUF** byly důkladně studovány v [11] a pro bližší informace je čtenář odkázán na uvedenou publikaci. Tato sekce obsahuje pouze výčet některých prokázaných skutečností stěžejních pro zbytek práce.

Veškeré příklady, lemmata a teorémy jsou převzaty z [11].

Příklad 4.4.1. Nechť

$$M = (Q, \Sigma, R, s, F)$$

je GJFA, kde

$$\begin{aligned} Q &= \{s\}, \\ \Sigma &= \{a, b, c, d\}, \\ R &= \{sab \to s, scd \to s\}, \\ F &= \{s\}, \end{aligned}$$

který přijímá jazyk $L(M)$, přičemž jej lze přirozeně převést na výraz zamíchání $S = (ab + cd)^{\sqcup\!\sqcup,*}$. Lze lehce ověřit, že každý řetězec, který je přijímaný M, může být zároveň vygenerován S, ale jelikož $acbd \in (L(S) \backslash L(M))$, potom vyplývá, že $L(M) \subsetneq L(S)$.

Lemma 4.4.1. *Nechť $M = (Q, \Sigma, R, s, F)$ je GJFA, kde*

$$
\begin{aligned}
Q &= \{s\}, \\
\Sigma &= \{a, b, c, d\}, \\
R &= \{sab \to s, scd \to s\}, \\
F &= \{s\}.
\end{aligned}
$$

Pak $L(M)$ není SHUF jazyk, viz [11].

Lemma 4.4.2. *Nechť $L = L(ac \sqcup\!\sqcup (bd)^{\sqcup\!\sqcup,*})$. $L \notin$ **GJFA**, viz [11].*

Teorém 4.4.3. **GJFA** *a* **SHUF** *jsou neporovnatelné.*

Důkaz. **SHUF** $\cap$ **GJFA** $\subsetneq$ **GJFA** vyplývá z definice a lemmatu 4.4.1. **SHUF** $\cap$ **GJFA** $\subsetneq$ **SHUF** vyplývá z definice a lemmatu 4.4.2. □

Kapitola 5

Vymazávací systémy

Skákající konečné automaty podléhají důkladnému studování a jedná se stále o aktuální téma výzkumu nejen v oblasti formální teorie, ale i jejich aplikace na reálné problémy. Nejobecnější varianta těchto automatů, obecný skákající konečný automat, jako stavové řízení využívá obecný konečný automat. V jisté paralele k regulovaným automatům lze uvažovat formální systém, který namísto stavového řízení využívá řídící regulární jazyk. Zároveň by bylo zachováno vymazávaní řetězců ze vstupní pásky, přičemž samotné regulární jazyky mohou být přijímány klasickými konečnými automaty.

Cílem této kapitoly je zavést nový formální systém v oblasti formální teorie, nazývaný *vymazávací systém*, prezentovat jej na několika příkladech a prokázat některé vztahy a vlastnosti rodiny jazyků přijímaných těmito systémy.

Celkově kapitola rozděluje jednotlivé úlohy do čtyř sekcí. Nejprve sekce 5.1 definuje samotný vymazávací systém včetně zavedení formalismů jako je *konfigurace* či *vymazávací krok*, a to nevyjímaje *jazyka přijímaného vymazávacím systémem*. Tato sekce rovněž představuje funkcionalitu vymazávacích systémů na několika ukázkových příkladech. Dále sekce 5.2 porovnává jejich sílu s dobře známými formálními zařízeními definujícími jazyk. Na tuto sekci navazuje sekce 5.3 prokazující vztahy rodiny jazyků přijímaných vymazávacími systémy a Dyckovými jazyky, včetně jazyků polo-Dyckových. Následně sekce 5.5 prezentuje vlastnosti jazykové rodiny přijímané vymazávacími systémy s jazyky generovanými výrazy zamíchání. Jako poslední, shrnuje sekce 5.6 dosažené teoretické výsledky zkoumání a diskutuje odlišné vlastnosti obecných skákajících konečných automatů a vymazávacích systémů.

5.1 Definice a příklady

Tato sekce definuje vymazávací systém a zavádí terminologie jako je konfigurace či relace vymazávacího kroku. Zároveň obsahuje veškerou používanou notaci v souvislosti s vymazávacími systémy. Následně uvedené formalismy a samotnou funkcionalitu nového formálního systému ilustruje na pěti názorných příkladech.

Definice 5.1.1. *Vymazávací systém* (anglicky *erasing system*), zkráceně *ES*, je trojice

$$ES = (\Sigma, E, R),$$

kde

- Σ je *abeceda*, která vždy obsahuje speciální symbol, #,

- $E \subseteq (\Sigma - \{\#\})^*$ je konečná množina *vymazávacích řetězců* (anglicky *erasing strings*),
- $R \subseteq (\Sigma - \{\#\})^*$ je *regulární jazyk*.

Definice 5.1.2. Nechť $ES = (\Sigma, E, R)$ je ES. Položme $\overline{\Sigma} = \Sigma - \{\#\}$. *Konfigurace* ES je libovolný člen $(u, v) \in \overline{\Sigma}^*\{\#\}\overline{\Sigma}^* \times \overline{\Sigma}^*$. Nechť pro každý řetězec $u \in E$ a pro každý řetězec $x, y, x', y', z \in \overline{\Sigma}^*$, přičemž $xy = x'y'$,

$$(x\#uy, z) \triangleright_{ES} (x'\#y', zu)$$

je *vymazávací krok*, kde $\# \in \Sigma$. Nechť $\triangleright_{ES}^n$ je n-tá mocnina relace $\triangleright_{ES}$, kde $n \geq 0$. Nechť $\triangleright_{ES}^+$ je tranzitivní uzávěr relace $\triangleright_{ES}$ a $\triangleright_{ES}^*$ je reflexivně-tranzitivní uzávěr relace $\triangleright_{ES}$.

Neformálně speciální symbol, #, určuje pozici ve vstupním řetězci, za kterou se nachází vymazávací řetězec, který bude v následujícím vymazávacím kroku vyjmut ze vstupního řetězce. V každém vymazávacím kroku vstupní řetězec obsahuje právě jeden symbol #.

Definice 5.1.3. Nechť $ES = (\Sigma, E, R)$ je ES a $\overline{\Sigma} = \Sigma - \{\#\}$. *Jazyk přijímaný* ES, značený $L(E, R)$, je definován jako

$$L(E, R) = \{uv \mid u, v \in \overline{\Sigma}^*, (u\#v, \varepsilon) \triangleright_{ES}^* (\#, w), \# \in \Sigma, w \in R\}.$$

Nechť $w \in L(E, R)$. Říkáme, že w *náleží* ES tehdy a jen tehdy, když $w \in L(E, R)$. Řetězec w *nenáleží* ES tehdy a jen tehdy, když $w \in \Sigma^* - L(E, R)$. Dva systémy ES definovány E, R a E', R' jsou *ekvivalentní* tehdy a jen tehdy, když $L(E, R) = L(E', R')$.

Definice 5.1.4. Nechť $ES = (\Sigma, E, R)$ je ES. ES je *stupně* n, kde $n \geq 0$, značeno ES^n, pokud $u \in E$ implikuje, že $|u| \leq n$.

Dále jsou ilustrovány předchozí definice na pěti příkladech.

Příklad 5.1.1. Uvažujme ES,

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned} \Sigma &= \{a, b, c\}, \\ E &= \{a, b, c\}, \\ R &= \{abc\}^*. \end{aligned}$$

Jazyk

$$L(E, R) = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$$

je jazykem přijímaným ES. Poté lze pro vstupní řetězec $w = ababcc$ provést vymazávací kroky

$$
\begin{aligned}
(ab\#abcc, \varepsilon) &\triangleright_{ES} (a\#bbcc, a) \\
&\triangleright_{ES} (abc\#c, ab) \\
&\triangleright_{ES} (\#abc, abc) \\
&\triangleright_{ES} (\#bc, abca) \\
&\triangleright_{ES} (\#c, abcab) \\
&\triangleright_{ES} (\#, abcabc),
\end{aligned}
$$

ze kterých vyplývá $abcabc \in R$, což implikuje $w \in L(E, R)$.

Lze poukázat na skutečnost, že jazyk $L(E, R)$ v příkladu 5.1.1 je dobře známý *kontextový jazyk*.

Příklad 5.1.2. Uvažujme ES,

$$ES = (\Sigma, E, R),$$

kde

$$
\begin{aligned}
\Sigma &= \{a, b\}, \\
E &= \{a, b\}, \\
R &= \{ab\}^*\{a\}^+.
\end{aligned}
$$

Jazyk

$$L(E, R) = \{w \in \{a, b\}^+ \,|\, |w|_a > |w|_b\}$$

je jazykem přijímaným ES. Poté lze pro vstupní řetězec $w = aabbaba$ provést vymazávací kroky

$$
\begin{aligned}
(aabb\#aba, \varepsilon) &\triangleright_{ES} (aa\#bbba, a) \\
&\triangleright_{ES} (aabb\#a, ab) \\
&\triangleright_{ES} (aab\#b, aba) \\
&\triangleright_{ES} (\#aab, abab) \\
&\triangleright_{ES} (a\#b, ababa) \\
&\triangleright_{ES} (\#a, ababab) \\
&\triangleright_{ES} (\#, abababa),
\end{aligned}
$$

ze kterých vyplývá $abababa \in R$, což implikuje $w \in L(E, R)$, přičemž jazyk $L(E, R)$ je *bezkontextový jazyk*.

Příklad 5.1.3. Uvažujme ES,

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= \{a, b\},\\ E &= \{a, b\},\\ R &= \{bb, a\}^*\{b\}.\end{aligned}$$

Jazyk

$$L(E, R) = \{w \in \{a, b\}^+ \,|\, |w|_b \text{ je lichý}\}$$

je jazykem přijímaným ES. Poté lze pro vstupní řetězec $w = aababba$ provést vymazávací kroky

$$\begin{aligned}(aababb\#a, \varepsilon) &\rhd_{ES} (aa\#babb, a)\\ &\rhd_{ES} (aaab\#b, ab)\\ &\rhd_{ES} (a\#aab, abb)\\ &\rhd_{ES} (\#aab, abba)\\ &\rhd_{ES} (\#ab, abbaa)\\ &\rhd_{ES} (\#b, abbaaa)\\ &\rhd_{ES} (\#, abbaaab),\end{aligned}$$

ze kterých vyplývá $abbaaab \in R$, což implikuje $w \in L(E, R)$, přičemž jazyk $L(E, R)$ je *regulární jazyk*.

Příklad 5.1.4. Uvažujme ES,

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= \{a, b\},\\ E &= \{a, b, ba\},\\ R &= \{ba\}\{a, b\}^*.\end{aligned}$$

Jazyk

$$L(E, R) = \{a, b\}^*\{ba\}\{a, b\}^*$$

je jazykem přijímaným ES. Poté lze pro vstupní řetězec $w = baabbaaa$ provést vymazávací kroky

$$
\begin{aligned}
(baab\#baaa, \varepsilon) &\rhd_{ES} (ba\#abaa, ba) \\
&\rhd_{ES} (bab\#aa, baa) \\
&\rhd_{ES} (\#baba, baaa) \\
&\rhd_{ES} (\#aba, baaab) \\
&\rhd_{ES} (\#ba, baaaba) \\
&\rhd_{ES} (\#a, baaabab) \\
&\rhd_{ES} (\#, baaababa),
\end{aligned}
$$

ze kterých vyplývá $baaababa \in R$, což implikuje $w \in L(E, R)$, přičemž jazyk $L(E, R)$ je regulární jazyk.

Příklad 5.1.5. Uvažujme ES,

$$ES = (\Sigma, E, R),$$

kde

$$
\begin{aligned}
\Sigma &= \{a, b, c\}, \\
E &= \{ab, c\}, \\
R &= \{c\}^*\{ab\}.
\end{aligned}
$$

Jazyk

$$L(E, R) = \{c\}^*\{a\}\{c\}^*\{b\}\{c\}^*$$

je jazykem přijímaným ES. Poté lze pro vstupní řetězec $w = cccaccbcc$ provést vymazávací kroky

$$
\begin{aligned}
(cc\#caccbcc, \varepsilon) &\rhd_{ES} (ccaccbc\#c, c) \\
&\rhd_{ES} (cca\#ccbc, cc) \\
&\rhd_{ES} (cca\#cbc, ccc) \\
&\rhd_{ES} (ccab\#c, cccc) \\
&\rhd_{ES} (\#ccab, ccccc) \\
&\rhd_{ES} (\#cab, cccccc) \\
&\rhd_{ES} (\#ab, ccccccc) \\
&\rhd_{ES} (\#, cccccccab),
\end{aligned}
$$

ze kterých vyplývá $cccccccab \in R$, což implikuje $w \in L(E, R)$, přičemž jazyk $L(E, R)$ je regulární jazyk.

Dále nechť **ES** značí rodinu jazyků přijímaných ES.

5.2 Vztahy se známými jazykovými rodinami

Cílem této kapitoly je stanovit vztahy mezi rodinou jazyků přijímaných vymazávacími systémy (dále jen **ES**) a známými jazykovými rodinami, včetně rodiny konečných jazyků (dále jen **FIN**), rodiny regulárních jazyků (dále jen **REG**), rodiny bezkontextových jazyků (dále jen **CF**) a v poslední řadě rodiny kontextových jazyků (dále jen **CS**). Následně obsažená ilustrace graficky vyobrazuje vztah **ES** s těmito rodinami jazyků.

Na základě jisté paralely vymazávacích systému (dále jen ES) s obecnými skákajícími konečnými automaty (dále jen GJFA) lze na definovaný systém aplikovat některé důkazy v upravené formě jako pro GJFA. Tudíž důkazy pro

- teorém 5.2.4, 5.2.5, 5.2.6,
- lemma 5.2.3

jsou inspirovány z [20], jenž zavádí GJFA do formální teorie.

Lemma 5.2.1. $\{a, b, ab\} \notin \mathbf{ES}$.

Důkaz. Lze provést sporem. Nechť $K = \{a, b, ab\}$. Předpokládejme, že existuje ES, $ES = (\Sigma, E, R)$, takový, pro který platí, že $L(E, R) = K$. Nechť $w = ba$, přičemž $w \notin K$. Poněvadž řetězce a, b a ab náleží K, musí náležet také do množiny vymazávacích řetězců E a do regulárního jazyka R. Vyplývá, že lze vymazávací řetězce a a b využít při přijímání w tak, že nejprve je vymazán řetězec a a poté řetězec b, přičemž řetězec vzniklý jejich konkatenací, ab, náleží R. To implikuje, že $w \in K$ a dochází ke sporu s předpokladem, že $L(E, R) = K$. Tudíž neexistuje ES takový, který by přijímal jazyk $\{a, b, ab\}$. □

Teorém 5.2.2. **FIN** a **ES** *jsou neporovnatelné.*

Důkaz. $\mathbf{ES} \not\subseteq \mathbf{FIN}$ vyplývá z příkladu 5.1.1. $\mathbf{FIN} \not\subseteq \mathbf{ES}$ vyplývá z lemmatu 5.2.1. □

Lemma 5.2.3. $\{a\}^*\{b\}^* \notin \mathbf{ES}$.

Důkaz. Lze provést sporem. Nechť $K = \{a\}^*\{b\}^*$. Předpokládejme, že existuje ES, $ES = (\Sigma, E, R)$, takový, pro který platí, že $L(E, R) = K$. Nechť $w = a^n b$, kde n je stupeň ES. Poněvadž $w \in K$, během vymazávání w vymazávací řetězec $a^i b \in E$, kde $0 \leq i < n$, musí být použit. Nicméně, pokud je řetězec $a^i b$ vymazán v jediném kroku a všechny další symboly ve w jsou stejné, řetězec $a^i b a^{n-i}$ může být přijat definovaným ES za použití stejných vymazávacích řetězců během přijímání w. To implikuje, že $a^i b a^{n-i} \in K$. Jedná se tedy o rozpor s předpokladem, že $L(E, R) = K$. Tudíž neexistuje ES takový, který by přijímal jazyk $\{a\}^*\{b\}^*$. □

Teorém 5.2.4. **REG** *a* **ES** *jsou neporovnatelné.*

Důkaz. $\mathbf{ES} \not\subseteq \mathbf{REG}$ vyplývá z příkladu 5.1.1. $\mathbf{REG} \not\subseteq \mathbf{ES}$ vyplývá z lemmatu 5.2.3. □

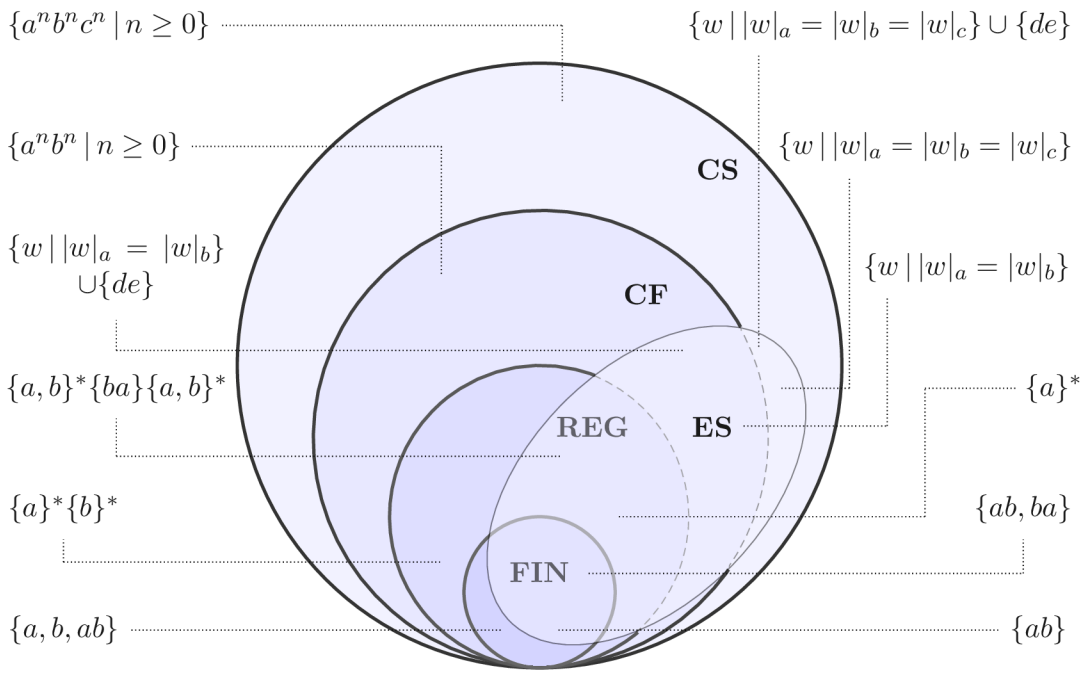
Teorém 5.2.5. **CF** *a* **ES** *jsou neporovnatelné.*

Důkaz. $\mathbf{ES} \not\subseteq \mathbf{CF}$ vyplývá z příkladu 5.1.1. $\mathbf{CF} \not\subseteq \mathbf{ES}$ vyplývá z lemmatu 5.2.3. □

Teorém 5.2.6. $\mathbf{ES} \subset \mathbf{CS}$.

Důkaz. Vymazávací krok může být simulován kontextovými gramatikami, tedy $\mathbf{ES} \subseteq \mathbf{CS}$. Z lemmatu 5.2.3 vyplývá, že $\mathbf{CS} - \mathbf{ES} \neq \emptyset$, což dokazuje teorém. □

Dále obrázek 5.1 vyobrazuje vztahy **ES** s jednotlivými uvedenými jazykovými rodinami.



Obrázek 5.1: Vztah rodiny jazyků přijímaných vymazávacími systémy (**ES**) s jazykovými rodinami dle Chomského hierarchie. **FIN** značí rodinu konečných jazyků, **REG** značí rodinu jazyků generovaných regulárními gramatikami, **CF** značí rodinu jazyků generovaných bezkontextovými gramatikami a **CS** značí rodinu jazyků generovaných kontextovými gramatikami.

5.3 Vztahy s Dyckovými a polo-Dyckovými jazyky

Významnou vlastností vymazávacích systémů jsou vztahy s Dyckovými a polo-Dyckovými jazyky. Obecně oba typy jazyků obsahují dvojice komplementárních symbolů, jenž mohou v případě klasických Dyckových jazyků představovat správné uzávorkování.

Pokud bude uvažováno, že jednotlivé dvojice odpovídajících symbolů, jenž jsou zapsány za sebou, budou z řetězce postupně vymazávány, lze takto ověřit, zda:

(i) řetězec obsahuje pouze povolené symboly,

(ii) obsahuje pouze správné vložení komplementárních dvojic, ať již se striktním určením pořadí dvou symbolů či nikoliv.

Z této myšlenky vychází následující dva teorémy, které prezentují, že Dyckovy jazyky a polo-Dyckovy jazyky mohou být přijímány vymazávacími systémy.

Teorém 5.3.1. $\boldsymbol{\mathcal{D}} \subset$ **ES**.

Důkaz. Nechť $L \in \boldsymbol{\mathcal{D}}$. Dále nechť T označuje abecedu terminálů gramatiky generující L (viz definice 2.4.1). Jelikož L je Dyckův jazyk, existuje $n \geq 1$ takové, že lze T vyjádřit jako

$$T = \{a_1, a'_1, a_2, a'_2, ..., a_n, a'_n\}.$$

Dále je definován ES,

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= T = \{a_1, a'_1, a_2, a'_2, ..., a_n, a'_n\},\\ E &= \{a_i a'_i \mid 1 \leq i \leq n\},\\ R &= E^*.\end{aligned}$$

Z definice 5.1.3 vyplývá, že jazyk přijímaný definovaným ES je jazyk $L(E, R) = L$. Tudíž $\boldsymbol{\mathcal{D}} \subseteq \mathbf{ES}$. Z příkladu 5.1.1 vyplývá, že $\mathbf{ES} - \boldsymbol{\mathcal{D}} \neq \emptyset$, což dokazuje teorém. □

Teorém 5.3.2. $\boldsymbol{\mathcal{S}} \subset \mathbf{ES}$.

Důkaz. Nechť $L \in \boldsymbol{\mathcal{S}}$. Dále nechť T označuje abecedu terminálů gramatiky generující L (viz definice 2.4.2). Jelikož L je polo-Dyckův jazyk, existuje $n \geq 1$ takové, že lze T vyjádřit jako

$$T = \{a_1, a'_1, a_2, a'_2, ..., a_n, a'_n\}.$$

Dále je definován ES,

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= T = \{a_1, a'_1, a_2, a'_2, ..., a_n, a'_n\},\\ E &= \{a_i a'_i, a'_i a_i \mid 1 \leq i \leq n\},\\ R &= E^*.\end{aligned}$$

Z definice 5.1.3 vyplývá, že jazyk přijímaný definovaným ES je jazyk $L(E, R) = L$. Tudíž $\boldsymbol{\mathcal{S}} \subseteq \mathbf{ES}$. Z příkladu 5.1.1 vyplývá, že $\mathbf{ES} - \boldsymbol{\mathcal{S}} \neq \emptyset$, což dokazuje teorém. □

5.4 Uzávěrové vlastnosti

V této sekci jsou prokázány uzávěrové vlastnosti rodiny jazyků přijímaných vymazávacími systémy (dále jen **ES**) vůči různým operacím.

