FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Algoritmy pro určování vlastností D0L systémů
Student:	Bc. Anežka Štěpánková
Vedoucí:	Ing. Karel Klouda, Ph.D.
Studijní program:	Informatika
Studijní obor:	Systémové programování
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

1) Seznamte se s teorií D0L systémů a s potřebnými pojmy z kombinatoriky na slovech.
2) Nastudujte si v doporučené literatuře algoritmy pro zjišťování vybraných vlastností D0L systémů: jde zejména o cirkularitu, repetitivitu a pushy D0L systémy.
3) Vybrané algoritmy co nejefektivněji implementujte v jazyce Python; ideálně v takovém formátu, aby bylo možné je zařadit do knihoven programu SageMath.
4) Pokuste se navrhnout a implementovat algoritmus pro (brute-force) ověřování injektivity D0L systému na jazyce.

Seznam odborné literatury

[1] Karel Klouda, Štěpán Starosta: Characterization of circular D0L systems, *Theoretical Computer Science* **790** (2019), 131-137;

[2] Karel Klouda, Štěpán Starosta: An Algorithm Enumerating All Infinite Repetitions in a D0L System, Journal of Discrete Algorithms, 33, (2015), 130–138;

[3] Karel Klouda: Bispecial factors in circular non-pushy D0L languages, Theoretical Computer Science, 445 (2012), 63-74;

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 7. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Algoritmy pro určování vlastností D0L systémů

Bc. Anežka Štěpánková

Katedra teoretické informatiky
Vedoucí práce: Ing. Karel Klouda, Ph.D.

16. září 2020

Poděkování

Děkuji vedoucímu práce Ing. Karlu Kloudovi, Ph.D., který mi vymyslel velmi zajímavé téma práce a v průběhu její tvorby byl trpělivý a se vším vždy dobře poradil. Děkuji také svým přátelům, kolegům a rodině, kteří na mne při psaní práce brali ohledy a se vším mi ochotně pomohli. Nakonec děkuji svému snoubenci Michalovi Černému za obrovskou trpělivost a podporu po celou dobu mého studia i psaní práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 16. září 2020 . . . . . .

Odkaz na tuto práci

Štěpánková, Anežka. *Algoritmy pro určování vlastností D0L systémů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Cílem této práce je v první řadě seznámení se se základními pojmy z kombinatoriky na slovech a s teorií D0L-systémů. Další část práce se zabývá nastudováním a porozuměním algoritmům pro zjišťování vybraných vlastností D0L-systémů, konkrétně jde o vlastnosti: pushy, injektivita, repetitivita a cirkularita. Tyto vybrané algoritmy implementovat v jazyce Python a následně pomocí nich zjistit tyto vlastnosti pro binární morfismy. Vyhodnocením výsledků vytvořit přehled vlastností testovaných binárních morfismů.

Klíčová slova kombinatorika na slovech, DOL-systém, morfismus, injektivita, pushy, repetitivita, cirkularita, synchronizační zpoždění

Abstract

The aim of this work is to present combinatorics on word and theory od D0L-systems. Further, to study and understand algorithms for determining selected properties of D0L-systems, namely: pushy, injectivity, repetitivity and circularity. Furthermore, to implement these selected algorithms in the language Python and then use them to find out these properties for binary morphisms and to evaluate the results of creating an overview of the properties of the tested binary morphisms.

Keywords combinatorics on word, DOL-system, morphism, injektivity, pushy, repetitivity, cirkularity, synchronization delay

Obsah

Úvod **1**

1 Teorie a základní pojmy **3**
1.1 Kombinatorika na slovech 3
1.1.1 Základní pojmy z kombinatoriky na slovech 3
1.1.2 Morfismus 6
1.2 D0L-systémy 8
1.3 Vlastnosti morfismů a D0L-systémů 10
1.3.1 Injektivita 10
1.3.2 Pushy 11
1.3.3 Repetitivita a neomezená repetitivita 12
1.3.4 Cirkularita a synchronizační zpoždění 13
1.3.4.1 Vztah mezi cirkularitou a neomezenou repetitivitou 16