Na základě podobnosti vymazávacího systému (dále jen ES) s obecnými skákajícími konečnými automaty (dále jen GJFA) lze na některé teorémy či lemmata aplikovat totožné důkazy v modifikované formě. Tomu tak je dle [20] pro

- teorém 5.4.1, 5.4.2, 5.4.3, 5.4.10, 5.4.12,
- důsledek 5.4.12.1,
- lemma 5.4.11

a dle [27] pro

- teorém 5.4.9, 5.4.13, 5.4.15, 5.4.16,
- lemma 5.4.5, 5.4.6, 5.4.7, 5.4.8,
- příklad 5.4.1.

Teorém 5.4.1. ES *není uzavřená vůči značení konce (anglicky endmarking).*

Důkaz. Uvažujme jazyk $K = \{a\}^*$, přičemž $K \in$ **ES**. Důkaz, že neexistuje ES, který by přijímal $K\{\$\}$, kde \$ je takový symbol, že $\$ \neq a$, lze provést analogicky k lemmatu 5.2.3. □

Teorém 5.4.2. ES *není uzavřená vůči konkatenaci.*

Důkaz. Teorém 5.4.1 implikuje, že **ES** není uzavřená vůči konkatenaci, přičemž uvažujme, že ES,

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= \{\$\},\\ E &= \{\$\},\\ R &= \{\$\},\end{aligned}$$

přijímá $\{\$\}$.

□

Teorém 5.4.3. ES *není uzavřená vůči doplňku.*

Důkaz. Uvažujme ES z příkladu 5.1.4. Doplňkem jazyka $L(E, R)$ daného ES je jazyk

$$\overline{L(E,R)} = \{a\}^*\{b\}^*,$$

přičemž neexistuje žádný ES takový, který by přijímal jazyk $\overline{L(E,R)}$ (viz lemma 5.2.3). □

Teorém 5.4.4. ES *není uzavřená vůči sjednocení.*

Důkaz. Lze provést sporem. Nechť $ES_1 = (\Sigma_1, E_1, R_1)$, $ES_2 = (\Sigma_2, E_2, R_2)$ a $ES_u = (\Sigma_u, E_u, R_u)$ jsou tři ES. Nechť je definováno, že

$$\begin{aligned}\Sigma_u &= \Sigma_1 \cup \Sigma_2,\\ E_u &= E_1 \cup E_2,\\ R_u &= R_1 \cup R_2,\end{aligned}$$

z čehož je předpokládáno, že jazyk přijímaný ES, ES_u, je jazyk $L(E_u, R_u) = L(E_1, R_1) \cup L(E_2, R_2)$. Uvažujme, že pro ES_1,

$$\Sigma_1 = \{a, c, d\},$$
$$E_1 = \{a, cd\},$$
$$R_1 = \{cda\},$$

přičemž ES_1 přijímá jazyk

$$L(E_1, R_1) = \{acd, cda\}.$$

Pro ES_2 uvažujme, že

$$\Sigma_2 = \{b, c, d\},$$
$$E_2 = \{b, c, d\},$$
$$R_2 = \{cdb\}.$$

Přijímaným jazykem ES_2 je jazyk

$$L(E_2, R_2) = \{bcd, cbd, cdb\}$$

Uvažujme řetězec $w = cad$, přičemž $w \notin L(E_1, R_1)$ a $w \notin L(E_2, R_2)$ dle definice 5.1.3. Poněvadž vymazávací řetězce $c, d \in E_u$, které obsahují jeden symbol a jenž původně náležely pouze do množiny E_2, mohou být použity při přijímání řetězce w pomocí ES_u, je implikováno, že $w \in L(E_1, R_1) \cup L(E_2, R_2)$. Tudíž dochází ke sporu vůči předpokladu, že $L(E_u, R_u) = L(E_1, R_1) \cup L(E_2, R_2)$. Ve skutečnosti je řetězec w přijat na základě faktu, že řetězec vzniklý konkatenací vymazávacích řetězců použitých při přijímání w náleží do R_1, přičemž $R_1 \subseteq R_u$. Lze si také povšimnout, že jako regulární jazyky R_1 a R_2 byly využity jazyky, které jsou zároveň jazyky konečnými. □

Lemma 5.4.5. *Nechť $ES = (\Sigma, E, R)$ je ES. Položme $\overline{\Sigma} = \Sigma - \{\#\}$. Pro každé $z, z'' \in \overline{\Sigma}^*$ a $w, w'' \in \overline{\Sigma}^*$ následující jsou ekvivalentní:*

(i) $(x_1\#x_2, z) \rhd_{ES}^* (x_1''\#x_2'', z'')$ *pro* $\# \in \Sigma$, $x_1, x_2, x_1'', x_2'' \in \overline{\Sigma}^*$ *s* $x_1x_2 = w$ *a* $x_1''x_2'' = w''$.

(ii) $z = z''$ *a* $w = w''$ *nebo*

$$w \in w'' \leftarrow u_d \leftarrow u_{d-1} \leftarrow ... \leftarrow u_2 \leftarrow u_1, \tag{5.1}$$

kde $u_1, u_2, ..., u_d \in E$ *a* $z'' = zu_1u_2...u_d$, *kde* $d \geq 1$.

Důkaz. Nejprve předpokládejme, že bod (i) platí a označme

$$c = (x_1\#x_2, z),$$
$$c'' = (x_1''\#x_2'', z'').$$

Pokud $c = c''$, potom $z = z''$ a $w = w''$. V opačném případě $d_1 \rhd_{ES} ... \rhd_{ES} d_i$ pro nějaké konfigurace $d_1, ..., d_i$ ES s $c = d_1$, $c'' = d_i$, kde $i \geq 2$. Následně bude použita indukce podle i.

Pokud $i = 2$, pak $c \rhd_{ES} c''$ a podle definice $\rhd_{ES}$ existuje $u \in E$ a $t_1, t_2, t_1'', t_2'', y \in \overline{\Sigma}^*$, přičemž $x_1 = t_1$, $x_2 = ut_2$, $x_1'' = t_1''$, $x_2'' = t_2''$ a $z'' = zu$. Tudíž $w \in w'' \leftarrow u$ a $z'' = zu$, kde $u \in E$.

Pokud $i \geq 3$, pak $c \rhd_{ES} c' \rhd_{ES} c''$ pro nějakou konfiguraci $c' = (x_1'\#x_2', z')$. Označme $w' = x_1'x_2'$. Dle indukčního předpokladu aplikovaného na z', z'', w', w'' získáme, že

$$z'' = z'u_2...u_d,$$

kde $u_2, ..., u_d \in E$ s $d \geq 2$ a $w' \in w'' \leftarrow u_d \leftarrow ... \leftarrow u_2$. Pro uzavření důkazu je toto dostatečné k prokázání, že $z' = zu$ a $w \in w' \leftarrow u$, kde $u \in E$ pro $u = u_1$, z čehož obojí vyplývá z $c \rhd_{ES} c'$ dle předchozí analýzy případu pro $i = 2$.

Za druhé, nechť bod (ii) platí. Pokud $z = z''$ a $w = w''$, pak $(x_1\#x_2, z) = (x_1''\#x_2'', z'')$ pro $x_1 = x_1'' = \varepsilon$, $x_2 = w$ a $x_2'' = w''$. V opačném případě stanovme

$$z'' = zu_1u_2...u_d,$$

kde $u_1, u_2, ..., u_d \in E$ pro $d \geq 1$. Dále bude použita indukce podle d. Označme $u = u_1$.

Nechť $d = 1$. Pak $w \in w'' \leftarrow u$ a $z'' = zu$, kde $u \in E$. Dle definice $\leftarrow$ (viz definice 2.4.3) existují $t_1, t_2 \in \overline{\Sigma}^*$, přičemž $w = t_1ut_2$ a $w'' = t_1t_2$. Přirozeně, $(t_1\#ut_2, z) \rhd_{ES} (t_1\#t_2, z'')$ vyplývá z definice $\rhd_{ES}$. Jelikož $\rhd_{ES}$ je speciální případ $\rhd_{ES}^*$, je důkaz pro $d = 1$ uzavřen.

Nechť $d \geq 2$. Z výrazu 5.1 a definice $\leftarrow$ vyplývá, že

$$w \in w' \leftarrow u \tag{5.2}$$

pro nějaké $w' \in \overline{\Sigma}^*$ s

$$w' \in w'' \leftarrow u_d \leftarrow u_{d-1} \leftarrow ... \leftarrow u_2. \tag{5.3}$$

Označme $z' = zu$. Dle indukčního předpokladu aplikovaného na z', z'', w', w'' získáme

$$(x_1'\#x_2', z') \rhd_{ES}^* (x_1''\#x_2'', z'')$$

pro nějaké $x_1', x_2', x_1'', x_2'' \in \overline{\Sigma}^*$ s $x_1'x_2' = w'$ a $x_1''x_2'' = w''$.

Zbývá prokázat, že $(x_1\#x_2, z) \rhd_{ES} (x_1'\#x_2', z')$ pro nějaké x_1, x_2 s $x_1x_2 = w$. Vzhledem k výrazu 5.2 existují $t_1, t_2 \in \overline{\Sigma}^*$, přičemž $w = t_1ut_2$ a $w' = t_1t_2$. Z definice $\rhd_{ES}$ dohromady s $x_1'x_2' = w' = t_1t_2$ a tím, že $u \in E$, vyplývá, že

$$(t_1\#ut_2, z) \rhd_{ES} (x_1'\#x_2', z').$$

Důkaz uzavřeme označením, že $x_1 = t_1$ a $x_2 = ut_2$. □

Lemma 5.4.6. *Nechť $ES = (\Sigma, E, R)$ je ES, $\overline{\Sigma} = \Sigma - \{\#\}$ a $w \in \overline{\Sigma}^*$. Potom $w \in L(E, R)$ tehdy a jen tehdy, když $w = \varepsilon$ a $\varepsilon \in R$ nebo*

$$w \in \varepsilon \leftarrow u_d \leftarrow u_{d-1} \leftarrow ... \leftarrow u_2 \leftarrow u_1,$$

kde $u_1, u_2, ..., u_d \in E$ a $u_1u_2...u_d \in R$ pro $d \geq 1$.

Důkaz. Pokud $w \in L(E,R)$, pak $w = x_1x_2$ pro $x_1, x_2 \in \overline{\Sigma}^*$ s $(x_1\#x_2, z) \rhd_{ES}^* (\#, z')$, kde $z = \varepsilon$ a $z' \in R$. Je aplikována dopředná implikace lemmatu 5.4.5 na z, z', w a ε. Na druhou stranu pro nějaké z' platí, že $z' \in R$ a je aplikována zpětná implikace lemmatu 5.4.5 na z, z', w a ε. □

Příklad 5.4.1. Uvažujme následující dva jazyky, které oba náleží **ES** (viz teorém 5.3.1).

1. Dyckův jazyk $\mathcal{D}$ nad abecedou $T_1 = \{a, \overline{a}\}$, $\mathcal{D} = \varepsilon \leftarrow^* a\overline{a}$.
2. Jakýkoliv Dyckův jazyk $\mathcal{D}_n$ nad abecedou $T_n = \{a_1, \overline{a_1}, a_2, \overline{a_2}, ..., a_n, \overline{a_n}\}$.

Lemma 5.4.7. **ES** $\subseteq$ **UC**.

Důkaz. Uvažujme, že $ES = (\Sigma, E, R)$ je ES. Nechť $\mathcal{P}$ je množina všech řetězců $v = u_1u_2...u_d$, přičemž $u_1, u_2, ..., u_d \in E$ a $v \in R$ pro $d \geq 1$. Podle lemmatu 5.4.6 definujeme

$$L(E,R) - \{\varepsilon\} = \bigcup_{p \in \mathcal{P}} (\varepsilon \leftarrow u_{p,d_p} \leftarrow u_{p,d_p-1} \leftarrow ... \leftarrow u_{p,2} \leftarrow u_{p,1}),$$

kde $u_{p,1}u_{p,2}...u_{p,d_p} = p$, $u_{p,1}, u_{p,2}, ..., u_{p,d_p} \in E$, $d_p \geq 1$. Jelikož $\{\varepsilon\}$ je kompozice, $L(E,R) \in UC_n$, kde $n = max\{|u| \mid u \in E\}$. □

Lemma 5.4.8. $\{ab\}^* \notin$ **ES**.

Důkaz. Lze provést sporem. Předpokládejme, že $L \in$ **ES**. Podle lemmatu 5.4.7, $L \in$ **UC**, a tudíž $L \in$ **UC**$_n$ pro nějaké $n \geq 0$. Pokud $n = 0$, lze pozorovat, že $L = \{\varepsilon\}$, a tudíž dochází ke sporu. V opačném případě stanovme $w = (ab)^{n+1}$. Dle definice UC_n, w náleží kompozici $K \subseteq L$ ve formě

$$K = \varepsilon \leftarrow u_d \leftarrow u_{d-1} \leftarrow ... \leftarrow u_2 \leftarrow u_1$$

stupně n. Jelikož $w \neq \varepsilon$, existuje nejméně c s $d \geq c \geq 1$ a $u_c \neq \varepsilon$. Navíc

$$K = K' \leftarrow u$$

pro vhodné K' a $u = u_c$. Tudíž $w = x_1ux_2$ pro $x_1x_2 \in K'$. Jelikož $|u| \leq n$, platí alespoň jeden z následujících případů:

(i) Předpokládejme, že $|x_1| \geq 2$, zapsáno jako $x_1 = ab\overline{x_1}$. Pokud u začíná symbolem a, vyplývá $aub\overline{x_1}x_2 \in K$. Pokud u začíná symbolem b, vyplývá $abu\overline{x_1}x_2 \in K$.

(ii) Předpokládejme, že $|x_2| \geq 2$, zapsáno jako $x_2 = \overline{x_2}ab$. Pokud u začíná symbolem a, vyplývá $x_1\overline{x_2}aub \in K$. Pokud u začíná symbolem b, vyplývá $x_1\overline{x_2}abu \in K$.

V každém případě K obsahuje řetězec s výskytem některého z faktorů aa, bb. Tudíž $K \not\subseteq L$ a dochází ke sporu. Neformálně každé w, které náleží do jazyka $L \in$ **UC**$_n$, musí obsahovat řetězec u maximální délky n. Tento řetězec může být vložen na jakékoliv místo v x_1x_2, přičemž výsledek stále náleží do jazyka L. V případě $L = \{ab\}^*$ tato vlastnost není dodržena. □

Teorém 5.4.9. **ES** *není uzavřená vůči průniku.*

Důkaz. Uvažujme ES,

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= \{a, \overline{a}\},\\ E &= \{\overline{a}a, a\overline{a}\},\\ R &= \{\overline{a}a\}^*\{a\overline{a}\}.\end{aligned}$$

Pro každé $d \geq 1$ existuje právě jeden řetězec v délky d, který náleží R. Podle lemmatu 5.4.6

$$L = \bigcup_{d\geq 1} K_d,$$

kde $K_1 = \varepsilon \leftarrow a\overline{a}$ a $K_{i+1} = K_i \leftarrow \overline{a}a$ pro $i \geq 1$. Bude ukázáno, že

$$\mathcal{D} \cap L = \{a\overline{a}\}^*,$$

kde $\mathcal{D} \in$ **ES** je Dyckův jazyk z příkladu 5.4.1 a jazyk $\{a\overline{a}\}^*$ nenáleží **ES** viz lemma 5.4.8. Prokázání $\{a\overline{a}\}^* \subseteq \mathcal{D} \cap L$ je snadné. Co se týče $\mathcal{D} \cap L \subseteq \{a\overline{a}\}^*$, vyplývá, že

$$\mathcal{D} \cap L = \mathcal{D} \cap \bigcup_{d\geq 1} K_d = \bigcup_{d\geq 1} (\mathcal{D} \cap K_d),$$

což je dostačující pro ověření, že $\mathcal{D} \cap K_d \subseteq \{a\overline{a}\}^*$ pro každé $d \geq 1$. Pro případ $d = 1$, $K_1 = \{a\overline{a}\}$. Pro pokračování dle indukce bude stanoveno $d \geq 2$. Pro každý řetězec $w \in \mathcal{D} \cap K_d$, $w = u_1\overline{a}au_2$ pro $u_1u_2 \in K_{d-1}$. Z $\mathcal{D} = \varepsilon \leftarrow^* a\overline{a}$ vyplývá, že $u_1 \in \mathcal{D}a\mathcal{D}$, $u_2 \in \mathcal{D}\overline{a}\mathcal{D}$, a tudíž $u_1u_2 \in \mathcal{D}$. Podle indukčního předpokladu, $u_1u_2 \in \{a\overline{a}\}^*$. Tudíž $w \in \{a\overline{a}\}^*\{\overline{a}a\}\{a\overline{a}\}^*$ nebo $w \in \{a\overline{a}\}^*\{a\}\{\overline{a}a\}\{\overline{a}\}\{a\overline{a}\}^*$. První případ implikuje $w \notin \mathcal{D}$ a dochází ke sporu. Pro druhý platí $w \in \{a\overline{a}\}^*$. □

Teorém 5.4.10. ES *není uzavřená vůči průniku s regulárními jazyky.*

Důkaz. Uvažujme jazyk $J = \{a, b\}^*$, který může být přijat například následujícím ES:

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= \{a, b\},\\ E &= \{a, b\},\\ R &= \{a, b\}^*.\end{aligned}$$

Dále uvažujme regulární jazyk $K = \{a\}^*\{b\}^*$. Poněvadž $J \cap K = K$, tento teorém vyplývá z lemmatu 5.2.3. □

Lemma 5.4.11. $\{a\}^*\{b\}^* \cup \{b\}^*\{a\}^* \notin$ **ES**.

Důkaz. Důkaz lze provést analogicky k lemmatu 5.2.3. □

Teorém 5.4.12. ES *není uzavřená vůči substituci.*

Důkaz. Uvažujme jazyk $K = \{ab, ba\}$, který je přijímaný následujícím ES:

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= \{a, b\},\\ E &= \{a, b\},\\ R &= \{ab\}.\end{aligned}$$

Definujme substituci σ z $\{a, b\}^*$ na $2^{\{a,b\}^*}$, přičemž $\sigma(a) = \{a\}^*$ a $\sigma(b) = \{b\}^*$. Zároveň $\sigma(a)$ a $\sigma(b)$ mohou být přijímány ES. Nicméně $\sigma(K)$ nemůže být přijímána žádným ES (viz lemma 5.4.11). □

Poněvadž substituce σ v důkazu teorému 5.4.12 je regulární, je získán následující důsledek.

Důsledek 5.4.12.1. ES *není uzavřená vůči regulární substituci.*

Teorém 5.4.13. ES *není uzavřená vůči*

(i) *konečné substituci,*

(ii) *homomorfismu bez ε,*

(iii) *obecnému homomorfismu.*

Důkaz. Pro homomorfismus bez ε uvažujme

$$\varphi : \{a\}^* \to \{a, b\}^*$$

s $\varphi(a) = ab$. Platí, že $L = \{a\}^* \in$ **ES** a $\varphi(L) = \{ab\}^* \notin$ **ES** (viz lemma 5.4.8). Jednoduše φ je také obecný homomorfismus a konečná substituce. □

Teorém 5.4.14. ES *není uzavřená vůči*

(i) *Kleeneho hvězdičce,*

(ii) *Kleeneho plus.*

Důkaz. Platí, že $\{ab\} \in$ **ES** a $\{ab\}^* \notin$ **ES** (viz lemma 5.4.8). Lze odvodit, že $\{ab\}^+ \notin$ **ES**, přičemž důkaz je možné provést analogicky k lemmatu 5.4.8. □

Teorém 5.4.15. ES *není uzavřená vůči inverznímu homomorfismu.*

Důkaz. Uvažujme následující ES

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= \{a_1, \overline{a_1}, a_2, \overline{a_2}\},\\ E &= \{a_1\overline{a_1}, \overline{a_1}a_1, a_2\overline{a_2}\},\\ R &= \{a_1\overline{a_1}, a_2\overline{a_2}\}^*\{\overline{a_1}a_1\}.\end{aligned}$$

Nechť $L = L(E, R)$. Lze pozorovat, že $L = \mathcal{D}_2\overline{a_1}\mathcal{D}_2a_1\mathcal{D}_2$, kde $\mathcal{D}_2$ je Dyckův jazyk se dvěma typy závorek, a to $a_1\overline{a_1}$ a $a_2\overline{a_2}$. Podle příkladu 5.4.1, $\mathcal{D}_2 \in$ **ES**. Nechť $\varphi : \{a, b\}^* \to \Sigma^*$ je definováno jako

$$\begin{aligned}\varphi(a) &= \overline{a_1}a_2,\\ \varphi(b) &= \overline{a_2}a_1\end{aligned}$$

a předpokládejme, že

$$\varphi^{-1}(L) = \{ab\}^*,$$

přičemž se jedná o jazyk, který není přijímaný žádným ES (viz lemma 5.4.8).

Pro každé $v = (ab)^i$ s $i \geq 0$ platí, že $\varphi(v) = \overline{a_1}(a_2\overline{a_2})^{i-1}a_1$, pokud $i \geq 1$ a $\varphi(v) = \varepsilon$, pokud $i = 0$. V obou případech $\varphi(v) \in L$, a tudíž $v \in \varphi^{-1}(L)$.

Dále uvažujme jakýkoliv řetězec $v \in \varphi^{-1}(L)$ a stanovme $w \in L$ s $\varphi(v) = w$. Jelikož $w \in L$, platí $w = u_1u_2u_3$ pro $u_1, u_3 \in \mathcal{D}_2$ a $u_2 \in \overline{a_1}\mathcal{D}_2a_1$. Jakmile $w \in range(\varphi)$, w začíná $\overline{a_1}$ nebo $\overline{a_2}$ a končí a_1 nebo a_2. Protože $u_1 \in \mathcal{D}_2$ nemůže začínat $\overline{a_1}$ ani $\overline{a_2}$ a $u_3 \in \mathcal{D}_2$ nemůže končit a_1 ani a_2, vyplývá, že $u_1 = u_3 = \varepsilon$ a $w \in \overline{a_1}\mathcal{D}_2a_1$. Označme $v = x_1x_2...x_m$ pro $x_1, x_2, ..., x_m \in \{a, b\}$. Jelikož $w \in \overline{a_1}\mathcal{D}_2a_1$, $x_1 = a$, $x_m = b$ a

$$w = \overline{a_1}a_2\varphi(x_2)...\varphi(x_{m-1})\overline{a_2}a_1,$$

kde

$$a_2\varphi(x_2)...\varphi(x_{m-1})\overline{a_2} \in \mathcal{D}_2.$$

Žádný z faktorů $a_2\overline{a_1}$ a $a_1\overline{a_2}$ se nemůže vyskytovat v $\mathcal{D}_2$. Z toho vyplývá, že pro každé $i = 2, ..., m-2$ platí, že

$$x_i = b \iff x_{i+1} = a,$$

což dohromady s $x_1 = a$ a $x_m = b$ implikuje, že $v \in \{ab\}^*$. □

Oproti původnímu znění poslední části důkazu v [27] byla v teorému 5.4.15 opravena chyba s hodnotami indexů pro x, podle kterých je určeno, kdy dané x zastupuje symbol a nebo symbol b.

Teorém 5.4.16. ES *není uzavřená vůči zamíchání.*

Důkaz. Uvažujme $\Sigma = \{a_1, \overline{a_1}, a_2, \overline{a_2}\}$ a uvažujme Dyckův jazyk $L = \mathcal{D}_2 \in \mathbf{ES}$ nad Σ. Je požadováno, aby $\mathcal{D}_2 \sqcup\!\sqcup \mathcal{D}_2 \notin \mathbf{ES}$. Podle lemmatu 5.4.7 předpokládáme spor, že $\mathcal{D}_2 \sqcup\!\sqcup \mathcal{D}_2 \in UC_n$ pro $n \geq 1$. Dále je značeno $w = a_1^n a_2^n \overline{a_1}^n \overline{a_2}^n$. Řetězec w náleží kompozici K stupně n mající formu $K = K' \leftarrow u$, tedy $w = x_1 u x_2$ pro $x_1 x_2 \in K'$.

Přirozeně existuje $y \in \{a_1, a_2\}$ takové, že je splněn alespoň jeden z následujících předpokladů:

(i) Předpokládejme, že u obsahuje y. Jestliže $|u| \leq n$, nemůže obsahovat $\overline{y}$. Řetězec $x_1 x_2 u$ náleží K, ovšem obsahuje výskyt y a zároveň neobsahuje výskyt $\overline{y}$ napravo od y, tedy nenáleží $\mathcal{D}_2 \sqcup\!\sqcup \mathcal{D}_2$.

(ii) Předpokládejme, že u obsahuje $\overline{y}$. Jestliže $|u| \leq n$, nemůže obsahovat y. Řetězec $u x_1 x_2$ náleží K, ovšem obsahuje výskyt $\overline{y}$ a zároveň neobsahuje výskyt y nalevo od $\overline{y}$, tedy nenáleží $\mathcal{D}_2 \sqcup\!\sqcup \mathcal{D}_2$.

□

Teorém 5.4.17. ES *není uzavřená vůči reverzaci.*

Důkaz. Lze provést sporem. Pro libovolný ES, $ES = (\Sigma, E, R)$, definujme $rev(ES)$ následovně:

$$rev(ES) = (\Sigma', E', R'),$$

kde

$$\begin{aligned} \Sigma' &= \Sigma, \\ E' &= E^R, \\ R' &= R^R. \end{aligned}$$

Jednoduše, $rev(rev(ES)) = ES$ pro každý ES. Předpokládáme, že

$$L(E', R') = L(E, R)^R,$$

přičemž $L(E, R)^R$ je vždy jazykem přijímaným nějakým ES. Dále uvažujme následující ES:

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned} \Sigma &= \{a, b, c\}, \\ E &= \{a, bc\}, \\ R &= \{bca\}, \end{aligned}$$

který přijímá jazyk

$$L(E, R) = \{abc, bca\}.$$

Reverzací daného jazyka je jazyk

$$L(E,R)^R = \{cba, acb\}.$$

Uvažujme řetězec $w = cab$, $w \notin L(E,R)^R$. Dále uvažujme

$$rev(ES) = (\Sigma', E', R'),$$

kde

$$\begin{aligned}\Sigma' &= \Sigma = \{a,b,c\},\\ E' &= E^R = \{a, cb\},\\ R' &= R^R = \{acb\},\end{aligned}$$

přičemž $w \in L(E',R')$ dle definice 5.1.3. To implikuje, že $w \in L(E,R)^R$, a tudíž dochází ke sporu vůči předpokladu, že $L(E',R') = L(E,R)^R$. □

Shrnutí uzávěrových vlastností rodiny **ES** je uvedeno v tabulce 5.1.

	ES	**GJFA**	**REG**
Značení konce	−	−	+
Konkatenace	−	−	+
Doplněk	−	−	+
Sjednocení	−	+	+
Průnik	−	−	+
Průnik s regulárními jazyky	−	−	+
Substituce	−	−	+
Regulární substituce	−	−	+
Konečná substituce	−	−	+
Homomorfismus	−	−	+
Homomorfismus bez ε	−	−	+
Inverzní homomorfismus	−	−	+
Kleeneho hvězdička	−	−	+
Kleeneho plus	−	−	+
Zamíchání	−	−	+
Reverzace	−	+	+

Tabulka 5.1: Uzávěrové vlastnosti jazykové rodiny přijímané vymazávacími systémy (**ES**). **GJFA** značí rodinu jazyků přijímaných obecnými skákajícími konečnými automaty a **REG** značí rodinu regulárních jazyků. Symbol „+“ značí, že rodina jazyků je uzavřena vůči operaci. Symbol „−“ naopak značí, že rodina jazyků není uzavřena vůči operaci.

5.5 Operace zamíchání

Hlavním cílem této sekce je představení vztahu rodiny jazyků přijímaných vymazávacími systémy (dále jen **ES**) s rodinou jazyků generovaných SHUF výrazy, dále označované jako **SHUF**.

Následující lemmata a teorémy dokazují, že také **ES** a **SHUF** jsou neporovnatelné shodně s rodinou jazyků přijímaných obecnými skákajícími konečnými automaty (**GJFA**) a **SHUF**.

Vzhledem k paralele vymazávacích systémů (dále jen ES) a obecných skákajících konečných automatů (dále jen GJFA) lze použít lemmata, teorém a příklad uvedené v [11].

Příklad 5.5.1. Na základě vlastností GJFA v [11] lze pro ES uvažovat totožný následující případ. Nechť

$$ES = (\Sigma, E, R)$$

je ES, kde

$$\begin{aligned}\Sigma &= \{a, b, c, d\},\\ E &= \{ab, cd\},\\ R &= \{ab, cd\}^*,\end{aligned}$$

který přijímá jazyk $L(E, R)$, přičemž jej lze přirozeně převést na výraz zamíchání $S = (ab{+}cd)^{\sqcup\!\sqcup,*}$. Lze jednoduše ověřit, že každý řetězec, který je přijímaný ES, může být zároveň vygenerován S, ale jelikož $acbd \in (L(S) \backslash L(E, R))$, potom vyplývá, že $L(E, R) \subsetneq L(S)$.

V následující části lze pozorovat, že jazykové rodiny **ES** a **SHUF** jsou neporovnatelné.

Lemma 5.5.1. *Nechť $ES = (\Sigma, E, R)$ je ES, kde*

$$\begin{aligned}\Sigma &= \{a, b, c, d\},\\ E &= \{ab, cd\},\\ R &= \{ab, cd\}^*.\end{aligned}$$

Pak $L(E, R)$ není SHUF jazyk.

Důkaz. Lze provést sporem. Nechť S je SHUF výraz, přičemž $L(S) = L(E, R)$. Jelikož počet výskytů a a d je neomezený v řetězcích náležejících $L(E, R)$, jeden z následujících dvou případů musí platit:

1. S obsahuje podvýraz $(T)^{\sqcup\!\sqcup,*}$, přičemž existuje $w \in L(T)$ s $|w|_a \geq 1$ a $|w|_d \geq 1$.
2. S obsahuje podvýraz $T_1 \sqcup\!\sqcup\, T_2$, přičemž existuje $w \in L(T_1)$ s $|w|_a \geq 1$ a $w' \in L(T_2)$ s $|w|_d \geq 1$.

Oba případy implikují, že $L(S)$ obsahuje řetězec s faktorem ad. Dochází tedy ke sporu, jelikož takové řetězce nenáleží do $L(E, R)$. □

Lemma 5.5.2. *Nechť $L = L(ac \sqcup\!\sqcup (bd)^{\sqcup\!\sqcup,*})$, pak $L \notin$ **ES**.*

Důkaz. Lze provést sporem. Předpokládejme, že jazyk L je přijímán ES značeným ES. Nechť n je větší než maximální délka řetězce $u \in E$ v ES a nechť $w = ab^n cd^n$. Výpočet ES, který bude přijímat w, použije právě jeden vymazávací řetězec u, který obsahuje c.

- Pokud $u = b^i cd^j$ pro $i, j \geq 0$, všechny dřívější vymazávané řetězce obsahují pouze faktory, které jsou zcela obsaženy v prefixu ab^{n-i} nebo v sufixu d^{n-j}, přičemž $w = ab^{n-i}(b^i cd^j)d^{n-j}$. Je implikováno, že za použití stejné sekvence vymazávacích řetězců ES může přijímat $w' = b^i cd^j ab^{n-i}d^{n-j}$.

- V opačném případě $u = ab^r cd^s$ pro $r, s \geq 0$, přičemž obsahuje a i c. Dle výběru n, dřívější vymazaný řetězec b^k s $k > 0$ byl použit. Nicméně to implikuje, že také $w'' = ab^{n-k}cd^n b^k$ je přijímaný ES.

Případ $w' \in L(E, R)$ porušuje podmínku, že symbol a předchází symbolu c v řetězcích z L, zatímco případ $w'' \in L(E, R)$ odporuje faktu, že řetězce v L nekončí symbolem b. □

Teorém 5.5.3. **ES** *a* **SHUF** *jsou neporovnatelné.*

Důkaz. **SHUF** ∩ **ES** ⊊ **ES** vyplývá z definice a lemmatu 5.5.1. **SHUF** ∩ **ES** ⊊ **SHUF** vyplývá z definice a lemmatu 5.5.2. □

5.6 Shrnutí dosažených výsledků

V této kapitole byly zavedeny vymazávací systémy jako nové formální modely v oblasti formální teorie jazyků, které reprezentují alternativní volbu k obecným skákajícím konečným automatům.

Kromě samotného formalizování nového systému pomocí definic a následně demonstrování jeho funkcionality na několika ukázkových příkladech, byly v této kapitole studovány a prokázány vlastnosti rodin jazyků přijímaných těmito systémy se známými jazyky dle Chomského hierarchie. V této oblasti vlastnosti vymazávacích systémů obsahují hlavní rozdíl oproti zmíněným obecným skákajícím konečným automatům, a to neporovnatelnost rodiny konečných jazyků (**FIN**) a rodiny jazyků přijímaných vymazávacími systémy (**ES**). Obecně lze uvažovat, že důvod, proč obecné skákající konečné automaty mohou přijímat jakýkoliv konečný jazyk, je především určen přechody mezi stavy, kdy lze přesně určit, jaký řetězec má být odstraněn ze vstupní pásky. Na druhou stranu vymazávací systém má určenou celkovou posloupnost symbolů řídícím regulárním jazykem, ovšem způsob, jakým budou vymazávány jednotlivé řetězce z množiny vymazávacích řetězců tak, aby řetězec vzniklý jejich konkatenací náležel do regulárního jazyka (a odpovídal jedné z možných posloupností symbolů), již nelze dále řídit. Není tedy možné určit, jaký vymazávací řetězec, ve kterém kroku vymazávání bude použit.

Následně jako velmi důležitý a prakticky využitelný znak těchto systémů jsou jeho vztahy s Dyckovými a polo-Dyckovými jazyky, jenž mohou velmi snadno nalézt reálné využití.

Významný rozdíl vymazávacích systémů oproti uvedené variantě skákajících automatů lze zpozorovat v uzávěrových vlastnostech jejich rodin jazyků. Ze všech vlastností, které si tato práce klade za cíl prozkoumat, se tyto dva jazykové modely shodují až na uzavřenost jejich rodin jazyků vůči operaci sjednocení a reverzaci. Na rozdíl od obecných skákajících konečných automatů rodina jazyků přijímaných vymazávacími systémy není uzavřena vůči těmto operacím.

V prvním případě pro operaci sjednocení, obecné skákající konečné automaty využívají svého rysu v podobě pravidel obsahujících označení stavů, kdy každé pravidlo přesně určuje, který řetězec může být vymazán ze vstupní pásky a odlišuje od sebe stavy jednotlivých automatů, které byly sjednoceny do celkového nového automatu (viz [20]). Toto ovšem

neplatí u vymazávacích systémů využívajících řídící regulární jazyk a vymazávací řetězce. Z tohoto důvodu totiž nelze určit, který vymazávací řetězec bude použit pro vymazání, pokud množina vymazávacích řetězců obsahuje také podřetězce některých vymazávacích řetězců s délkou větší než dva symboly, přičemž konkatenace podřetězců odpovídá původnímu řetězci. Pokud taková situace nastane pro nový vymazávací systém, který vznikl sjednocením dvou rozdílných vymazávacích systémů, na základě této skutečnosti přijímaný jazyk bude odlišný od jazyka definovaného sjednocením jazyků přijímaných jednotlivými vymazávacími systémy.

V druhém případě s operací reverzace, odlišností obecných skákajících konečných automatů oproti vymazávacím systémům je, že mohou při reverzaci automatu využívat pouze reverzace řetězců reprezentujících značení přechodů (viz [27]). Naproti tomu vymazávací systémy při této operaci, kromě reverzace řetězců v množině vymazávacích řetězců, musí provést reverzaci celého regulárního jazyka. Tudíž dochází také ke změně pořadí vymazávání řetězců dle řídícího jazyka, z čehož vyplývá, že jazyk přijímaný reverzovaným systémem je odlišný od reverzace jazyka přijímaného původním vymazávacím systémem. Ovšem kromě těchto dvou rozdílných rysů se vlastnosti obou jazykových modelů ve zkoumaných oblastech shodují.

Současně s aktuálním výzkumem skákajících automatů byly na vymazávací systém aplikovány důkazy ohledně vztahů jejich jazykové rodiny s rodinou jazyků generovaných výrazy zamíchání. V tomto ohledu se nový formální systém ztotožňuje s obecnými skákajícími konečnými automaty.

Kapitola 6

Aplikace vymazávacích systémů

Druhou velmi důležitou součástí této práce je studium užití a aplikací zavedeného formálního systému, který byl představen v kapitole 5, na reálné procesy.

Tato práce si klade za cíl využít nový formální model, nazývaný *vymazávací systém*, v oblasti bioinformatiky pro molekulární biologii, dále v textových editorech a v poslední řadě uvažuje o jeho aplikaci na problémy v kompozičním šachu.

Kapitola dle uvedených oblastí užití rozděluje navržené aplikace do tří sekcí. Jako první, sekce 6.1 navrhuje tři aplikace vymazávacího systému pro zpracování sekvencí, které se využívají při buněčných procesech. Dále sekce 6.2 na základě získaných teoretických poznatků studuje aplikaci vymazávacích systémů na úlohy používané v textových editorech. V poslední řadě sekce 6.3 představuje trochu netradiční aplikaci formálního systému na ověřování správnosti řešení problému n dam z oblasti kompozičního šachu.

6.1 Aplikace v oblasti molekulární biologie

Formální modely dle svého návrhu mohou reprezentovat reálné buněčné procesy, viz například [21]. Výzkum aplikace nových formálních modelů v oblasti biologie je stále velmi aktuální. To platí také pro dříve představené *skákající konečné automaty*. Na základě podobných vlastností uvedených automatů a nově zavedeného vymazávacího systému lze uvažovat jeho užití v bioinformatice.

Tato podkapitola představuje tři aplikace vymazávacího systému v oblasti bioinformatiky pro molekulární biologii.

Nejprve podsekce 6.1.1 uvádí základní pojmy a poznatky pro molekulární biologii. Poté podsekce 6.1.2 zavádí první aplikaci vymazávacího systému pro vyhledávání podsekvencí v molekulárních sekvencích. Následně podsekce 6.1.3 na základě získaných poznatků o aplikaci vymazávacích systémů pro vyhledávání modifikuje navrženou aplikaci také pro vyhledávání v sekvencích proteinů a samotných aminokyselin, které je tvoří. Jako poslední užití vymazávacích systémů v této oblasti aplikace představená v podsekci 6.1.4 studuje vlastnosti sekundární struktury RNA.

6.1.1 Úvod do molekulární biologie

Cílem této podsekce je zasvětit čtenáře do základů molekulární biologie nezbytných pro pochopení následující části textu. Postupně jsou představeny základní pojmy, jako *protein*, *DNA* či *RNA*. Tato podkapitola také popisuje proces *transkripce* a *translace*. Jako poslední uvádí takzvanou *sekundární strukturu RNA*.

Struktura proteinů

Tato podkapitola byla převzata z [1]. *Proteiny* čili bílkoviny tvoří většinu suché hmotnosti buňky. Nejsou však pouhými stavebními kameny, z nichž je buňka vytvořena, ale obstarávají většinu buněčných funkcí.

Proteiny jsou tvořeny sekvencí *aminokyselin* (nazývána taktéž jako *aminokyselinová sekvence*), přičemž kovalentní vazba mezi dvěma sousedními aminokyselinami se nazývá *peptidová vazba.* Aminokyselinová sekvence určuje, jak bude daný protein sbalen do funkční informace. V proteinech se běžně nachází jednadvacet druhů aminokyselin, z nichž každá má jiné vlastnosti. Tato skupina dvaceti jedna aminokyselin se vyskytuje napříč všemi proteiny od bakterií přes rostliny až k živočichům.

Molekula proteinu je tedy tvořena řetězcem těchto aminokyselin spojených se svými sousedy kovalentní peptidovou vazbou. Každý typ proteinu má jedinečné pořadí aminokyselin, které je u všech molekul tohoto proteinu stejné.

Struktura DNA

DNA (*deoxyribonukleová kyselina*) je molekula, která hraje ústřední roli v mikrobiologii. V biochemickém světě velkých a malých molekul, polymerů a monomerů je DNA polymer, který je spojen dohromady z monomerů nazývaných *deoxyribonukleotidy.* DNA je klíčovou molekulou v živých buňkách a má strukturu, která podporuje dvě nejdůležitější funkce DNA: kódování pro produkci proteinů a vlastní replikaci, takže přesná kopie je předána do buněk potomstva, a tudíž zastává funkci nosiče genetické informace [21].

Jak již bylo řečeno, monomery používané pro konstrukci DNA jsou deoxyribonukleotidy, kde každý deoxyribonukleotid se skládá ze tří složek: pětiuhlíkového sacharidu (cukru), jedné či více fosfátových skupin a dusíkaté báze. Pro zjednodušení názvosloví se používá termín *nukleotidy* namísto deoxirybonukleotidů z důvodu, že oním sacharidem je vždy *deoxyribósa* [21].

Molekula DNA se skládá ze dvou dlouhých polynukleotidových vláken složených ze čtyř typů nukleotidových podjednotek. Obě tato vlákna jsou nazývána jako *řetězce DNA* nebo *vlákna DNA* a jsou vzájemně spojena *vodíkovými můstky* mezi bázemi nukleotidů. V případě nukleotidů bází může být *adenin (A)*, *cytosin (C)*, *guanin (G)* nebo *thymin (T).* Nukleotidy jsou spojeny v řetězec kovalentními vazbami mezi sacharidy a fosfáty, které tak tvoří „kostru“ ze dvou střídajících se částí (cukr-fosfát-cukr-fosfát...). Protože se podjednotky DNA liší pouze bázemi, symboly A, C, G a T jsou obvykle používány k zaznamenávání čtyř různých nukleotidů — bází s navázaným sacharidem a fosfátovou skupinou [1].

Způsob, jakým jsou nukleotidy spojeny, dává řetězci DNA polaritu. Pokud si představíme na každém nukleotidu výčnělek místo fosfátu, a jamku, do které výčnělek zapadá, místo cukru, mohou se nukleotidy spojovat jen jedním, a to orientovaným způsobem. Navíc jsou oba konce řetězce snadno odlišitelné, neboť na jednom konci je výčnělek a na druhém jamka. Díky této polaritě řetězce DNA je konvenčně jeden konec označován jako $3'$ (končí -OH skupinou sacharidu) a druhý jako $5'$ (končí fosfátovou skupinou) [1].

Oba polynukleotidové řetězce jsou v *dvoušroubovici* DNA drženy pohromadě vodíkovými můstky mezi bázemi různých řetězců. Báze nukleotidů se dělí na dva druhy: *puriny* (adenin a guanin) a *pyrimidiny* (cytosin a thymin). Ty se párují tak, že adenin se váže s thyminem a guanin s cytosinem a dochází ke *komplementárnímu párování* [1].

Báze se mohou spolu párovat ve dvoušroubovici jen v případě, že jsou oba řetězce vůči sobě *antiparalelní*, to znamená, že polarita jednoho řetězce je opačná k polaritě druhého řetězce DNA. Z párování bází vyplývá, že první řetězec DNA molekuly obsahuje sekvenci

nukleotidů, která je přesně *komplementární* k nukleotidové sekvenci druhého, partnerského řetězce [1].

Struktura RNA

RNA (*ribonukleová kyselina*) je polymer, který má zásadní význam pro živé buňky. Jeho struktura je velmi blízká struktuře DNA. Je tvořen z monomerů zvaných *ribonukleotidy*. Ribonukleotid se liší od deoxyribonukleotidu dvěma způsoby:

(i) Obsahuje *ribózový cukr* namísto deoxyribózového cukru.

(ii) Thyminová báze je nahrazena v ribonukleotidu *uracilovou* bází (U). Čtyři možné báze jsou tedy A, U, C a G [21].

Zatímco se DNA v buňce vyskytuje vždy jako dvoušroubovice (s výjimkou některých virů), RNA je jednořetězcová molekula [1].

Transkripce a translace

Obsah této podkapitoly vychází z [1]. Ještě před identifikací genetického kódu bylo známo, že DNA nějakým způsobem řídí vznik proteinů. Proteiny jsou jednou ze základních složek buněk, u kterých určují nejen jejich strukturu, ale i jejich funkce.

DNA neřídí syntézu bílkovin, ovšem jestliže buňka potřebuje nějaký konkrétní protein, je nukleotidová sekvence v patřičné oblasti dlouhé molekuly DNA nejprve zkopírována do RNA. Tato RNA se přímo využívá jako templát pro tvorbu proteinů. Genetická informace je tedy předávána z DNA do RNA a následně z RNA do proteinu. Přepis DNA do RNA je známý také pod pojmem *transkripce* a využití RNA k syntéze proteinů jako *translace*.

Transkripce (přepis) probíhá tak, že jednotlivé části DNA (geny) jsou přepisovány do RNA a dochází k jejich kopii. Další proces zpracování RNA se liší u *eukaryotních* (živočišných a rostlinných) buněk a u buněk *prokaryotních* (bakteriálních). Eukaryotní RNA podléhají *posttranskripčním úpravám* před samotným překladem na sekvenci aminokyselin. Nejprve transkripty podléhají dvěma základním úpravám: *přidání čepičky* na $5'$ konci RNA a takzvané *polyadenylaci*, při které je přidán *poly(A)-konec* na $3'$ konec RNA. Po zmíněných úpravách jsou eukaryotní geny přerušovány nekódujícími sekvencemi. Tyto úseky se nazývají *introny*. Úseky kódujících sekvencí se nazývají *exony*. Další úpravou transkriptu je *sestřih RNA*, při kterém dochází k odstranění intronů. Jakmile se tato modifikace transkriptu provede, může být následně překládán na proteiny. RNA, ať již eukaryotní či prokaryotní, která bude překládána na proteiny, je nazývána také jako *mediátorová* RNA (*mRNA*).

Při translaci (překladu) RNA dochází k převodu jednotlivých trojic nukleotidů RNA na aminokyseliny. Začátek je rozpoznán pomocí start kodonu, který je označen trojicí nukleotidů AUG. Následující trojice nukleotidů jsou poté překládány na jednotlivé aminokyseliny dokud není nalezen stop kodon, který je označen jednou trojicí nukleotidů z následujících tří sekvencí, a to UUA, AAG nebo UGA. Protože RNA je lineární polymer složený ze čtyř podjednotek, spojením tří nukleotidů lze vytvořit $4 * 4 * 4 = 64$ kombinací: AAA, AUA, AUG a tak dále. V proteinech se však vyskytuje obvykle pouze jednadvacet aminokyselin. Skupina několika tripletů (trojic nukleotidů) se tedy překládá na stejnou aminokyselinu. Například pro aminokyselinu *Alanin*, která je značena symbolem A, to mohou být triplety GCA, GCC, GCG nebo GCU.

U eukaryot zjednodušeně translace probíhá tak, že start kodon AUG je vyhledáván od začátku mRNA s využitím přidané čepičky na $5'$ konci. Eukaryotní mRNA obvykle nesou informaci jen pro jediný protein. Výběr start-kodonu u prokaryot (bakterií) je odlišný. Bakteriální mRNA nemá na svém $5'$ konci čepičku. Namísto toho mají několik nukleotidů před kodonem AUG signální sekvenci značící počátek translace (za účelem navázání takzvaného *ribosomu*). Díky tomu může být počátek translace proveden i uprostřed mRNA a z jedné mRNA může být přeloženo několik proteinů.

Genetický kód, tedy samotné převody tripletů nukleotidů na aminokyseliny, je ve formě tabulky uveden v příloze C.

Sekundární struktura RNA

RNA se na základě párování bází může sbalit do různých tvarů podobně jako polypeptidový řetězec do konečného tvaru bílkoviny. V rámci jedné molekuly RNA se tedy mohou tvořit nukleotidové páry. I přes to, že je RNA jednořetězcová molekula, často obsahuje úseky, které se mohou párovat s komplementárními sekvencemi nacházejícími se na jiném místě téže molekuly [1]. Na základě toho RNA vytváří sekundární a terciární strukturu.

Se zaměřením pouze na sekundární strukturu jsou jednotlivé opakující se prvky klasifikovány do kategorií. Například se může jednat o *stonkové smyčky* (anglicky *stem-loops*), *pseudouzly* (anglicky *pseudoknots*) nebo o *vnitřní smyčky* (anglicky *internal loop*). Více viz například [15, 22].

Existují různé typy reprezentací sekundární struktury RNA pro algoritmické zpracování či vizualizaci. Jednou z nejrozšířenějších notací, kterou využívá také tato práce, je takzvaná *tečkovaná závorková notace* (anglicky *dot-bracket notation*), přičemž je využívána rozšířená verze této notace. To znamená, že kromě symbolů pro závorky „(“ a „)“ pro párové nukleotidy a symbolu „.“ pro nepárové nukleotidy, jsou povoleny také další páry závorek, jako {}, [], či <> a komplementární dvojice velkých a malých znaků anglické abecedy[1]. Díky tomu je možná také reprezentace složitějších elementů, jako jsou například zmíněné pseudouzly. Jelikož různé nástroje pro predikci sekundární struktury RNA uvažují i jiné symboly pro označení nepárového nukleotidu, souhrnně může být reprezentován symbolem „.“, „:“ nebo „|“.

6.1.2 Vyhledávání sekvencí v DNA, RNA a sekundární struktuře RNA

Na základě skákajících vlastností vymazávacích systémů lze předpokládat, že tyto vlastnosti jsou vhodné pro vyhledávání více jak jednoho symbolu či podřetězce ve vstupním řetězci. Vlastnosti vymazávacího systému, který pracuje také s vymazávacími řetězci, jenž obsahují dva a více symbolů, ovšem obsahují jistá úskalí, která jsou dále popsána.

Nechť $ES = (\Sigma, E, R)$ je vymazávací systém. Dále jsou uvažovány dva rozdílné řetězce $u, v \in E$, přičemž $|u| \geq 2$, $|v| \geq 2$, a také řetězec $w \in (\Sigma - \{\#\})^*$, který je přečten ze vstupní pásky. Nelze s jistotou tvrdit, že pokud byl nejprve například vymazán řetězec u z řetězce w a následně řetězec v, že druhý řetězec nevznikl konkatenací dvou řetězců, které se před vymazáním nacházely napravo a nalevo od vymazávaného řetězce u.

Pokud by ovšem bylo uvažováno, že ještě před prvním vymazáním vymazávacího řetězce ze vstupního řetězce w by byly všechny nepřekrývající se výskyty označeny speciálním symbolem, například symbolem $\sim$, poté je možné prohlásit, že pokud se v řetězci nachází všechny vyhledávané podřetězce začínající symbolem $\sim$, daný vstupní řetězec obsahuje

[1] https://www.tbi.univie.ac.at/RNA/ViennaRNA/doc/html/rna_structure_notations.html

alespoň jeden výskyt každého vyhledávaného podřetězce, přičemž se zároveň nepřekrývá s jiným nalezeným podřetězcem. Označení vyhledávaných podřetězců ve vstupním řetězci w je možné pomocí konečného převodníku.

Na základě definice konečného převodníku (viz definice 3.2.1) pro vstupní řetězec, výstupem příslušného převodníku může být více jak jednoprvková množina výstupních řetězců. Tato situace nastává i v případě označování vyhledávaných podřetězců, jelikož nedeterminismus převodníku způsobí, že se vygeneruje více výstupů s tím, zda případný nalezený podřetězec je označen či nikoliv. Je tomu tak i v případě překrývajících se výskytů vyhledávaných řetězců, kdy může být označen pouze jeden z podřetězců, a tudíž dokonce jenom pro tento konkrétní případ budou vytvořeny dva výstupy. Lze s jistotou prohlásit, že pokud se všechny vyhledávané sekvence nachází ve vstupním řetězci, tak, že mohou být vyjmuty, aniž by se překrývaly, pak jeden z výstupních řetězců správně sestaveného převodníku obsahuje patřičné označení podřetězců.

Vyhledávání podsekvencí přirozeně může mít praktické využití v oblasti molekulární biologie při práci se sekvencemi RNA, DNA či se sekvencí sekundární struktury RNA. Pokud dané sekvence obsahují jisté podsekvence, které se zároveň nepřekrývají, mohou tím splňovat určité vlastnosti, které lze následně podrobněji zkoumat.

Popis algoritmu vyhledávání sekvencí v RNA

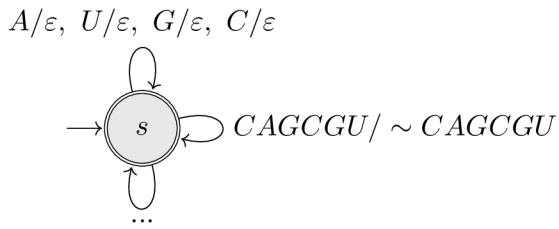
Následně bude popsán celý algoritmus včetně vytvořeného konečného převodníku a vymazávacího systému. Jelikož postup pro odlišné vstupní struktury je velmi podobný, dále jako vstup bude uvažována sekvence RNA ve směru $5' \rightarrow 3'$. Aplikace pro DNA a sekundární strukturu by se především lišila v použité abecedě ($\Sigma_{DNA} = \{A, T, G, C\}$ pro DNA a $\Sigma_{RNA\_SEC} = \{., (,), [,], \{, \}, <, >, A, ..., Z, a, ..., z\}$ pro sekundární strukturu RNA při využití tečkované-závorkové notace a za použití symbolu „.“ pro nepárový znak.). Při aplikaci na vstupní strukturu DNA lze uvažovat, zda se mají vyhledávané podsekvence vyhledávat na obou vláknech DNA, či pouze na jednom. Navíc pro druhé vlákno ve směru $3' \rightarrow 5'$ lze uvažovat, zda má být vyhledávání prováděno v totožném směru či v opačném směru. Tuto veškerou funkcionalitu nabízí implementace dané aplikace.

Dále bude uvažováno, že jako vstupní řetězce w bude sekvence RNA ve směru $5' \rightarrow 3'$, přičemž $w \in \Sigma^*_{RNA}, \Sigma_{RNA} = \{A, U, G, C\}$. Nechť je požadováno, aby vstupní sekvence obsahovala například podsekvenci $CAGCGU$ a pak následně ještě několik dalších podsekvencí, které nyní pro demonstraci algoritmu nejsou přímo uvedeny. Je definován konečný převodník

$$T = (Q, \Sigma, \Delta, \sigma, s, F),$$

kde

$$\begin{aligned}
Q &= \{s\}, \\
\Sigma &= \{A, U, G, C\}, \\
\Delta &= \{A, U, G, C\} \cup \{\sim\}, \\
\sigma &= \{sA \rightarrow s\varepsilon, ..., sCAGCGU \rightarrow s \sim CAGCGU, ...\}, \\
F &= \{s\}.
\end{aligned}$$



Obrázek 6.1: Stavový diagram konečného převodníku T.

Lze si povšimnout, že jednotlivé vstupní nukleotidy, které se nenachází ve vyhledávané podsekvenci, se převádí na prázdný řetězec ε. To je prováděno za účelem, aby vymazávací systém nemusel ověřovat abecedu, nad kterou je definovaný vstupní řetězec, přičemž by docházelo k duplicitě kontroly (již se provádí pomocí konečného převodníku).

Tento převodník T, jehož stavový diagram je uveden na obrázku 6.1, označí všechny nepřekrývající se výskyty vyhledávaných řetězců ve w, přičemž ponechá pouze nalezené podsekvence, a vytvoří množinu výstupních řetězců, která obsahuje všechny možnosti označení vyskytujících se podsekvencí, které jsou vyhledávány.

Při implementaci lze algoritmus optimalizovat tak, že výstupní řetězce, které neobsahují alespoň takový počet označených podsekvencí, jako je počet vyhledávaných sekvencí, nejsou uvažovány a jsou z množiny výstupních řetězců odstraněny. Pokud po tomto kroku je množina neprázdná, lze s využitím datové struktury seznam seřadit výstupní řetězce od nejvyššího počtu označených podsekvencí, přičemž je možné uvažovat, že s vyšším počtem označených řetězců se může zvyšovat pravděpodobnost nalezení všech vyhledávaných sekvencí. Pokud je jeden z řetězců přijat dále definovaným vymazávacím systémem, zbylé řetězce již nejsou ověřovány.

Následně jednotlivé výstupní řetězce jsou postupně řetězcem, který je přečten ze vstupní pásky pomocí vymazávacího systému

$$ES = (\Sigma, E, R),$$

kde

$$\begin{aligned}\Sigma &= \{\sim, A, U, G, C\},\\ E &= \{\sim CAGCGU, ...\},\\ R &= \{\sim CAGCGU\}^{+}....\end{aligned}$$

Je-li vstupní řetězec přijat definovaným vymazávacím systémem, pak vstupní řetězec obsahuje všechny vyhledávané podřetězce. V opačném případě, pokud se ve vstupním řetězci nevyskytují všechny požadované podřetězce alespoň jednou, není řetězec přijat daným vymazávacím systémem.

Pokud je alespoň jeden výstupní řetězec z množiny výstupních řetězců konečného převodníku přijat definovaným vymazávacím systémem, pak vstupní sekvence RNA obsahuje všechny vyhledávané podsekvence, jejichž nalezené výskyty se nepřekrývají.

6.1.3 Vyhledávání proteinů a sekvencí aminokyselin proteinů

Na základě aplikace vymazávacího systému na vyhledávání sekvencí v oblasti molekulární biologie, kterou představila podsekce 6.1.2, lze uvažovat ještě další možné užití na bázi vyhledávání. Konkrétně nebyla zmíněna ještě sekvence, která v oblasti mikrobiologie zastupuje jednu z nejdůležitějších úloh, a to *sekvence aminokyselin*, jenž reprezentuje samotný *protein*.

Algoritmus pro vyhledávání proteinů a sekvencí aminokyselin proteinů

Dále bude popsán algoritmus převodu sekvence RNA (mRNA) na sekvence proteinů tvořených aminokyselinami a samotné vyhledávání celých proteinů či jejich jednotlivých podčástí (podsekvencí aminokyselin) pomocí vymazávacího systému.

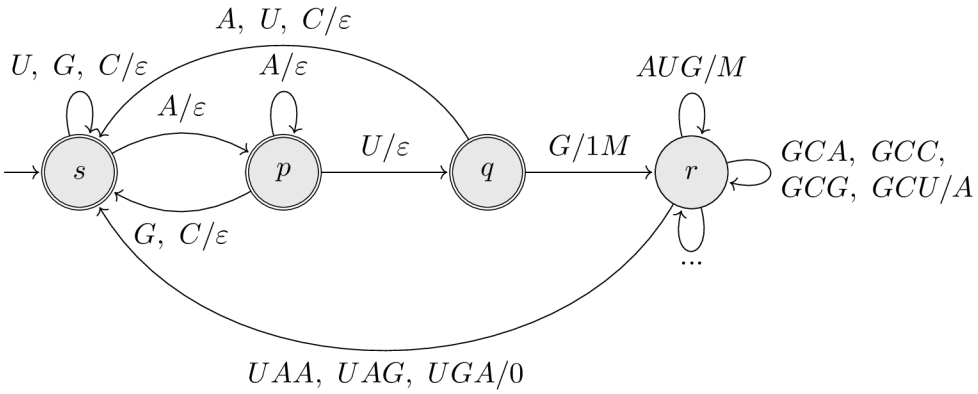
Nejprve je ovšem nutné zmínit důležitou vlastnost vstupních řetězců RNA, a to tu, že aplikace uvažuje pouze *prokaryotní RNA*. Důvody, proč aplikace nemůže pracovat s *eukaryotními RNA* jsou následující. Jak již bylo uvedeno v podsekci 6.1.1, eukaryotní RNA po *transkripci* podstupují ještě *posttranskripční úpravy*, včetně takzvaného *sestřihu*, jelikož eukaryotní RNA obsahuje také *nekódující sekvence*. Tudíž některé podsekvence RNA (takzvané *introny*) se při *translaci* již nevyskytují v této upravené RNA (mRNA). Navíc může docházet také k *alternativnímu sestřihu*, kdy z RNA může vzniknout pokaždé odlišná mRNA (viz [1]). Byla zde zmíněna pouze nejdůležitější úskalí, která jsou ale pro vyloučení aplikace algoritmu na eukaryotní RNA dostačující. Po rozpoznávání jednotlivých kódujících a nekódujících sekvencí je nutné se opřít o pravděpodobnostní modely, viz například [29] s využitím *skrytých Markovových modelů* (anglicky *hidden Markov models*, zkráceně známo jako HMM) nebo například [10] za použití *neuronových sítí* (anglicky *neural networks*, zkráceně známo pod NN).

Algoritmus aplikace je následující. Samotnou translaci sekvence mRNA, která je zadána ve směru $5' \to 3'$, na sekvence proteinů lze provést pomocí konečného převodníku. Nechť je definován konečný převodník

$$T_1 = (Q_1, \Sigma_1, \Delta_1, \sigma_1, s, F_1),$$

kde

$$\begin{aligned}
Q_1 &= \{s, p, q, r\}, \\
\Sigma_1 &= \{A, U, G, C\}, \\
\Delta_1 &= \{\varepsilon, 1, 0, A, C, ..., Y\}, \\
\sigma_1 &= \{sU \to s\varepsilon, sG \to s\varepsilon, sC \to s\varepsilon, sA \to p\varepsilon, \\
&\quad pA \to p\varepsilon, pG \to s\varepsilon, pC \to s\varepsilon, pU \to q\varepsilon, \\
&\quad qA \to s\varepsilon, qU \to s\varepsilon, qC \to s\varepsilon, qG \to r1M, \\
&\quad rAUG \to rM, rGCA \to rA, ..., \\
&\quad rUAA \to s0\$, rUAG \to s0, rUGA \to s0\$\$\}, \\
F_1 &= \{s, p, q\},
\end{aligned}$$



Obrázek 6.2: Stavový diagram konečného převodníku T_1.

který provede uvedenou translaci vstupní sekvence mRNA na sekvence proteinů. Stavový diagram T_1 je uveden na obrázku 6.2. Jednotlivé celé proteiny začínají symbolem 1 a jsou ukončeny symbolem 0. Tento konečný převodník je navržen tak, že jeho přechody neobsahují nedeterminismus v tom smyslu, že v každém stavu a pro každý vstupní řetězec při daném stavu, převodník může použít právě jedno pravidlo pro přechod do nového stavu. Tudíž množina výstupních řetězců obsahuje pouze jeden výstupní řetězec. Souhrnně přechody konečného převodníku T_1 dodržují genetický kód (viz příloha C). Dle tohoto genetického kódu převzatého z [1] se neuvažuje, že kodon zastoupený trojicí nukleotidů UGA slouží pro syntézu *selenocysteinu*, a tudíž zastává pouze úlohu terminačního kodonu pro základní variantu genetického kódu.

Při implementaci dané aplikace se uvažuje, že stav r v Q_1 pro T_1 může volitelně také náležet do množiny koncových stavů F_1. To znamená, že překlad na určitý protein započal, ovšem příslušná sekvence není ukončena některým z *terminačních kodonů*, čímž se umožňuje podobná funkcionalita, jako například nabízí nástroj *ExPASy — Translate Tool*[2].

Dále uvažujme, že se požaduje, aby vstupní mRNA sekvence po překladu na jednotlivé aminokyseliny obsahovala protein složený ze sekvence aminokyselin $MRTGNAN$ a podsekvenci aminokyselin KCF, přičemž pro popis algoritmu je jako protein uvažován velmi krátký protein *Microcin C7*[3]. Dále mohou být požadovány další proteiny a podsekvence, ovšem pro popis algoritmu nejsou přímo uvedeny.

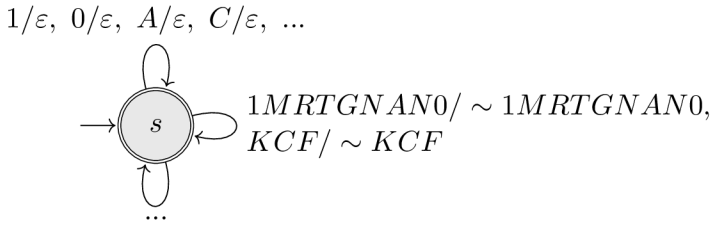
Výstup konečného převodníku T_1 je zpracován konečným převodníkem

$$T_2 = (Q_2, \Sigma_2, \Delta_2, \sigma_2, s, F_2),$$

kde

[2] https://web.expasy.org/translate/
[3] https://www.uniprot.org/uniprotkb/Q47505/

$$
\begin{aligned}
Q_2 &= \{s\},\\
\Sigma_2 &= \{1, 0, A, C, ..., Y\},\\
\Delta_2 &= \{1, 0, A, C, ..., Y\} \cup \{\sim\},\\
\sigma_2 &= \{s1 \to s\varepsilon, s0 \to s\varepsilon, sA \to s\varepsilon, ...,\\
&\qquad s1MRTGNAN0 \to s \sim 1MRTGNAN0, ...,\\
&\qquad sKCF \to s \sim KCF, ...\},\\
F_2 &= \{s\},
\end{aligned}
$$



Obrázek 6.3: Stavový diagram konečného převodníku T_2.

který označí a ponechá ve výstupním řetězci všechny vyhledávané proteiny a nepřekrývající se podsekvence aminokyselin. Stavový diagram T_2 je uveden na obrázku 6.3. Význam speciálního symbolu $\sim$ je totožný jako v podsekci 6.1.2. To stejné platí rovněž při zpracování množiny výstupních řetězců, přičemž pro popis lze odkázat na uvedenou podkapitolu. Jednotlivé výstupní řetězce konečného převodníku T_2 jsou následně vstupem vymazávacího systému, který je definován jako

$$ES = (\Sigma, E, R),$$

kde

$$
\begin{aligned}
\Sigma &= \{1, 0, \sim, A, C, ..., Y\},\\
E &= \{\sim 1MRTGNAN0, \sim KCF, ...\},\\
R &= \{\sim 1MRTGNAN0\}^+ ... \{\sim KCF\}^+ ....
\end{aligned}
$$

Je-li vstupní řetězec, který je přečtený ze vstupní pásky vymazávacím systémem, přijat, pak vstupní řetězec obsahuje všechny vyhledávané proteiny a sekvence aminokyselin, přičemž jejich nalezené výskyty se nepřekrývají.

Z uvedeného lze konstatovat, že vstupní sekvence mRNA obsahuje patřičné podsekvence tripletů, které kódují jednotlivé proteiny či sekvence aminokyselin. Zároveň platí, že kódující triplety se nachází v sekvenci mRNA mezi start kodonem a jedním ze stop kodonů, takže ve skutečnosti jsou při translaci opravdu překládány.

6.1.4 Vlastnosti sekundární struktury RNA

Jednotlivé nukleotidy RNA mohou vytvářet vazbu s komplementárními nukleotidy pomocí vodíku. Sekundární struktura RNA poté obsahuje určitý počet párových nukleotidů (nukleotidy, které jsou ve vazbě) a nukleotidů nepárových (více viz podsekce 6.1.1). Pro bližší zkoumání RNA ovšem může být vhodné studovat pouze ty sekvence, které obsahují určitý počet párových nukleotidů.

Z důvodu velkého množství nukleotidů v jednom řetězci RNA je vhodné zvolit procentuální definování množství. Tím lze určit, že řetězec je vhodné dále zkoumat pouze za podmínky, pokud splňuje minimální a maximální množství párových nukleotidů vůči všem obsaženým nukleotidům (párovým a nepárovým). Pro testování pouze minimální nebo pouze maximální hranice lze pro druhou hodnotu uvažovat její základní hodnotu (100% pro maximální množství a 0% pro množství minimální).

Na základě skutečnosti, že vymazávací systém je schopný ověřovat, zda vstupní řetězec obsahuje příslušný vztah množství jednotlivých symbolů, lze po příslušném převodu vstupní sekvence sekundární struktury RNA ověřovat popsané vlastnosti.

Popis algoritmu pro ověření vlastností sekundární struktury RNA

Nejprve bude algoritmus popsán pro jednu zvolenou procentuální hranici množství párových nukleotidů (minimální či maximální). Po zvolení procentuální hranice bude určen požadovaný vztah počtu párových a nepárových nukleotidů za účelem splnění podmínky. To lze provést následující sekvencí kroků.

Nechť je uvažováno celé číslo $0 \leq x \leq 100$, které udává požadované minimální či maximální množství párových nukleotidů vůči celkovému počtu nukleotidů RNA v procentech (například $x = 50$ určuje hranici 50%). Zároveň příslušná hodnota udává požadovaný minimální či maximální počet párových nukleotidů vůči 100 nukleotidům.

Pro minimalizaci počtu nukleotidů z implementačního hlediska je vhodné zmenšit počet párových nukleotidů vůči 100 nukleotidům při zachování stejného poměru. Toho lze docílit pomocí nalezení největšího společného dělitele čísel x a 100 pomocí funkce *gcd* (z anglického *greatest common divisor*),

$$gcd(x, 100) = d. \tag{6.1}$$

Následně lze obě uvedená čísla vydělit největším společným dělitelem d a vypočítat počet symbolů pro jeden párový nukleotid (p z anglického *paired*) a jeden nukleotid nepárový (u z anglického *unpaired*):

$$u = \frac{x}{d}, \tag{6.2}$$

$$p = \frac{100}{d} - u. \tag{6.3}$$

Na základě určených hodnot, při zachování požadovaného poměru párových nukleotidů vůči celkovému počtu nukleotidů, lze sestavit příslušný převodník. Ten převede vstupní sekvenci sekundární struktury RNA na řetězec, ve kterém každý symbol zastupující párový či nepárový nukleotid je nahrazen řetězcem s příslušným počtem symbolů se zachováním párových vlastností.

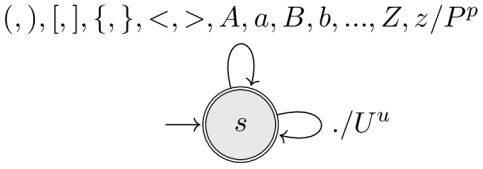
Vstupní sekundární struktura RNA může být zadaná v libovolném směru, a to $5' \rightarrow 3'$ nebo $3' \rightarrow 5'$. Dále je uvažována tečkovaná závorková notace, při použití symbolu „.“ pro nepárový nukleotid (viz podsekce 6.1.1).

Nechť

$$T = (Q, \Sigma, \Delta, \sigma, s, F) \tag{6.4}$$

je deterministický zobecněný sekvenční stroj (dále jen DGSM), přičemž

$$\begin{aligned} Q &= \{s\}, \\ \Sigma &= \{., (,), ..., A, a, ..., Z, z\}, \\ \Delta &= \{U, P\}, \\ \sigma &= \{s. \rightarrow sU^u, s(\rightarrow sP^p, ...\}, \\ F &= \{s\}. \end{aligned}$$



Obrázek 6.4: Stavový diagram deterministického zobecněného sekvenčního stroje T.

V definovaném DGSM T, jehož stavový diagram je vyobrazen na obrázku 6.4, P^p značí *p-tou* mocninu řetězce obsahujícího pouze symbol P a U^u značí *u-tou* mocninu řetězce obsahujícího pouze symbol U. Při implementaci daného převodníku probíhá samotná kontrola validity vstupní sekundární struktury RNA ještě před jeho sestavením, tudíž implementovaný DGSM uvažuje zároveň všechny povolené symboly pro nepárové nukleotidy, a to „.“, „:“ a „|“ dle podkapitoly 6.1.1.

Doposud popsaný algoritmický postup se provede jednou pro minimální množství párových nukleotidů a jednou pro maximální množství. Pokud minimální či maximální hodnota obsahuje základní hodnotu, lze celé ověřování vstupního řetězce vynechat, jelikož je splněna každým vstupním řetězcem sekundární struktury RNA. Navíc vynecháním všech algoritmických kroků pro minimální hodnotu 0% je zabráněno, aby vstupem funkce *gcd* bylo číslo nula.

Dále může být sestaven vymazávací systém pro minimální počet nukleotidů (včetně požadované hranice),

$$ES_{min} = (\Sigma_{min}, E_{min}, R_{min}), \tag{6.5}$$

kde

$$\begin{aligned} \Sigma_{min} &= \{U, P\}, \\ E_{min} &= \{U, P\}, \\ R_{min} &= \{UP\}^*\{P\}^* \end{aligned}$$

a vymazávací systém pro maximální počet nukleotidů (včetně požadované hranice),

$$ES_{max} = (\Sigma_{max}, E_{max}, R_{max}),$$

kde

$$\begin{aligned}\Sigma_{max} &= \{U, P\},\\ E_{max} &= \{U, P\},\\ R_{max} &= \{UP\}^*\{U\}^*.\end{aligned}$$

Jazyky přijímané těmito vymazávacími systémy jsou

$$\begin{aligned}L(E_{min}, R_{min}) &= \{w \in \{U, P\}^* \,|\, |w|_P \geq |w|_U\},\\ L(E_{max}, R_{max}) &= \{w \in \{U, P\}^* \,|\, |w|_U \geq |w|_P\}.\end{aligned}$$

Pro oba uvedené vymazávací systémy je vstupním řetězcem výstupní řetězec převodníku T z jednoprvkové množiny výstupních řetězců. Jediný rozdíl v těchto dvou definovaných vymazávacích systémech, ES_{min} a ES_{max}, je v řídícím regulárním jazyku.

Při minimálním požadovaném množství párových nukleotidů je nutné, aby bylo docíleno toho, že řetězec obsahuje minimálně totožný počet symbolů U a P, které byly do vstupního řetězce vymazávacího systému zakódovány pomocí DGSM T v požadovaném množství pro párové a nepárové nukleotidy. Po odstranění dvojic těchto symbolů může řetězec obsahovat pouze symbol P v libovolném počtu. Toho je docíleno pomocí definovaného regulárního jazyka $\{UP\}^*\{P\}^*$, kdy po vymazání všech dvojic symbolů U a P může být vymazán ze vstupní pásky libovolný počet symbolu P.

Naopak pro maximální požadované množství musí po odstranění všech dvojic symbolů U a P vstupní řetězec obsahovat pouze libovolný počet symbolu U, čehož je docíleno pomocí regulárního jazyka $\{UP\}^*\{U\}^*$.

Následně navržený algoritmus bude ještě popsán na jednom ukázkovém příkladu.

Příklad 6.1.1. Nechť je uvažován vstupní řetězec sekvence sekundární struktury RNA zadaný ve směru $5' \rightarrow 3'$, $w = ((.(([[.[[)))).)].]]]$, přičemž se jedná o teoreticky sestavený řetězec, který obsahuje přesně 80% párových nukleotidů. Požadované minimální množství párových nukleotidů vůči celkovému počtu nukleotidů v sekvenci RNA je 80%, tudíž vyplývá $x = 80$. Dále dle návrhu algoritmu je nejprve vypočítán největší společný dělitel čísel 80 a 100:

$$gcd(80, 100) = 20. \tag{6.6}$$

Poté mohou být obě vstupní čísla funkce gcd vydělena největším společným dělitelem a vypočítán počet symbolů pro jeden párový nukleotid (p, viz výraz 6.8) a počet symbolů pro nukleotid nepárový (u, viz výraz 6.7):

$$u = \frac{80}{20} = 4, \tag{6.7}$$

$$p = \frac{100}{20} - 4 = 1. \tag{6.8}$$

Následně může být sestaven deterministický zobecněný sekvenční stroj (dále jen DGSM) pro převod vstupní sekundární struktury RNA do patřičného tvaru pro vymazávací systém. Jako tento převodník je uvažován DGSM T (viz výraz 6.4). Celkově tedy bude řetězec w převeden na výstupní řetězec w', přičemž

$$w' = PPUUUUPPPPUUUUPPPPPUUUUPPUUUUPPP.$$

Jelikož požadované maximální množství párových nukleotidů je základní hodnotou určeno jako 100%, tato vlastnost nemusí být ověřována, protože je splněna každým vstupním řetězcem sekundární struktury RNA. Dále bude testována tedy pouze minimální hranice počtu párových nukleotidů. Jako vymazávací systém se uvažuje vymazávací systém ES_{min} (viz výraz 6.5). Ten může provést takovou sekvenci vymazávacích kroků, že lze přijmout vstupní řetězec w', a tudíž je možné prohlásit, že vstupní sekundární struktura RNA obsahuje minimálně 80% párových nukleotidů. Ve skutečnosti platí, že $|w'|_P = |w'|_U$. Tato situace také reprezentuje, že vstupní řetězec w obsahuje přesně 80% párových nukleotidů.

6.2 Aplikace v oblasti textových editorů

Následující sekce se zaměřuje na použití vymazávacího systému pro zpracování obecného a zdrojového textu v různých programovacích jazycích s využitím v textových editorech.

Podsekce 6.2.1 nejprve uvádí základní návrh využití vymazávacího systému pro zpracování obecného a zdrojového textu. Poté představuje návrh algoritmu navržené aplikace s možným využitím v textových editorech, včetně jednoduchého názorného příkladu.

6.2.1 Vyvážené závorky v textových editorech

Jak již bylo dříve prokázáno, vymazávací systém přijímá jakýkoliv Dyckův jazyk dle teorému 5.3.1. Tento jazyk je dobře známým bezkontextovým jazykem, a to především svou schopností definovat všechny řetězce s vyváženými závorkami [6]. Takové řetězce pak obsahují odpovídající počet dvojic závorek zapsaných v příslušném pořadí. Nejčastěji se jedná o páry, kdy se nejprve zapisuje takzvaná „otevírací“ závorka a poté její komplementární, závorka „uzavírací“, přičemž jednotlivé dvojice jsou správně zanořeny.

Při zpracování textu je možné pracovat celkem se třemi typy dvojic závorek, a to (), [] a {}. Pár závorek <> není uvažován. Dané závorky se mohou nacházet jak v obecném textu, tak například i ve zdrojových kódech programů v různých programovacích jazycích. Mezi vyjmenovanými dvojicemi se pak dále může nacházet libovolný tisknutelný znak vyjma znaků pro závorky a znaky pro mezery (samotná mezera, znaky označující nový řádek a další).

Algoritmus pro ověření vyvážených závorek

Pro účely demonstrace algoritmu budou uvažovány pouze znaky ASCII[4] namísto v současné době nejrozšířenějšího kódování UTF-8[5]. K samotné problematice lze přistoupit dvěma odlišnými způsoby při zpracování textu pomocí vymazávacího systému. První přístup definuje takový vymazávací systém, který nejprve vymaže všechny znaky ASCII vyjma znaků pro závorky. Po jejich vymazání jsou vymazávány jednotlivé dvojice závorek. Druhý přístup zařazuje před zpracování textu vymazávacím systémem deterministický zobecněný

[4] https://www.asciitable.com/
[5] https://home.unicode.org/

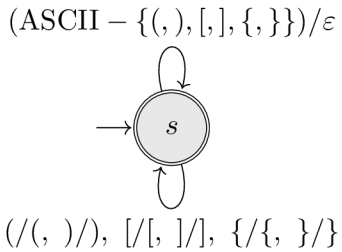
sekvenční stroj (dále jen DGSM), který slouží především pro přepis všech znaků, vyjma znaků pro závorky, na prázdný symbol. Pro aplikaci a implementaci bude zvolen druhý z popsaných přístupů především za účelem minimalizace nedeterminismu při vyhledávání symbolů ve vstupním řetězci vymazávacím systémem.

Vstupní text je nejprve zpracován DGSM

$$T = (Q, \Sigma, \Delta, \sigma, s, F), \tag{6.9}$$

kde

$$\begin{aligned} Q &= \{s\}, \\ \Sigma &= \{alph(ASCII - \{(,), [,], \{,\}\})\}, \\ \Delta &= \{\varepsilon, (,), [,], \{,\}\}, \\ \sigma &= \{sa \to s\varepsilon, sb \to s\varepsilon, ..., s(\to s(, ...\}, \\ F &= \{s\}, \end{aligned}$$



Obrázek 6.5: Stavový diagram deterministického zobecněného sekvenčního stroje T.

jehož stavový diagram je uveden na obrázku 6.5. Výstup tohoto DGSM je dále zpracován vymazávacím systémem

$$ES = (\Sigma, E, R), \tag{6.10}$$

kde

$$\begin{aligned} \Sigma &= \{(,), [,], \{,\}\}, \\ E &= \{(), [], \{\}\}, \\ R &= \{(), [], \{\}\}^*. \end{aligned}$$

Nutno poznamenat, že při implementaci dané aplikace jsou symboly závorek nevhodné pro definování regulárního výrazu reprezentujícího regulární jazyk. Také na základě algoritmu vymazávacího systému nejsou tyto symboly povoleny, pokud nezastupují určitý účel v regulárním výrazu a zároveň implementace vymazávacího systému povoluje příslušnou funkcionalitu použít. Ovšem dané omezení lze jednoduše vyřešit a minimalizovat jeho dopady. Implementace převádí jednotlivé symboly závorek na znaky z anglické abecedy

při využití velkých a malých znaků — symboly (,), [,], { a } po řadě zastupují znaky A, a, B, b, C a c.

Dále bude navržený algoritmus demonstrován na názorném příkladu.

Příklad 6.2.1. Nechť je uvažováno, že vstupem deterministického zobecněného sekvenčního stroje T (viz výraz 6.9) je jednoduchý program v jazyce C:

```
#include <stdio.h>

int main()
{
    printf("Hello world.\n");
}
```

Zdrojový kód tohoto programu převede daný konečný převodník na řetězec $w' = (\,)\{()\}$, který následně zpracovává vymazávací systém ES (viz výraz 6.10). Lze provést vymazávací kroky

$$\begin{aligned}((\,)\{\#()\}, \varepsilon) &\triangleright_{ES} ((\,)\#\{\}, ()) \\ &\triangleright_{ES} (\#(), (\,)\{\}) \\ &\triangleright_{ES} (\#, (\,)\{\}()),\end{aligned}$$

ze kterých vyplývá $(\,)\{\}() \in R$, což implikuje $w' \in L(E, R)$. Může být tedy konstatováno, že vstupní text obsahuje vyvážené závorky.

6.3 Aplikace v oblasti kompozičního šachu

Cílem této podkapitoly je představit aplikaci vymazávacího systému ve velmi specifické oblasti, a to v oblasti kompozičního šachu. Konkrétně se obsah následující části zaměřuje na problém n dam rozmístěných na šachovnici s n^2 poli. Samotný vymazávací systém přímo neřeší daný problém, ovšem jeho vlastnosti lze označit za ideální pro ověření, zda poskytnuté řešení daného problému odpovídá pravidlům a je správným řešením.

Následující podsekce, podsekce 6.3.1, nejprve představuje úvod do kompozičního šachu a popisuje kritéria vyhodnocení problému n dam. Následně obecně představuje nosnou myšlenku aplikace vymazávacího systému v dané oblasti a definuje algoritmus krok po kroku včetně názorných příkladů.

6.3.1 Problém n dam

Již v roce 1848 německý šachista Max Bezzel, který se věnoval problematice kompozičního šachu, definoval problém osmi královen [3], který lze formulovat následovně. Úkolem je položit osm šachových dam na šachovnici 8x8 tak, aby žádná z nich neležela v útočném poli některé z ostatních. Jinými slovy, žádné dvě dámy nesmí ležet ve stejném řádku, sloupci nebo diagonále. Existuje celkem *92 řešení*, která spadají do *12 tříd* (11 z 8 řešení a 1 ze 4), pokud jsou identifikována symetrická řešení [3]. Následně v roce 1869 Franz Nauck rozšířil tento problém na problém n dam, při rozmístění na šachovnici s n^2 poli [7].

Ověření pravidel problému n dam vymazávacím systémem

Pokud bude ověřováno kritérium, že žádné dvě dámy se nesmí nacházet ve stejném řádku, znamená to, že při označení řádků posloupností čísel $1, 2, ..., n$, se nesmí pro dvě figury dámy vyskytovat v souřadnici na vertikální hraně šachovnice stejné číslo (*osa y* v rovinné kartézské soustavě souřadnic). Taktéž je tomu i v případě druhého pravidla, kdy při posloupnosti symbolů z anglické abecedy se nesmí v souřadnici na horizontální hraně šachovnice (*osa x* v rovinné kartézské soustavě souřadnic) vyskytovat pro dvě figury dámy dva totožné symboly.

Na základě této skutečnosti lze uvážit, že vymazávací systém může ověřit správný počet výskytů symbolů ve vstupním řetězci. Tak je tomu i v případě kontroly pravidel pro řádky a sloupce problému n dam, kdy každé označení bude zastoupeno právě jednou. V případě diagonálních souřadnic (levá a pravá diagonální souřadnice) při počtu souřadnic $2 * n - 1$ a při počtu n figurek dam není možné použít stejný vymazávací systém z toho důvodu, že $(2 * n - 1) - n = n - 1$ symbolů nebude ve vstupním řetězci zastoupeno. Diagonální souřadnice šachovnice jsou vyobrazeny na obrázku D.2 v příloze D. Nabízí se dvě možná řešení:

(i) Každý symbol, který označuje diagonální souřadnici, se může vyskytovat v řetězci nejvýše jednou. V případě řídícího regulárního jazyka to znamená, že buď se na vstupní pásce daný symbol vyskytuje právě jednou nebo se uvažuje prázdný řetězec ε.

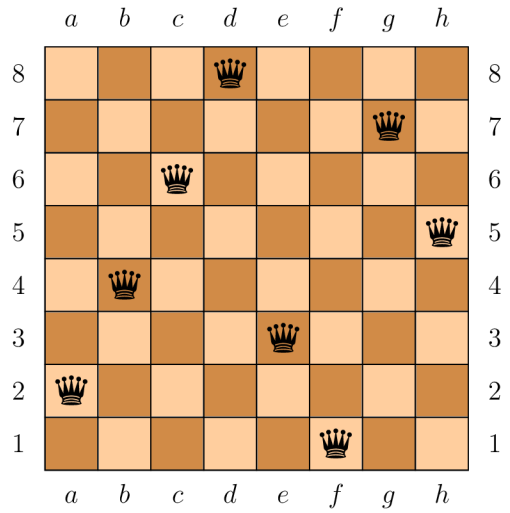
(ii) Lze pomocí vymazávacího systému ověřit, zda není porušena podmínka, že se žádné dvě figurky dámy nesmí nacházet na totožné diagonální souřadnici. To znamená, že pokud příslušný vymazávací systém přijímá vstupní řetězec, výsledek je negací vyhodnocení, tudíž vstupní řešení neodpovídá správnému rozmístění a nejméně dvě šachové dámy jsou umístěny na stejné diagonále. V opačném případě, pokud daný vymazávací systém nepřijme řetězec, který je přečten ze vstupní pásky, pak ve vstupním řešení se nenachází žádné dvě figurky dámy na stejné diagonální souřadnici.

Za účelem eliminace prázdného řetězce v řídícím regulárním jazyku návrh algoritmu využívá druhý z uvedených algoritmických postupů.

Návrh algoritmu pro validaci řešení problému n dam

Nejprve je nutné navrhnout zakódování rozmístění dam na šachovnici do řetězce. To lze přirozeně provést pomocí konkatenace jednotlivých symbolů souřadnic každé z dam, kdy nejprve se zapíše souřadnice řádku, na kterém se figura nachází, a poté sloupce. Například pro šachovnici 8x8 jedno z možných řešení, uvedené na obrázku 6.6, může být zakódováno do řetězce $2a4b6c8d3e1f7g5h$. Stanovený postup zakódování lze přímo využít pro vstupní řetězce vymazávacího systému pro $n \leq 9$. Pokud je hodnota $n \geq 10$, není možné pomocí vymazávacího systému správně rozpoznat vstupní zakódování z důvodu, že číselná hodnota se zapisuje ve dvou či více symbolech. Pro využití vymazávacího systému při ověřování řešení i pro $n \geq 10$ je nutné číselné hodnoty převést do zakódování, kde každý řádek označuje právě jeden symbol. Jelikož každý sloupec označují znaky anglické abecedy, přirozeně se nabízí převést číselné hodnoty na znaky abecedy řecké ve stejném pořadí. Tato abeceda obsahuje celkem 24 znaků, kdy každý znak lze zapsat jako malý či velký, tedy celkem řecká abeceda nabízí k dispozici 48 symbolů[6]. Na základě skutečnosti, že pro zakódování diagonálních

[6] https://www.britannica.com/topic/Greek-alphabet



Obrázek 6.6: Vybrané řešení problému 8 dam.

souřadnic je zapotřebí $2 * n - 1$ symbolů, největší možný rozměr šachovnice se stanoví jako $n = \lfloor \frac{48+1}{2} \rfloor = 24$. Ovšem pro $n = 24$ při počtu 227 514 171 973 736 řešení[7] je povolený rozsah rozměru šachovnice pro demonstrační účely více než dostatečný. Pro větší rozměry šachovnice by bylo nutné zvolit další unikátní symboly pro zakódování.

Na základě návrhu zakódování lze definovat konečný převodník pro zpracování vstupního řetězce. Nutno poznamenat, že při implementaci aplikace se daný převodník vytváří dynamicky podle hodnoty n. Nyní pro demonstraci algoritmu bude zvoleno $n = 8$. Jedná se tedy o běžný rozměr šachovnice. Nechť

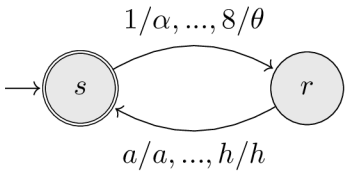
$$T_8 = (Q, \Sigma, \Delta, \sigma, s, F) \tag{6.11}$$

je konečný převodník, kde

$$\begin{aligned} Q &= \{s, r\}, \\ \Sigma &= \{1, ..., 8, a, ..., h\}, \\ \Delta &= \{\alpha, ..., \theta, a, ..., h\}, \\ \sigma &= \{s1 \to r\alpha, ..., s8 \to r\theta, \\ &\qquad ra \to sa, ..., rh \to sh\}, \\ F &= \{s\}, \end{aligned}$$

který je vyobrazen na obrázku 6.7.

[7] https://oeis.org/A000170



Obrázek 6.7: Stavový diagram konečného převodníku T_8.

Výstup konečného převodníku T_8 zpracovává vymazávací systém, který je sestaven přesně pro daný rozměr šachovnice, tedy pro dané n. Tím zároveň pomocí příslušného vymazávacího systému proběhne kontrola, zda vstupní řetězec, který obsahuje zakódované řešení, obsahuje pouze platné symboly pro n (hodnoty v povoleném rozsahu pro rozměr šachovnice).

Je definován vymazávací systém

$$ES_8 = (\Sigma_8, E_8, R_8), \tag{6.12}$$

kde

$$\begin{aligned}\Sigma_8 &= \{\alpha, ..., \theta, a, ..., h\},\\ E_8 &= \{\alpha, ..., \theta, a, ..., h\},\\ R_8 &= \{\alpha...\theta a...h\}.\end{aligned}$$

Definovaný vymazávací systém, ES_8, přijme vstupní řetězec tehdy a jen tehdy, když se nenachází žádné dvě dámy ve stejném sloupci či řádku šachovnice a po šachovnici s $8^2 = 64$ poli je rozmístěno 8 dam. Provede se tudíž kontrola řešení v horizontálním a vertikálním směru šachovnice.

Pro kontrolu řešení v diagonálních souřadnicích je nyní nutné převést souřadnice jednotlivých dam na šachovnici ze souřadnic na horizontálních a vertikálních hranách na souřadnice diagonální. Základní a diagonální souřadnice jsou vyobrazeny v příloze D.

Pro dynamické mapování souřadnic dle hodnoty n je zapotřebí určení vztahu pozice dámy v horizontálně-vertikálních souřadnicích a souřadnicích diagonálních.

Nechť proměnná i značí pořadí znaku v řecké abecedě při číslování od nuly. Pro znaky $\alpha, \beta, \gamma, ...$ je odpovídající hodnota proměnné i po řadě $0, 1, 2, ...$ nejvíce do $n-1$. Totožným postupem lze odvodit pořadí symbolů v anglické abecedě, jenž značí proměnná j. Pro určení nového symbolu označujícího levou diagonální souřadnici proměnná g uchovává pozici nového znaku v řecké abecedě při zachování číslování od nuly. Opět jako v předchozím případě je tomu i s proměnnou e pro symbol anglické abecedy označující pravou diagonální souřadnici. Na základě definování proměnných lze určit následující vztahy:

$$g = n + i - j - 1, \tag{6.13}$$
$$e = i + j. \tag{6.14}$$

Názorná ukázka výpočtu převodu souřadnic pro řešení uvedené v obrázku 6.6 je ve formě příkladu připojena v příloze D.

Po provedení převodu z horizontálně-vertikálních souřadnic je možné ověřit, zda vstupní řešení neobsahuje dvě či více figur na stejné diagonále, tedy ať již na levé či pravé diagonální souřadnici. Vícenásobný výskyt dam na stejné diagonále by byl v převedeném řetězci zastoupen mnohačetným výskytem symbolu. Ověření správnosti vstupního řešení provádí následující vymazávací systém, který je definován jako

$$ES_{D8} = (\Sigma_{D8}, E_{D8}, R_{D8}), \tag{6.15}$$

kde

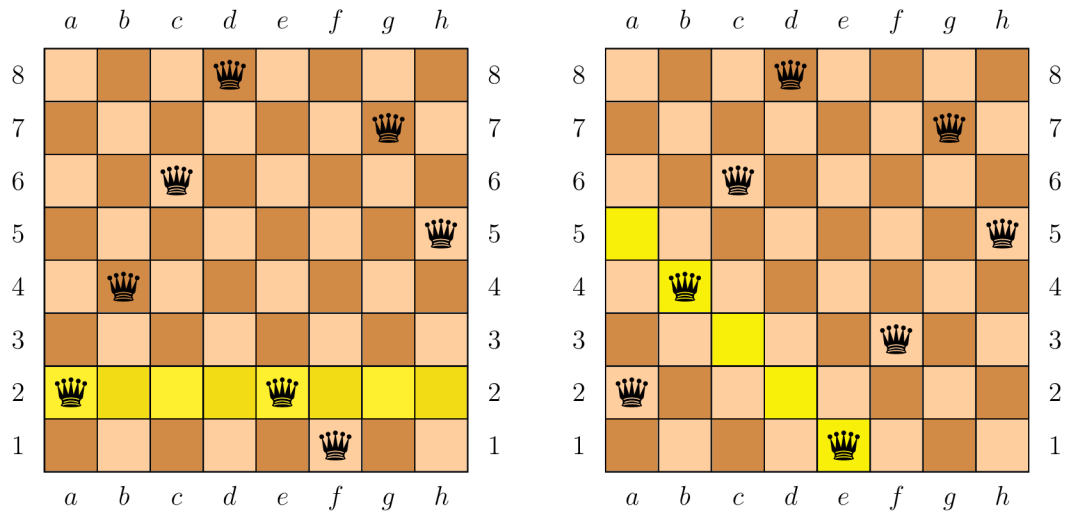
$$\begin{aligned}
\Sigma_{D8} &= \{\alpha, ..., o, a, ..., o\}, \\
E_{D8} &= \{\alpha, ..., o, a, ..., o\}, \\
R_{D8} &= (\alpha\alpha, ..., oo, aa, ..., oo)(\alpha, ..., o, a, ..., o)^*.
\end{aligned}$$

Jak již bylo dříve zmíněno, ověření validity vstupního řešení v diagonálních souřadnicích je negací výsledku vymazávacího systému ES_{D8}. Pokud tedy daný vymazávací systém přijme vstupní řetězec, ve vstupním řešení se vyskytují alespoň dvě figury dámy na stejné diagonále. Nejsou tedy splněny podmínky problému n dam pro $n = 8$ a vstupní řešení není správným řešením problému. V opačném případě, pokud ES_{D8} nepřijme řetězec, řešení obsahuje na každé diagonální souřadnici nejvýše jednu figuru dámy. Jsou tedy splněny podmínky problému a řešení lze prohlásit za správné.

Pro ukázku bude uvedený algoritmus ještě demonstrován na následujících dvou příkladech, přičemž žádné z uvedených vstupních řešení nesplňuje pravidla problému n dam.

Příklad 6.3.1. Nechť řetězec $w = \mathbf{2}a4b6c8d\mathbf{2}e1f7g5h$ je vstupem algoritmu pro ověření správnosti řešení problému rozmístění n dam na šachovnici s n^2 poli pro $n = 8$. Vstupní rozmístění figur dam vyobrazuje obrázek 6.8a. Řetězec w nejprve zpracuje konečný převodník T_8 (viz výraz 6.11), který jej převede na řetězec $w' = \boldsymbol{\beta}a\delta b\zeta c\theta d\boldsymbol{\beta}e\alpha f\eta g\epsilon h$. Následně w' je řetězcem, který je přečten ze vstupní pásky pomocí vymazávacího systému ES_8 (viz výraz 6.12). Řetězec w' ovšem není přijat daným vymazávacím systémem na základě skutečnosti, že vstupní řešení obsahuje dvě figury dámy umístěné na stejném řádku, a to přesněji na druhém řádku označeným číslem 2. Lze tedy prohlásit, že vstupní rozmístění dam na šachovnici není správným řešením problému n dam pro $n = 8$.

Příklad 6.3.2. Nechť řetězec $w = 2a4b6c8d1e3f7g5h$ je vstupem algoritmu pro ověření správnosti řešení problému rozmístění n dam na šachovnici s n^2 poli pro $n = 8$. Vstupní rozmístění figur dam vyobrazuje obrázek 6.8b. Řetězec w nejprve zpracuje konečný převodník T_8 (viz výraz 6.11), který jej převede na řetězec $w' = \beta a\delta b\zeta c\theta d\alpha e\gamma f\eta g\epsilon h$. Následně w' je řetězcem, který je přečten ze vstupní pásky pomocí ES_8 (viz výraz 6.12) a tímto vymazávacím systémem je také přijat. Dle návrhu algoritmu následuje převod souřadnic řešení z horizontálně-vertikálních souřadnic do souřadnic diagonálních. Řetězec w' je tedy převeden na řetězec $w'' = \iota b\kappa\boldsymbol{e}\lambda h\mu k\delta\boldsymbol{e}\epsilon h\theta m\epsilon l$, který následně zpracovává vymazávací systém ES_{D8} (viz výraz 6.15). Ovšem w'' není přijato uvedeným vymazávacím systémem na základě skutečnosti, že vstupní řešení obsahuje dvě figury dámy umístěné na stejné diagonále, a to přesněji na páté pravé diagonální souřadnici označené e (viz obrázek D.2b). Lze tedy prohlásit, že vstupní rozmístění dam na šachovnici není správným řešením problému n dam pro $n = 8$.



(a) Řešení se dvěma dámami v jednom řádku. (b) Řešení se dvěma dámami v jedné diagonále.

Obrázek 6.8: Nesprávné rozmístění dam na šachovnici pro $n = 8$.

Kapitola 7

Implementace vymazávacího systému a jeho aplikací

Následující kapitola prezentuje algoritmus činnosti vymazávacího systému, který byl zaveden do oblasti formální teorie jazyků v kapitole 5. Následně na základě návrhu algoritmu popisuje jeho implementaci, včetně programového řešení navržených aplikací z kapitoly 6.

Pro implementaci byl zvolen vysoko-úrovňový interpretovaný programovací jazyk *Python 3*. Jako výhody volby uvedeného programovacího jazyka lze označit především vhodnou základní funkcionalitu pro práci s řetězci, rozsáhlou standardní knihovnu a uživatelskou přívětivost. Na základě skutečnosti, že vytvořené aplikace slouží především pro demonstrační účely navrženého formálního systému a nejedná se tedy o kritické systémy z hlediska výkonu, lze použít uvedený interpretovaný jazyk oproti jazykům kompilovaným či jazykům využívajícím oba zmíněné přístupy. Implementace poskytuje jednotlivé aplikace vymazávacího systému jako příslušné nástroje ve formě konzolových aplikací. Cílem těchto aplikací je především předložení možných praktických využití vymazávacích systémů.

Hlavní struktura této kapitoly je rozdělena na dvě logické části. Nejprve sekce 7.1 popisuje navržený algoritmus činnosti vymazávacího systému včetně úvodu s důkladným popisem jednotlivých podčástí algoritmu, jenž také zachycuje uvedený pseudoalgoritmus. Za druhé popisuje sekce 7.2 základní hierarchickou strukturu programového řešení aplikací vymazávacího systému.

7.1 Algoritmus činnosti vymazávacího systému

Cílem této sekce je představit základní algoritmus činnosti vymazávacího systému, který byl zaveden do oblasti formální teorie jazyků v kapitole 5.

Celkem je sekce rozdělena do tří částí. Nejprve podsekce 7.1.1 popisuje základní koncept algoritmu, včetně jeho důkladného vysvětlení a navrženého pseudoalgoritmu. Podsekce 7.1.2 popisuje rozdíly mezi dvěma použitými typy kvantifikátorů. Nakonec podsekce 7.1.3 přibližuje důvody využití testování všech pozic vymazávacích řetězců.

7.1.1 Hlavní algoritmus vymazávacího systému

Vytvořené programové řešení poskytuje obecnou implementaci činnosti vymazávacího systému bez přímého definování řešení jakékoliv jeho aplikace. Tuto zmíněnou entitu, v podobě třídy dle objektově orientovaného návrhu ve vybraném jazyce, využívají jednotlivé nástroje.

Samotný hlavní tok algoritmu činnosti vymazávacího systému je nejprve obecně představen ve výpisu uvedeném v příloze E. Stejná příloha jej také prezentuje ve formě pseudokódu.

Obecně řečeno cílem algoritmu je ověřit, zda existuje taková posloupnost vymazávacích kroků, že výsledný řetězec vzniklý jejich konkatenací náleží do definovaného regulárního jazyka. Tento jazyk řídí samotný postup vymazávání, ovšem algoritmus obsahuje nedeterministické vlastnosti, kdy v příslušném kroku může být pro vymazání dle řídícího jazyka zvoleno více vymazávacích řetězců. Nelze však vyhodnotit, použití kterého vymazávacího řetězce vede ke správnému řešení. Vyplývá, že samotná činnost vymazávacího systému, který lze jednoduše formálně definovat, se transformuje na úlohu prohledávání stavového prostoru (více viz [25]).

Částečně volba vymazávacích řetězců je určena typem kvantifikátoru, jenž bude popsán v následující podkapitole. Nutno ještě uvést, že kromě volby vymazávacích řetězců pro některé vstupní řetězce také může velmi záležet na správné volbě jejich výskytů v řetězci, pokud se na vstupní pásce nacházejí více jak jednou. Na základě struktury aktuální podoby řetězce totiž může být nutné vymazat právě jeden konkrétní výskyt, aby po jeho vymazání a spojení obou stran vstupní pásky, nalevo a napravo od vymazávaného řetězce, vznikl nový podřetězec, který musí být následně vymazán dle řídícího jazyka.

Při implementaci je ovšem pro některé aplikace prohledávání všech pozic vymazávacích řetězců značně neefektivní a na základě definované abecedy nemusí být ani tato možnost testována. Proto je volba, zda mají být ověřovány všechny pozice aplikovatelných vymazávacích řetězců či nikoliv, ponechána volitelně.

7.1.2 Chamtivý a líný kvantifikátor

Při implementaci popsaný hlavní algoritmus činnosti vymazávacího systému obsahuje rozšíření o typy kvantifikátorů dle regulárních výrazů (viz [13]), přičemž je možné aplikovat buď takzvaný *„chamtivý“* kvantifikátor (z anglického *greedy quantifier*), který je nastaven jako výchozí typ kvantifikátoru, nebo poté takzvaný *„líný“* kvantifikátor (z anglického *lazy quantifier*). Rozdíl ve zpracování vstupního řetězce pomocí vymazávacího systému lze názorně popsat na následujících dvou vymazávacích systémech. Nechť $ES_1 = (\Sigma_1, E_1, R_1)$ je vymazávací systém, kde

$$\begin{aligned}\Sigma_1 &= \{a, b, c\},\\ E_1 &= \{ab, ac\},\\ R_1 &= \{ab\}^*\{ac\}\{ab\}^*,\end{aligned}$$

a řetězec $w_1 = abacab$ je řetězcem, který je přečten ze vstupní pásky pro ES_1. Dále nechť $ES_2 = (\Sigma_2, E_2, R_2)$ je vymazávací systém, kde

$$\begin{aligned}\Sigma_2 &= \{a, b, c, d\},\\ E_2 &= \{ab, ac, bd\},\\ R_2 &= \{ab\}^*\{bdac\}\{ab\}^*,\end{aligned}$$

a řetězec $w_2 = abdaabcb$ je řetězcem, který je přečten ze vstupní pásky pro ES_2. Je uvažováno, že vymazávací systémy jsou nastaveny v režimu, kdy se neověřují všechny pozice vy-

mazávacích řetězců, tedy pro algoritmus 1 v příloze E platí, že $test\_all\_positions \leftarrow False$. Zpracování vstupních řetězců pomocí jednotlivých vymazávacích systémů pro dva typy kvantifikátorů bude následující:

- *Chamtivý kvantifikátor*: Cílem chamtivého kvantifikátoru je, aby operátory řídícího regulárního jazyka, jako je „$*$“ a „$+$“, pojmuly co největší množství znaků ze vstupního řetězce, přičemž je zachován jejich základní význam („$*$“ znamená nula a více iterací, „$+$“ jednu a více iterací). Z vizuálního hlediska je regulární jazyk procházen lineárně zleva doprava a každý operátor pojme co nejvíce řetězců, pro které je přiřazen. Pokud by byl například uvažován regulární jazyk R_1, vstupní řetězec w_1 a vymazávací řetězce z E_1, budou nejprve vymazány všechny podřetězce ab z w_1 a teprve jako poslední bude vymazán podřetězec ac.

  Implementace ovšem zachovává postup chamtivého kvantifikátoru v regulárních výrazech. Pokud by tedy daný operátor některé podčásti regulárního jazyka zpracoval co největší množství řetězců, ke kterým je přiřazen, a zároveň na základě tohoto chování by nemohl být v aktuální fázi vstupní řetězec přijat definovaným vymazávacím systémem, je proveden zpětný krok (anglicky *backtracking*, více viz [25]) a ověřován jiný postup vymazávání podřetězců ze vstupního řetězce.

  Dané chování lze demonstrovat na definovaném vymazávacím systému ES_2, vstupním řetězci w_2 a vymazávacích řetězcích z E_2. Postup daného kvantifikátoru je nejprve takový, že ze vstupního řetězce jsou vymazány všechny podřetězce ab, přičemž řetězec na vstupní pásce po jejich vymazání je řetězec $w_2' = dacb$. Ovšem dle regulárního jazyka R_2 a dle vymazávacích řetězců z E_2 by měl být jako následující vymazán podřetězec bd a následně podřetězec ac. V aktuální fázi však není možné aplikovat vymazávací řetězec bd. Bude tedy proveden zpětný krok, jelikož není možné vymazat jakýkoliv jiný podřetězec. Následně se ověří jiný postup vymazávání řetězců z E_2. Pokud bude uvažováno, že jako poslední podřetězec byl vymazán řetězec ab v pořadí první zleva ve vstupním řetězci w_2, tedy vizuálně znázorněno jako $w_2 = \boldsymbol{ab}daabcb$, bude v aktuální fázi pro řetězec $abdacb$, na rozdíl od předchozího postupu, vymazán řetězec bd a následně i řetězec ac. Po vymazání těchto dvou řetězců bude řetězcem na vstupní pásce řetězec ab, pro který se jako poslední vymazávací řetězec použije právě řetězec ab z E_2 a vstupní řetězec w_2 bude přijat vymazávacím systémem ES_2.

- *Líný kvantifikátor*: Naopak cílem líného kvantifikátoru je, aby operátory regulárního jazyka „$*$“ a „$+$“ pojmuly co nejméně řetězců, ke kterým jsou přiřazeny, tedy pro „$*$“ to je v ideálním případě žádný řetězec a pro „$+$“ právě jeden. Postup líného kvantifikátoru lze uvažovat jako zcela opačný ke kvantifikátoru chamtivému. Pokud je tedy regulární jazyk procházen lineárně zleva doprava, každý operátor pojme co nejméně řetězců, ke kterým je přiřazen. Pokud by byl například uvažován regulární jazyk R_1, vstupní řetězec w_1 a vymazávací řetězce z E_1, bude nejprve vymazán podřetězec ac a následně až poté všechny podřetězce ab.

  Jako je tomu v případě chamtivého kvantifikátoru, i líný kvantifikátor zachovává postup z regulárních výrazů a je využíván algoritmus zpětného kroku. Tento algoritmus bude dále popsán pro vymazávací systém ES_2, tedy pro regulární jazyk R_2, vstupní řetězec w_2 a vymazávací řetězce z E_2. Postup bude následující. Jako první se ze vstupního řetězce w_2 vymaže podřetězec bd z E_2, tudíž první podčást regulárního jazyka, $\{ab\}^*$, nebude prozatím využita pro vymazávání. Ze vstupního řetězce jako následující musí být vymazán podřetězec ac. To v aktuální fázi není možné a provede se tedy

zpětný krok. Po provedení zpětného kroku opět není prozatím žádný podřetězec vymazán ze vstupního řetězce a namísto vymazávacího řetězce bd bude nejprve použit řetězec ab. Uvažuje se, že jako řetězec ab bude zvolen v pořadí druhý podřetězec ab, graficky znázorněno jako $w_2 = abda\boldsymbol{ab}cb$. Po jeho vymazání pro řetězec $w_2' = abdacb$ se vymazávací systém pro líný kvantifikátor opět nejprve pokusí vymazat podřetězec bd a následně podřetězec ac. Nyní je možné jak první, tak i druhý ze zmíněných podřetězců vymazat z důvodů vymazání v pořadí druhého podřetězce ab z původního tvaru vstupního řetězce. Poté již vymazávací systém vyjme ze vstupního řetězce $w_2'' = ab$ poslední podřetězec ab a vstupní řetězec je přijat ES_2.

Pro lepší vizuální demonstraci celého postupu zpracování vstupního řetězce w_2 pomocí ES_2, který byl popsán pro oba typy kvantifikátorů, lze odkázat na obrázek 7.1. Varianta zpracování vstupního řetězce w_2 pro chamtivý kvantifikátor je uvedena v obrázku 7.1a. Pro variantu zpracování řetězce pro líný kvantifikátor viz obrázek 7.1b. Je ovšem uvažováno, že podřetězec ab nacházející se v druhé úrovni stromu reprezentuje pro chamtivý kvantifikátor v pořadí první podřetězec v řetězci $w_2 = \boldsymbol{ab}daabcb$ a pro líný kvantifikátor až druhý zleva, $w_2 = abda\boldsymbol{ab}cb$. Jinak je zapotřebí uvažovat ještě zvolení správné pozice podřetězce a může být nutné provést během vykonávání algoritmu více zpětných kroků. Pro demonstraci algoritmu je nyní uvažována pro oba typy kvantifikátorů nejoptimálnější varianta.

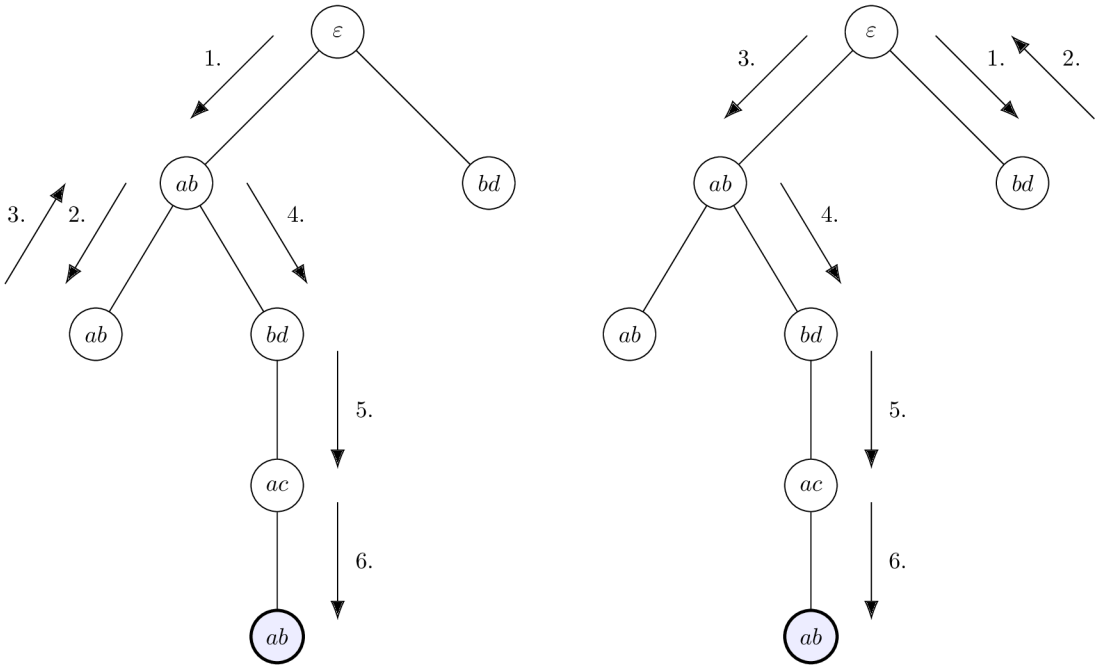
Na základě stavového prostoru řešení vstupního problému při přijímání vstupních řetězců se dá také na algoritmus nahlížet jako na algoritmus *prohledávání do hloubky* (anglicky *depth first search*, více viz [25]), který je řízen regulárním jazykem a nedeterministicky vymazávanými řetězci. Jednotlivé vymazávací řetězce v obrázku 7.1 tvoří uzly a při procházení stromu je jako řetězec vzniklý konkatenací uvažován řetězec, který vznikne konkatenací všech řetězců zapsaných v uzlech od kořenového uzlu po aktuálně vyhodnocovaný.

Na první pohled se může zdát, že jak pro chamtivý, tak i pro líný kvantifikátor, přijme vymazávací systém vstupní řetězec s využitím stejného počtu kroků. Ovšem počet kroků k přijetí vstupního řetězce vzhledem k chování daných typů kvantifikátorů se především odvíjí od definice řídícího regulárního jazyka a struktury vstupního řetězce. Algoritmické zpracování vstupního řetězce pro totožný vymazávací systém tedy může být na základě těchto vlastností velmi rozdílné dle zvoleného typu kvantifikátoru.

V implementačním řešení pro oba typy kvantifikátorů jsou aplikovatelné vymazávací řetězce označeny hodnotou, která určuje úroveň pozice v regulárním jazyce. Například pro regulární jazyk definovaný regulárním výrazem $(ab)^*(cd)^*(ef)^*$, podvýrazy (ab), (cd) a (ef) regulárního výrazu odpovídají po řadě úrovním 1, 2 a 3. Zároveň pokud by pro daný regulární výraz mohly být použity vymazávací řetězce, které v sobě obsahují iterace podvýrazů, jsou pro každý vymazávací řetězec označeny i počty opakování jednotlivých podvýrazů. Pro chamtivý kvantifikátor jsou nejprve zvoleny nejvyšší číselné hodnoty a pro líný hodnoty nejnižší.

7.1.3 Testování všech pozic vymazávacích řetězců

Jak již bylo uvedeno, správné vyhodnocení, zda vstupní řetězec může být přijat definovaným vymazávacím systémem či nikoliv, nezávisí pouze na vymazávacích řetězcích, ale také na zvolených pozicích, ze kterých jsou řetězce vymazávány, pokud se dané vymazávací řetězce vyskytují ve vstupním řetězci vícekrát. Pro názorný příklad uvažujme vstupní řetězec $w_2 = abdaabcb$ a vymazávací systém ES_2 z podsekce 7.1.2. Aby vstupní řetězec mohl být přijat definovaným vymazávacím systémem, je nutné, aby byl nejprve vymazán v pořadí



(a) Zpracování pro chamtivý kvantifikátor.

(b) Zpracování pro líný kvantifikátor.

Obrázek 7.1: Zpracování řetězce *abdaabcb* s využitím obou typů kvantifikátorů (chamtivý nebo líný) a za implicitní podmínky zvolení správných pozic vymazávacích řetězců pro docílení nejoptimálnějšího chování.

druhý podřetězec *ab* z řetězce $w_2 = abda\boldsymbol{ab}cb$. Teprve poté je možné vymazat následně podřetězce *bd* a *ac* a následně poslední podřetězec *ab*, který vznikne konkatenováním prvního a posledního symbolu řetězce w_2 po spojení vstupní pásky. Tudíž při činnosti algoritmu vymazávacího systému popsaného v podsekci 7.1.1 je zapotřebí při nedeterministickém výběru vymazávacího řetězce také provést nedeterministický výběr jeho pozice.

Pro některé specifické úlohy ovšem může testování všech pozic příslušného vymazávacího řetězce způsobit neefektivnost algoritmu, a to například pro aplikaci vymazávacího systému pro vyhodnocení vlastností sekundární struktury RNA, která byla blíže popsána v podsekci 6.1.4. Jelikož se vstupní řetězec vymazávacího systému skládá pouze ze dvou symbolů, a to U a P, nemá sebemenší význam testovat pozice těchto symbolů na základě skutečnosti, že vymazávací řetězce jsou jednosymbolové a zároveň totožné s uvedenými dvěma symboly. Naopak v některých případech, jako je tomu například pro řetězec w_2 a vymazávací systém ES_2, je přímo nutné otestování všech pozic vymazávacích řetězců, pokud prozatím nebylo nalezeno takové řešení, že by mohl být vstupní řetězec přijat.

V popisu hlavního algoritmu v podsekci 7.1.1 pro definování, zda mají být testovány všechny pozice vymazávacích řetězců či nikoliv, slouží proměnná `test_all_positions`.

7.2 Architektura programového řešení

Následující sekce popisuje architekturu implementace algoritmu vymazávacího systému představeného v sekci 7.1 a jednotlivých aplikací popsaných v kapitole 6. Jednotlivé formální

modely byly otestovány pomocí vlastních jednotkových testů, které využívají knihovnu *unittest*[1] ze standardní knihovny jazyka Python 3.

Tato podkapitola je rozdělena celkem na čtyři části. Nejprve podsekce 7.2.1 popisuje základní architekturu celého programového řešení. Poté podsekce 7.2.2 přibližuje rozvržení implementací jednotlivých typů převodníků a podsekce 7.2.3 vysvětluje stěžejní části implementace vymazávacího systému. V poslední řadě podsekce 7.2.4 přiřazuje pro jednotlivé aplikace vymazávacího systému jejich nástroje v implementačním řešení.

7.2.1 Základní architektura

Jednotlivé aplikace vymazávacího systému představené v kapitole 6 implementují příslušné nástroje. Každý nástroj představuje jednu implementaci aplikace a ve formě skriptu je spouštěn každý zvlášť s příslušnými argumenty.

Základní architektura implementačního řešení obsahujícího jednotlivé nástroje se skládá z následujících částí:

- Zpracování vstupních dat.
  - Zpracování argumentů příkazové řádky.
  - Načtení dat ze vstupního souboru.
- Zpracování chybových stavů.
  - Chybové návratové kódy.
  - Vlastní výjimky.
- Formální modely.
  - Vymazávací systém.
  - Převodníky.
- Nástroje implementující aplikace.
- Pomocné nástroje.
- Testy pro formální modely.

7.2.2 Převodníky

Jak již bylo dříve uvedeno v kapitole 6, jednotlivé aplikace vymazávacího systému využívají různých typů převodníků pro předzpracování vstupního řetězce. Konkrétně aplikace využívají dva typy převodníků, a to ten nejobecnější, *konečný převodník*, a pak striktnější, *deterministický zobecněný sekvenční stroj*. Po řadě uvedené převodníky implementují metody tříd `FiniteTransducer` a `DeterministicGeneralizedSequentialMachine`, které s využitím dědičnosti jsou potomky abstraktní třídy `FormalModel`.

Zároveň totožně s formálním přístupem je třída `DeterministicGeneralizedSequentialMachine` odvozena od třídy `FiniteTransducer`. Pro verifikaci jednotlivých implementací převodníků pro každý typ převodníku přináleží jednotlivé testy, které obsahují příslušné třídy `FiniteTransducerTest` a `DeterministicGeneralizedSequentialMachineTest`.

[1] https://docs.python.org/3/library/unittest.html

Hlavní funkci konečného převodníku implementuje metoda `run()` náležející do třídy `FiniteTransducer`. Základní logika funkce vychází z implementace konečného převodníku[2] v knihovně `pyformlang`[3], která byla vytvořena v [23].

7.2.3 Vymazávací systém

Vymazávací systém definovaný v kapitole 5 a návrh příslušného algoritmu činnosti, popsaného v sekci 7.1, implementují metody třídy `ErasingSystem`. Daná třída s využitím dědičnosti je potomkem abstraktní třídy `FormalModel`, jako je tomu i v případě převodníků.

Implementace algoritmu činnosti vymazávacího systému

Samotný algoritmus využívá zpětného kroku, a tudíž dle intuitivního přístupu i rekurzivních volání na základě samotné rekurzivní definice problematiky. Metoda implementující hlavní algoritmus, `_is_accepted()`, při své činnosti využívá i pomocných metod a funkcí. Za fundamentální lze především označit metodu `_match_regular_language()`, která implementuje samotné ověřování, že řetězec, který by vznikl konkatenací doposud vymazaných řetězců a následně nedeterministicky vybraného vymazávacího řetězce, by po libovolném počtu kroků mohl náležet do řídícího regulárního jazyka. Implementace algoritmu činnosti vymazávacího systému také uvažuje typy kvantifikátorů (chamtivý nebo líný kvantifikátor), jenž byly blíže popsány v podsekci 7.1.2.

Jako tomu je v případně převodníků, tak i pro verifikaci vymazávacího systému přísluší jednotkové testy implementované v metodách třídy `ErasingSystemTest`.

7.2.4 Nástroje implementující aplikace vymazávacího systému

Na základě navržených aplikací vymazávacího systému, jenž byly uvedeny a popsány v kapitole 6, implementují jednotlivé aplikace příslušné nástroje, jejichž vztahy k aplikacím jsou uvedeny v tabulce 7.1. Veškeré hlavní algoritmy nástrojů dodržují popsané a demonstrované algoritmické postupy.

Aplikace vymazávacího systému	**Příslušející třída**
Vyhledávání sekvencí v DNA, RNA a sekundární struktuře RNA	`SequenceSearchTool`
Vyhledávání proteinů a sekvencí aminokyselin proteinů	`AminoAcidSequenceSearchTool`
Vlastnosti sekundární struktury RNA	`SecondaryStructurePropertiesTool`
Vyvážené závorky v textových editorech	`BalancedBracketsTool`
Validace řešení problému n dam	`NQuensProblemTool`

Tabulka 7.1: Vztahy aplikací vymazávacího systému a příslušných tříd implementovaných nástrojů.

[2] https://github.com/Aunsiels/pyformlang/blob/ab999ff9ba73e4f5f9ec59dd919da1d85038700d/pyformlang/fst/fst.py#L164

[3] https://github.com/Aunsiels/pyformlang

Kapitola 8

Závěr

Cílem této práce bylo zavedení nového formálního jazykového systému nazývaného *vymazávací systém*. Hlavní motivací vytvoření uvedeného systému bylo studování formálního modelu, který se skákajícími vlastnostmi podobá *obecným skákajícím konečným automatům*. Na základě inspirace *regulovanými konečnými automaty* je ale využíván řídící regulární jazyk namísto stavového řízení.

Tato práce definuje samotný vymazávací systém, včetně jeho *konfigurace*, *vymazávacího kroku* či *přijímaného jazyka*. Zároveň odpovídá na otázku, jaká je jeho *přijímací síla* za použití jazykových rodin dle *Chomského hierarchie*. Dále poukazuje na jeho vztahy s *Dyckovými* a *polo-Dyckovými jazyky*. Velmi důležitou součástí této práce je prokázání *uzávěrových vlastností* vymazávacích systémů vůči běžně studovaným operacím v oblasti formální teorie jazyků. V uvedených oblastech bylo dosaženo překvapujících výsledků a prokázány rozdíly oproti nejobecnější variantě skákajících konečných automatů. Poslední úlohou teoretické části bylo studování vztahů rodiny jazyků přijímaných vymazávacími systémy s rodinou jazyků *zamíchání*.

Druhou, stejně důležitou část této práce, představuje výzkum užití nového formálního systému na reálné problémy a návrhy jeho aplikací. Podařilo se nalézt uplatnění především v oblasti bioinformatiky pro molekulární biologii, a to při práci se sekvencemi jako je DNA, RNA či sekundární struktura RNA. Po určitých úpravách navrženého algoritmu lze vymazávací systém využít také při práci s proteiny tvořenými sekvencemi aminokyselin. Vymazávací systémy nalezly uplatnění i v lingvistické oblasti, kde se za použití Dyckových jazyků využívají na problémy řešené v textových editorech. Jako poslední, pro již více teoretickou aplikaci, byl navržen algoritmus pro ověření správnosti řešení známého problému n dam z oblasti kompozičního šachu s využitím zavedeného formálního modelu.

Veškeré navržené aplikace a algoritmy byly implementovány a tento text poskytuje popis algoritmu vymazávacího systému za použití dvou typů kvantifikátorů pro regulární výrazy definující regulární jazyk. Následně je popsána hierarchická struktura implementace a popis stěžejních algoritmických postupů.

Během výzkumu vlastností a užití vymazávacího systému byly také důkladně studovány procesy z oblasti molekulární biologie, a to *sestřih* RNA, ke kterému dochází během *transkripce* a také vytváření sekundárních struktur RNA. Při pokusech aplikace vymazávacího systému v daných oblastech nedisponoval systém potřebnými vlastnostmi. V prvním případě je zapotřebí se opřít o statistické modely a v druhém případě nedeterminismus skákajících vlastností nedovoloval řídit rozpoznávání jednotlivých známých sekundárních struktur.

Tato práce přináší otevřené problémy a nové oblasti studia formálních modelů. S přispěním pana prof. RNDr. Alexandera Meduny, CSc. jsou položeny následující otázky:

- Na základě překvapivých výsledků v oblasti uzávěrových vlastností především pro operaci sjednocení, vůči které není rodina jazyků přijímaných vymazávacími systémy uzavřena, lze uvažovat uzávěrové vlastnosti jako námět pro další studium. Jak by se změnily vlastnosti vymazávacího systému, pokud by jako řídící jazyk byl uvažován jiný než regulární jazyk? Například by bylo vhodné studovat, pokud by jako řídící jazyky byly využívány různé takzvané podregulární jazyky. Jaká by byla rodina jazyků přijímaná takovými vymazávacími systémy?

Pan Ing. Zbyněk Křivka, Ph.D. přispěl několika dalšími otevřenými problémy:

- Jaký je vztah rodiny jazyků přijímaných vymazávacími systémy (**ES**) a rodiny jazyků přijímaných obecnými skákajícími konečnými automaty (**GJFA**). Platí, že **ES** $\subset$ **GJFA**?

- Jaké jsou vzájemné vlastnosti vymazávacích systémů a obecných skákajících automatů, které využívají pouze pravé či pouze levé skoky?

Dále lze uvažovat takový vymazávací systém, kde by byly jeho skoky dále řízeny tak, že bude určeno, zda následující skok se provede doprava nebo doleva od současné pozice speciálního symbolu (návěští). Proto kromě řídícího regulárního jazyku může být uvažován ještě jazyk, který řídí pořadí skoků (pravé či levé skoky). Jak by se změnily vlastnosti takového vymazávacího systému?

V další řadě lze poukázat na následující téma zkoumání vlastností vymazávacích systémů, a to na studium rozhodnutelnosti některých problémů.

Na základě aktuálnosti tématu aplikace formálních systému v oblasti bioinformatiky pro mikrobiologii by bylo vhodné položit si otázku, zda lze navrhnout takový vymazávací systém, který by dokázal v sekundární struktuře RNA rozpoznávat, zda obsahuje jednotlivé podstruktury, jako například *pseudouzly*, *vnitřní smyčky* a další.

Tato práce byla nominována a výsledky prezentovány na studentské konferenci inovací, technologií a vědy v IT — Excel@FIT2023.

Literatura

[1] Alberts, B., Bray, D., Johnson, A., Lewis, J., Raff, M. et al. *Základy buněčné biologie: Úvod do molekulární biologie buňky.* 1. vyd. Ústí nad Labem: Espero Publishing, 1998. ISBN 80-902906-0-4.

[2] Bradford, P. G. Efficient exact paths for dyck and semi-dyck labeled path reachability (extended abstract). In: IEEE. *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON).* 1. vyd. říjen 2017, abs/1802.05239, s. 247–253. ISBN 978-1-5386-1104-3.

[3] Campbell, P. J. Gauss and the eight queens problem: A study in miniature of the propagation of historical error. *Historia Mathematica.* 1. vyd. 1977, sv. 4, č. 4, s. 397–404. ISSN 0315-0860.

[4] Chomsky, N. Three models for the description of language. *IRE Transactions on Information Theory.* 1. vyd. 1956, sv. 2, č. 3, s. 113–124.

[5] Chomsky, N. On certain formal properties of grammars. *Information and Control.* 1. vyd. 1959, sv. 2, č. 2, s. 137–167. ISSN 0019-9958.

[6] Daintith, J. a Wright, E. *A Dictionary of Computing.* 7. vyd. Oxford University Press, 2008. 592 s. ISBN 9780191726576.

[7] Dealy, S. *Common Search Strategies and Heuristics With Respect to the N-Queens Problem: CS504 Term Project.* Albuquerque, 2004. Disertační práce. The University of New Mexico, UNM Computer Science.

[8] Diestel, R. *Graph Theory.* 5. vyd. Heidelberg: Springer-Verlag, 2016. 447 s. Graduate Texts in Mathematics. ISBN 978-3-662-53621-6.

[9] Ehrenfeucht, A., Haussler, D. a Rozenberg, G. On regularity of context-free languages. *Theoretical Computer Science.* 1. vyd. 1983, sv. 27, č. 3, s. 311–332. ISSN 0304-3975. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982.

[10] Fernandez Castillo, E., Barbosa Santillán, L. I., Falcon Morales, L. a Sánchez Escobar, J. J. Deep Splicer: A CNN Model for Splice Site Prediction in Genetic Sequences. *Genes.* 1. vyd. Květen 2022, sv. 13, č. 5, s. 907. ISSN 2073-4425.

[11] Fernau, H., Paramasivan, M., Schmid, M. L. a Vorel, V. Characterization and complexity results on jumping finite automata. *Theoretical Computer Science.* 1. vyd. 2017, sv. 679, C, s. 31–52. ISSN 0304-3975. Implementation and Application of Automata.

[12] Gathen, J. von zur a Gerhard, J. *Modern Computer Algebra*. 3. vyd. Cambridge: Cambridge University Press, 2013. ISBN 9781139856065.

[13] Goyvaerts, J. a Levithan, S. *Regular Expressions Cookbook: Second Edition*. 2. vyd. Sebastopol: O'Reilly Media, Inc., srpen 2012. 594 s. ISBN 978-1-449-31943-4.

[14] Gross, J. L. a Yellen, J. *Graph Theory and Its Applications: Second Edition*. 2. vyd. Boca Raton, London, New York: Chapman & Hall/CRC, 2005. Discrete Mathematics and Its Applications. ISBN 978-1-58488-505-4.

[15] Han, K. a Byun, Y. PseudoViewer2: visualization of RNA pseudoknots of any type. *Nucleic Acids Research*. 1. vyd. Červenec 2003, sv. 31, č. 13, s. 3432–3440. ISSN 0305-1048.

[16] Kari, L. *On Insertion and Deletion in Formal Languages*. Turku, Finland, 1991. Disertační práce. University of Turku, Faculty of Mathematics and Natural Sciences. ISBN 952-90-3386-9.

[17] Kernighan, B. W. a Ritchie, D. M. *Programovací jazyk C*. 2. vyd. Brno: Computer Press, 2019. ISBN 978-80-251-4965-2.

[18] McHugh, J. A. *Algorithmic Graph Theory*. 1. vyd. USA: Prentice-Hall, Inc., 1989. 327 s. ISBN 978-0-13-023615-9.

[19] Meduna, A. *Automata and languages: theory and applications*. 1. vyd. London: Springer-Verlag, 2000. 916 s. ISBN 1-85233-074-0.

[20] Meduna, A. a Zemek, P. *Regulated Grammars and Automata*. 1. vyd. New York: Springer, 2014. 694 s. Computer science. ISBN 978-1-4939-0368-9.

[21] Paun, G., Rozenberg, G. a Salomaa, A. *DNA Computing: New Computing Paradigms*. 1. vyd. Berlin: Springer-Verlag, 1998. 402 s. Texts in theoretical computer science an EATCS series. ISBN 3-540-64196-3.

[22] Quadrini, M., Tesei, L. a Merelli, E. An algebraic language for RNA pseudoknots comparison. *BMC Bioinformatics*. 1. vyd. Duben 2019, sv. 20, Suppl 4. ISSN 1471-2105.

[23] Romero, J. Pyformlang: An Educational Library for Formal Language Manipulation. In: Sherriff, M., Merkle, L. D., Cutter, P. A., Monge, A. E. a Sheard, J., ed. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. New York: Association for Computing Machinery, 2021, s. 576–582. SIGCSE '21. ISBN 9781450380621.

[24] Rozenberg, G. a Salomaa, A., ed. *Handbook of Formal Languages: Volume 1. Word, Language, Grammar*. 1. vyd. Berlin, Heidelberg: Springer-Verlag, 1997. 873 s. ISBN 978-3-642-63863-3.

[25] Russell, S. a Norvig, P. *Artificial Intelligence: A Modern Approach. Third Edition*. 3. vyd. USA: Prentice Hall, 2010. 1132 s. ISBN 0-13-604259-7.

[26] Stroustrup, B. *The C++ Programming Language: Fourth Edition*. 4. vyd. Ann Arbor: Addison-Wesley, 2013. 1346 s. ISBN 978-0-321-56384-2.

[27] Vorel, V. On Basic Properties of Jumping Finite Automata. *International Journal of Foundations of Computer Science*. 1. vyd. World Scientific. Leden 2018, sv. 29, č. 01, s. 1–15. ISSN 0129-0541.

[28] Wood, D. *Theory of Computation*. 1. vyd. New York: Harper & Row, Publishers, Inc., 1987. 558 s. Computer Science and Technology Series. ISBN 0-06-047208-1.

[29] Yin, M. M. a Wang, J. T. Effective hidden Markov models for detecting splicing junction sites in DNA sequences. *Information Sciences*. 1. vyd. 2001, sv. 139, č. 1, s. 139–163. ISSN 0020-0255.

Příloha A

Seznam vybraných zkratek a značení

CFA	úplný konečný automat (complete finite automaton)
CFG	bezkontextová gramatika (context-free grammar)
CJFA	úplný skákající konečný automat (complete jumping finite automaton)
CSG	kontextová gramatika (context-sensitive grammar)
DFA	deterministický konečný automat (deterministic finite automaton)
DGSM	deterministický zobecněný sekvenční stroj (deterministic generalized sequential machine)
DJFA	deterministický skákající konečný automat (deterministic jumping finite automaton)
ES	vymazávací systém (erasing system)
FA	konečný automat (finite automaton)
FT	konečný převodník (finite transducer)
GFA	obecný konečný automat (general finite automaton)
GJFA	obecný skákající konečný automat (general jumping finite automaton)
GSM	zobecněný sekvenční stroj (generalized sequential machine)
JFA	skákající konečný automat (jumping finite automaton)
LG	lineární gramatika (linear grammar)
PSG	frázová gramatika (phrase-structure grammar)
RG	regulární gramatika (regular grammar)

CF	rodina jazyků generovaná bezkontextovými gramatikami
CS	rodina jazyků generovaná kontextovými gramatikami
$\boldsymbol{\mathcal{D}}$	rodina jazyků generovaná gramatikami pro Dyckovy jazyky

DJFA	rodina jazyků přijímaná deterministickými skákajícími konečnými automaty
ES	rodina jazyků přijímaná vymazávacími systémy
FIN	rodina konečných jazyků
GJFA	rodina jazyků přijímaná obecnými skákajícími konečnými automaty
$\mathbf{GJFA}^{-\varepsilon}$	rodina jazyků přijímaná obecnými skákajícími konečnými automaty bez ε-přechodů
JFA	rodina jazyků přijímaná skákajícími konečnými automaty
$\mathbf{JFA}^{-\varepsilon}$	rodina jazyků přijímaná skákajícími konečnými automaty bez ε-přechodů
LIN	rodina jazyků generovaná lineárními gramatikami
RE	rodina rekurzivně spočetných jazyků
REG	rodina jazyků generovaná regulárními gramatikami
$\boldsymbol{\mathcal{S}}$	rodina jazyků generovaná gramatikami pro polo-Dyckovy jazyky
SHUF	rodina jazyků generovaná výrazy zamíchání
$\mathbf{UC}_n$	rodina kompozic (jazyků) stupně n
UC	sjednocení kompozic (union of compositions)

DNA	deoxyribonukleová kyselina (deoxyribonucleic acid)
GCD	největší společný dělitel (greatest common divisor)
HMM	skryté Markovovy modely (hidden Markov models)
NN	neuronové sítě (neural networks)
RNA	ribonukleová kyselina (ribonucleic acid)

Příloha B

Uzávěrové vlastnosti jazykových rodin dle Chomského hierarchie

Uzávěrové vlastnosti jednotlivých jazykových rodin dle Chomského hierarchie jsou vyznačeny v tabulce B.1.

	RE	**CS**	**CF**	**LIN**	**REG**
Sjednocení	+	+	+	+	+
Průnik	+	+	−	−	+
Doplněk	−	+	−	−	+
Konkatenace	+	+	+	−	+
Kleeneho hvězdička	+	+	+	−	+
Průnik s regulárními jazyky	+	+	+	+	+
Substituce	+	−	+	−	+
Substituce bez ε	+	+	+	−	+
Homomorfismus	+	−	+	+	+
Homomorfismus bez ε	+	+	+	+	+
Inverzní homomorfismus	+	+	+	+	+
Levý / pravý kvocient	+	−	−	−	+
Levý / pravý kvocient s regulárními jazyky	+	−	+	+	+
Levá / pravá derivace	+	+	+	+	+
Zamíchání	+	+	−	−	+
Reverzace	+	+	+	+	+

Tabulka B.1: Uzávěrové vlastnosti jazykových rodin z Chomského hierarchie. Převzato z [24].

Příloha C

Genetický kód

Cílem této přílohy je poskytnou vztahy převodu trojic nukleotidů z mRNA na jednotlivé aminokyseliny dle *genetického kódu*. Tabulka C.1 prezentuje kodony (trojice nukleotidů) pro jednotlivé aminokyseliny.

Aminokyseliny a jejich symboly			**Kodony**					
Alanin	Ala	A	GCA	GCC	GCG	GCU		
Arginin	Arg	R	AGA	AGG	CGA	CGC	CGG	CGU
Asparagová kyselina	Asp	D	GAC	GAU				
Asparagin	Asn	N	AAC	AAU				
Cystein	Cys	C	UGC	UGU				
Glutamová kyselina	Glu	E	GAA	GAG				
Glutamin	Gln	Q	CAA	CAG				
Glycin	Gly	G	GGA	GGC	GGG	GGU		
Histidin	His	H	CAC	CAU				
Isoleucin	Ile	I	AUA	AUC	AUU			
Leucin	Leu	L	UUA	UUG	CUA	CUC	CUG	CUU
Lysin	Lys	K	AAA	AAG				
Methionin	Met	M	AUG					
Fenylalanin	Phe	F	UUC	UUU				
Prolin	Pro	P	CCA	CCC	CCG	CCU		
Serin	Ser	S	AGC	AGU	UCA	UCC	UCG	UCU
Threonin	Thr	T	ACA	ACC	ACG	ACU		
Tryptofan	Trp	W	UGG					
Tyrosin	Tyr	Y	UAC	UAU				
Valin	Val	V	GUA	GUC	GUG	GUU		
Stop-kodony			UAA	UAG	UGA			

Tabulka C.1: Genetický kód. Podle konvence 5′-konec nukleotidové sekvence je vždy zapisován vlevo. Tři kodony nespecifikují žádnou aminokyselinu, místo toho mají roli terminačních kodonů (stop-kodony). Jeden kodon, AUG, funguje jednak jako iniciační kodon (signalizuje začátek translace) a také jako kodon, ke kterému je přiřazen methionin. Další kodon, UGA, funguje buď jako terminační nebo kóduje *selenocystein* (zkráceně *Sec* nebo *U*), pokud se vyskytuje v sousedství určitých nukleotidů [1]. V této práci se uvažuje základní varianta genetického kódu, a tudíž dle genetického kódu uvedeného v [1] kodon UGA slouží pouze jako terminační kodon. Převzato z [1].

Příloha D

Základní a diagonální souřadnice šachovnice

Tato příloha vyobrazuje základní, levé a pravé diagonální souřadnice šachovnice. Pro šachovnici s n^2 poli pro dané n je celkem $2 * n - 1$ diagonálních souřadnic pro každou stranu. V této příloze je uveden příklad s výpočty diagonálních souřadnic pro vybrané řešení problému n dam.

Při převodu mezi souřadnicovými systémy je nutné převést souřadnice jednotlivých dam na šachovnici ze souřadnic na horizontálních a vertikálních hranách (viz obrázek D.1) na souřadnice diagonální. Základní souřadnice se uvažují již po převodu číselných hodnot označení na znaky řecké abecedy, více viz kapitola 6, sekce 6.3. Mapování jednotlivých polí šachovnice do levých diagonálních souřadnic je vyobrazeno na obrázku D.2a. Dále do pravých diagonálních souřadnic na obrázku D.2b.

Příklad D.0.1. Pro názornou demonstraci výpočtu převodu souřadnic figur dam ze základních souřadnic na souřadnice diagonální lze uvažovat řešení uvedené na obrázku 6.6 z kapitoly 6, sekce 6.3. Dané řešení v zakódovaném řetězci $w = 2a4b6c8d3e1f7g5h$ je pomocí definovaného konečného převodníku T_8 (viz kapitola 6, sekce 6.3, výraz 6.11) převedeno na řetězec $w' = \beta a \delta b \zeta c \theta d \gamma e \alpha f \eta g \epsilon h$.

Na základě přijetí řetězce pomocí vymazávacího systému ES_8 (viz kapitola 6, sekce 6.3, výraz 6.12) následuje převod základních souřadnic šachovnice na souřadnice diagonální. To lze pomocí vztahu 6.13 a vztahu 6.14, z kapitoly 6, sekce 6.3, provést následovně. Pro označení symbolů ve vstupním řetězci bude dále uvažováno, že proměnná k značí pozici dvojice symbolů ve vstupním řetězci w' při číslování od nuly. Tedy $k = 0$ značí dvojici βa, $k = 1$ dvojici δb, a tak dále. Pro daný vstupní řetězec jsou nové souřadnice vypočítány následovně:

1. $k = 0$, dvojice βa:

$$g = n + i - j - 1 = 8 + 1 - 0 - 1 = 8 \Rightarrow \iota, \tag{D.1}$$
$$e = i + j = 1 + 0 = 1 \Rightarrow b. \tag{D.2}$$

2. $k = 1$, dvojice δb:

$$g = n + i - j - 1 = 8 + 3 - 1 - 1 = 9 \Rightarrow \kappa, \tag{D.3}$$
$$e = i + j = 3 + 1 = 4 \Rightarrow e. \tag{D.4}$$

3. $k = 2$, dvojice ζc:

$$g = n + i - j - 1 = 8 + 5 - 2 - 1 = 10 \Rightarrow \lambda, \tag{D.5}$$
$$e = i + j = 5 + 2 = 7 \Rightarrow h. \tag{D.6}$$

4. $k = 3$, dvojice θd:

$$g = n + i - j - 1 = 8 + 7 - 3 - 1 = 11 \Rightarrow \mu, \tag{D.7}$$
$$e = i + j = 7 + 3 = 10 \Rightarrow k. \tag{D.8}$$

5. $k = 4$, dvojice γe:

$$g = n + i - j - 1 = 8 + 2 - 4 - 1 = 5 \Rightarrow \zeta, \tag{D.9}$$
$$e = i + j = 2 + 4 = 6 \Rightarrow g. \tag{D.10}$$

6. $k = 5$, dvojice αf:

$$g = n + i - j - 1 = 8 + 0 - 5 - 1 = 2 \Rightarrow \gamma, \tag{D.11}$$
$$e = i + j = 0 + 5 = 5 \Rightarrow f. \tag{D.12}$$

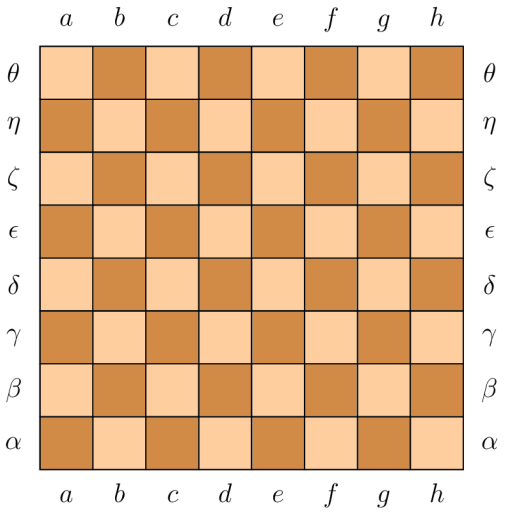
7. $k = 6$, dvojice ηg:

$$g = n + i - j - 1 = 8 + 6 - 6 - 1 = 7 \Rightarrow \theta, \tag{D.13}$$
$$e = i + j = 6 + 6 = 12 \Rightarrow m. \tag{D.14}$$

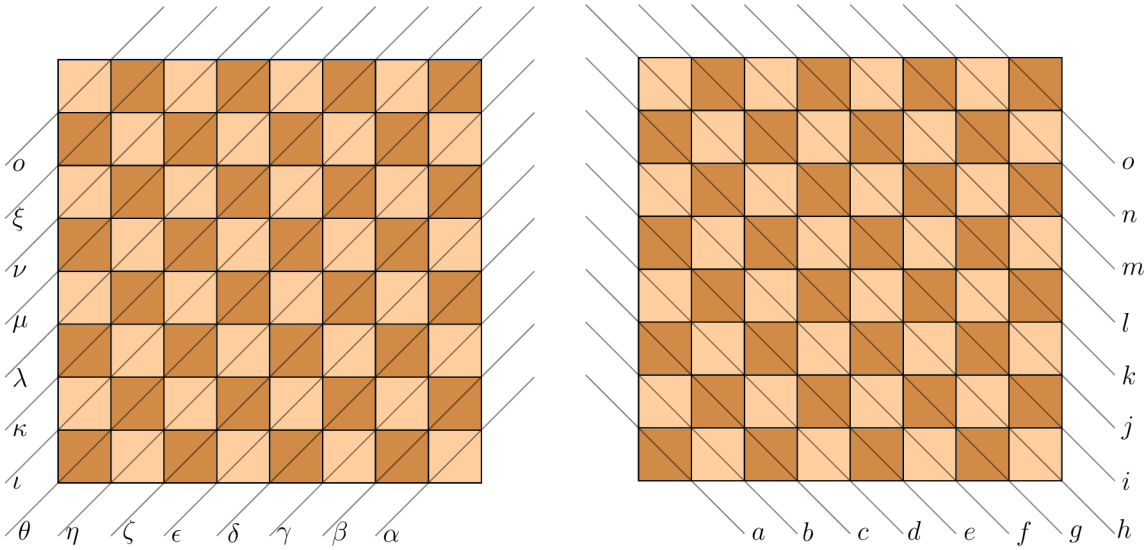
8. $k = 7$, dvojice ϵh:

$$g = n + i - j - 1 = 8 + 4 - 7 - 1 = 4 \Rightarrow \epsilon, \tag{D.15}$$
$$e = i + j = 4 + 7 = 11 \Rightarrow l. \tag{D.16}$$

Vstupní řetězec $w' = \beta a \delta b \zeta c \theta d \gamma e \alpha f \eta g \epsilon h$ v horizontálně-diagonálních souřadnicích je po převodu do diagonálních souřadnic zapsán jako řetězec $w'' = \iota b \kappa e \lambda h \mu k \zeta g \gamma f \theta m \epsilon l$.



Obrázek D.1: Základní souřadnice šachovnice.



(a) Levé diagonální souřadnice.

(b) Pravé diagonální souřadnice.

Obrázek D.2: Šachovnice v diagonálních souřadnicích.

Příloha E

Algoritmus činnosti vymazávacího systému

Cílem této přílohy je prezentace důkladného popisu navrženého algoritmu činnosti vymazávacího systému včetně jeho zápisu s využitím pseudokódu.

Pro dodržení formálních definic uvažují uvedené zápisy algoritmu návrh vymazávacího systému, jenž byl představen v kapitole 5, dle definic 5.1.1, 5.1.2 a 5.1.3. Proto vstupní řetězec je definovaný nad $(\Sigma - \{\#\})^*$. Podobně je tomu tak pro *konfiguraci*, množinu *vymazávacích řetězců* či *regulární jazyk*. Pro podrobnější popis je čtenář odkázán na uvedené definice. Při samotném návrhu algoritmů a programového řešení již není uvažována práce se speciálním symbolem, #, v té podobě, že je přímo zapisován na vstupní pásku.

Koncept hlavního toku algoritmu nejprve představuje výpis E.1, jehož transformaci do pseudokódu zachycuje algoritmus 1.

Nechť $ES = (\Sigma, E, R)$ je vymazávací systém. Dále se uvažuje řetězec $v = \varepsilon \in (\Sigma - \{\#\})^*$ a vstupní řetězec w, $w \in (\Sigma - \{\#\})^*$. Nechť proměnná otestování_všech_pozic obsahuje pravdivostní hodnotu a definuje, zda mají být vyhodnocovány všechny pozice, i překrývající se, vymazávacího řetězce na vstupní pásce. Je zavolána funkce je_přijat() s řetězci w a v po řadě jako parametry, zapsáno je_přijat(w, v).

je_přijat(w', v'):

1. Pokud $w' = \varepsilon$:
   a. Pokud $v' \in R$:
      A. Navrátí se $w \in L(E, R)$.
   b. Jinak:
      B. Navrátí se $w \notin L(E, R)$.

2. Nechť U značí množinu všech aplikovatelných vymazávacích řetězců, přičemž je stanoveno $U = \emptyset$.

3. Jsou nalezeny všechny vymazávací řetězce u, $u \in E$, přičemž pro každé u existují řetězce $x, y, z \in (\Sigma - \{\#\})^*$ takové, že $w' = xuy$ a $v'uz \in R$. Neformálně řečeno řetězec z slouží pro vymezení, že pokud bude konkatenován příslušný řetězec u k řetězci v', stále existuje řetězec definovaný nad $(\Sigma - \{\#\})^*$ takový, že výsledný řetězec vzniklý konkatenací použitých vymazávacích řetězců bude náležet do řídícího regulárního jazyka

R. Všechny nalezené řetězce u, které odpovídají definovaným pravidlům, jsou vloženy do množiny U, $U = U \cup \{u\}$ pro každé u.

4. Opakuje se, dokud $U \neq \emptyset$:
   a. Nedeterministicky je zvolen jeden z vymazávacích řetězců u', $u' \in U$.

   b. Dále se vyhledají všechny, i překrývající se, výskyty zvoleného vymazávacího řetězce u' v řetězci w'. Je uvažována množina $P = \emptyset$, která značí množinu obsahující všechny pozice výskytů u' ve w. Uvedená pozice je značena jako p. Všechny nalezené výskyty se vloží do množiny P, přičemž $P = P \cup \{p\}$ pro každé p.

   c. Opakuje se, dokud $P \neq \emptyset$:
      A. Nedeterministicky je zvolena jedna pozice vymazávacího řetězce u' ve vstupním řetězci w', značena p'.

      B. Zvolený výskyt p' vymazávacího řetězce u' se vymaže z řetězce w' pro $w' = xu'y$, kde $x, y \in (\Sigma - \{\#\})^*$. Dále se jako řetězec w'' uvažuje nově vzniklý řetězec xy, $w'' = w' = xy$.

      C. Vymazaný řetězec u' je konkatenován k řetězci v', který obsahuje konkatenaci dříve vymazaných vymazávacích řetězců, přičemž jako nový řetězec v'' se dále uvažuje nově vzniklý řetězec $v'u'$, $v'' = v'u'$.

      D. S novými hodnotami řetězců w' a v', tedy s w'' a v'', zavolá funkce je_přijat() sama sebe, zapsáno je_přijat(w'', v'').

      E. Pokud je navráceno $w \in L(E, R)$:
         i. Navrátí se $w \in L(E, R)$.

      F. Jinak pokud proměnná otestování_všech_pozic obsahuje kladnou pravdivostní hodnotu:
         i. $P = P - \{p'\}$.

      G. Jinak:
         i. $P = \emptyset$.

   d. Množina U se stanoví jako množina $U - \{u'\}$, tudíž $U = U - \{u'\}$.

5. Navrátí se $w \notin L(E, R)$.

Výpis E.1: Popis hlavního toku algoritmu pro činnost vymazávacího systému.

Algoritmus 1 Algoritmus činnosti vymazávacího systému přijímajícího jazyk $L(E, R)$.

Input: input string $w \in \overline{\Sigma}^*$, $v = \varepsilon \in \overline{\Sigma}^*$, $w' = w \in \overline{\Sigma}^*$ and $v' = v \in \overline{\Sigma}^*$, where $\overline{\Sigma} = \Sigma - \{\#\}$, $test\_all\_positions \in \{True, False\}$.

Output: *True* if and only if $w \in L(E, R)$, *False* otherwise.

```
1:  procedure IS_ACCEPTED(w', v')
2:      if w' = ε then
3:          if v' ∈ R then
4:              return True
5:          else
6:              return False
7:      U ← ∅
8:      for all u ∈ E do
9:          if w' = xuy and v'uz ∈ R and x, y, z ∈ (Σ − {#})* then
10:             U ← U ∪ {u}
11:     while U ≠ ∅ do
12:         u' ← random(U)
13:         P ← all_overlapping_positions(w', u')
14:         while P ≠ ∅ do
15:             p' ← random(P)
16:             w'' ← erase(w', p')
17:             v'' ← v'u'
18:             if IS_ACCEPTED(w'', v'') then
19:                 return True
20:             else if test_all_positions = True then
21:                 P ← P − {p'}
22:             else
23:                 P ← ∅
24:         U ← U − {u'}
        return False
```

Příloha F

Obsah přiloženého paměťového média

Přiložené médium obsahuje následující adresáře a soubory:

- `thesis_text/` — adresář obsahující zdrojové soubory textu práce a její verze v `pdf`:
  - `thesis_pdf` — adresář obsahující obě verze textu práce v `pdf`,
  - `thesis_src` — adresář obsahující zdrojové soubory textu práce v LaTeX a další soubory potřebné k jejímu vygenerování,
- `src/` — zdrojové soubory, programová dokumentace, testovací data pro ukázkové příklady a soubory potřebné k instalaci balíčku a vygenerování dokumentace:
  - `es-tools` — adresář, který je kořenovým adresářem programového řešení,
- `excel_fit/` — adresář obsahující veškeré materiály použité pro studentskou konferenci Excel@FIT2023,
- `README.md` — soubor obsahující popis rozvržení dat do adresářů na paměťovém médiu a další doplňující informace.

Příloha G

Manuál k přiloženým ukázkovým příkladům

Tato příloha poskytuje manuál na instalaci balíčku a spuštění ukázkových příkladů pro jednotlivé nástroje. Pro spuštění jednotlivých nástrojů s vlastními parametry lze odkázat na přiloženou dokumentaci.

Dle klasického přístupu je možné ukázkové příklady spustit po instalaci balíčku. Pro demonstrační účely ovšem příslušný nástroj také nabízí spuštění ukázkového příkladu v režimu bez instalace.

G.1 Spuštění příkladu v režimu bez instalace balíčku

Pro spuštění ukázkového příkladu v režimu bez instalace je nutné nacházet se ve složce *examples*, jejíž cesta od kořenové složky projektu (*es-tools*) je `es-tools/es_tools/examples`. Dále se vyžaduje nainstalovaný jazyk *Python3*[1] verze *3.10* nebo vyšší. Následně je možné spustit příklad pomocí příkazu:

- Unix/macOS:
  `python3 run_example.py -t název_nástroje -n číselné_označení_testu -i`
- Windows:
  `py run_example.py -t název_nástroje -n číselné_označení_testu -i`

Parametr `-i` (v dlouhé variantě `--noinstall`) určuje, že má být ukázkový příklad spuštěn v režimu bez instalace.

Popis významu dalších parametrů nástroje a konkrétní ukázkový příklad blíže představuje sekce G.3.

G.2 Instalace balíčku a spuštění příkladu

Pro instalaci balíčku *es-tools* je nutné nacházet se v kořenové složce projektu — složka *es-tools*, která obsahuje soubor *pyproject.toml*. Pro sestavení je vyžadován nainstalovaný jazyk *Python3*[1] (pro spuštění programového řešení se požaduje verze *3.10* nebo vyšší) a nástroj

[1] https://www.python.org/

pip[2] (otestováno s verzí *22.0.2*). Dále se požadují balíčky *setuptools*[3] verze rovna nebo vyšší *43.0.0* (otestováno s verzí *58.1.0* a vyšší) a *wheel*[4] (otestováno s verzí *0.37.1* a vyšší). Samotnou instalaci lze provést následujícím příkazem (*upozornění: symbol „.“ je součástí příkazu a je nutné jej zadat*):

- Unix/macOS:
  `python3 -m pip install .`
- Windows:
  `py -m pip install .`

Po úspěšné instalaci lze přejít ke spuštění jednotlivých ukázkových příkladů. Nejprve je nutné nacházet se ve složce *examples*, jejíž cesta od kořenové složky projektu je `es-tools/es_tools/examples`. Následně je možné spustit příklad pomocí příkazu:

- Unix/macOS:
  `python3 run_example.py -t` *`název_nástroje`* `-n` *`číselné_označení_testu`*
- Windows:
  `py run_example.py -t` *`název_nástroje`* `-n` *`číselné_označení_testu`*

Popis významu jednotlivých parametrů nástroje a konkrétní ukázkový příklad blíže představuje sekce G.3.

G.3 Popis parametrů nástroje a demonstračního příkladu

Tato sekce blíže popisuje význam a funkcionalitu jednotlivých parametrů nástroje, který je určen pro spuštění ukázkového příkladu již pro konkrétní nástroj implementující aplikaci vymazávacího systému. Na závěr sekce je představen jeden konkrétní příklad pro aplikaci vymazávacího systému na ověření řešení problému n dam, kterou implementuje nástroj *n_queens_problem_tool*.

Parametry `-t` (v dlouhé variantě `--tool`) a `-n` (v dlouhé variantě `--number`) jsou povinné a značí po řadě název spouštěného nástroje a číslo spouštěného testu. Název nástroje může nabývat jedné z hodnot:

(i) amino_acid_sequence_search_tool,

(ii) balanced_brackets_tool,

(iii) n_queens_problem_tool,

(iv) secondary_structure_properties_tool,

(v) sequence_search_tool.

Číslo spouštěného testu nabývá hodnoty dle počtu testů pro daný nástroj, jenž lze nalézt v přiložené dokumentaci (většinou se pohybuje v rozsahu 1-10).

[2] https://pypi.org/project/pip/
[3] https://pypi.org/project/setuptools/
[4] https://pypi.org/project/wheel/

Volitelně lze využít parametr `-v` (v dlouhé variantě `--verbose`), jenž slouží pro výpis aktuálního tvaru vstupní pásky, jednotlivých použitých vymazávacích řetězců a ověřovaného řetězce vzniklého konkatenací doposud aplikovaných vymazávacích řetězců.

Posledním volitelným parametrem je parametr `-l` (v dlouhé variantě `--lazy`). Při zadání uvedeného parametru veškeré vymazávací systémy používají líný kvantifikátor. V opačném případě je využíván chamtivý kvantifikátor jako výchozí.

Lze uvažovat názorný příklad:

- Unix/macOS:
  `python3 run_example.py -t n_queens_problem_tool -n 1 -v -l`
- Windows:
  `py run_example.py -t n_queens_problem_tool -n 1 -v -l`

V uvedeném příkladu je spuštěn test číslo *1* pro nástroj *n_queens_problem_tool* při vypisování dodatečných informací pomocí parametru `-v` a použití líného kvantifikátoru pro vymazávací systémy pomocí parametru `-l`.

Otestováno na třech zařízeních s následujícími parametry:

(i) První zařízení:

- Operační systém: Ubuntu 22.04.1 LTS,
- Verze *Python3*: 3.10.6,
- Verze *pip*: 22.0.2,
- Verze *setuptools*: 59.6.0,
- Verze *wheel*: 0.37.1.

(ii) Druhé zařízení:

- Operační systém: Windows 10 Education 22H2,
- Verze *Python3*: 3.10.4,
- Verze *pip*: 22.0.4,
- Verze *setuptools*: 58.1.0,
- Verze *wheel*: 0.40.0.

(iii) Třetí zařízení:

- Operační systém: macOS High Sierra version 10.13.6,
- Verze *Python3*: 3.11.3,
- Verze *pip*: 23.0.1,
- Verze *setuptools*: 67.6.1,
- Verze *wheel*: 0.40.0.

Jednotlivé testy využívají data z několika databází, nástrojů a zdrojů, přičemž pro každý příklad je uveden zdroj vstupních dat. Z využitých databází lze vyjmenovat: *RNAcentral*[5],

[5] https://rnacentral.org/

PseudoBase++2.0 Database[6] a *bpRNA-1m*[7]. Pro generování sekvencí DNA byly využity nástroje *Random DNA Sequence*[8] a *Reverse Complement*[9]. Pro nástroj *balanced_brackets_-tool* byly použity příklady z [17, 26].

[6] https://rnavlab.utep.edu/database
[7] https://bprna.cgrb.oregonstate.edu/index.html
[8] https://www.bioinformatics.org/sms2/random_dna.html
[9] https://www.bioinformatics.org/sms/rev_comp.html