2 Algoritmy a popis implementace **19**
2.1 Nalezení faktorů dané délky 19
2.1.1 Popis algoritmu na nalezení faktorů dané délky s ukázkou běhu algoritmu 19
2.1.2 Pseudokód algoritmu na nalezení faktorů dané délky . . 21
2.2 Injektivita 22
2.2.1 Popis algoritmu na rozhodnutí o injektivitě s ukázkou běhu algoritmu 22
2.2.1.1 Neinjektivní příklad 22
2.2.1.2 Injektivní příklad 23
2.2.2 Pseudokód algoritmu na injektivitu 24
2.3 Pushy 25
2.3.1 Popis algoritmu pushy s ukázkou běhu algoritmu 25
2.3.1.1 Non-pushy příklad 25

2.3.1.2 Pushy příklad 26
2.3.2 Pseudokód algoritmu pushy 27
2.4 Repetitivita 28
2.4.1 Popis algoritmu na repetitivitu s ukázkou běhu algoritmu 28
2.4.2 Pseudokód algoritmu na repetitivitu 29
2.5 Cirkularita 29
2.5.1 Popis algoritmu na cirkularitu s ukázkou běhu algoritmu 30
2.5.2 Pseudokód algoritmu na cirkularitu 30
2.6 Synchronizační zpoždění 30
2.6.1 Popis algoritmu na nalezení synchronizačního zpoždění a ukázka běhu algoritmu 31
2.6.2 Pseudokód algoritmu na nalezení synchronizačního zpoždění 33

3 Experiment **35**
3.1 Zadání experimentu 35
3.2 Generování dat 35
3.3 Generování výsledků 36
3.4 Vyhodnocení výsledků 38
3.4.1 Souhrn všech vlastností 38
3.4.2 Pushy 39
3.4.3 Injektivita 40
3.4.4 Repetitivita 42
3.4.5 Cirkularita 43

Závěr **51**

Literatura **53**

A Obsah přiloženého CD **55**

Seznam obrázků

1.1 Ukázka propojení kombinatoriky na slovech s jinými matematickými odvětvími. [5] 4
1.2 Ukázka jednotlivých kroků přepisování podle přepisovacích pravidel $a \to ab$ a $b \to a$ z počátečního axiomu a. 9
1.3 Ukázka grafů neomezených písmen: UL_G (vlevo) a UR_G (vpravo) pro pushy D0L-systém. 12
1.4 Ukázka grafů neomezených písmen: UL_G (vlevo) a UR_G (vpravo) pro non-pushy D0L-systém. 13
1.5 Dvě interpretace z definice cirkularity s $v_i = v'_j$.[3] 15
1.6 Dvě interpretace synchronizované na pozicích znázorněných tečkovanými čarami.[3] 15

2.1 Graf neomezených písmen zprava. 26
2.2 Graf neomezených písmen zleva. 26
2.3 Graf neomezených písmen zprava. 27
2.4 Graf neomezených písmen zleva. 27

3.1 Grafy neomezených písmen zleva (levý) a zprava (pravý) pro morfismus $a \to ba$, $b \to b$. 39
3.2 Grafy neomezených písmen zleva (levý) a zprava (pravý) pro morfismus $a \to abbbb$, $b \to b$. 39
3.3 Grafy neomezených písmen zleva (levý) a zprava (pravý) pro morfismus $a \to bbbabb$, $b \to b$. 39

Seznam tabulek

3.1 Přehled počtu morfismů s jednotlivými vlastnostmi. 38
3.2 Přehled neinjektivních morfismů s příslušnými faktory, které mají dva a více předků pro $n = 2, 3$. 40
3.3 Přehled neinjektivních morfismů s příslušnými faktory, které mají dva a více předků pro $n = 4$. 40
3.4 Přehled neinjektivních morfismů s příslušnými faktory, které mají dva a více předků pro $n = 5$. 41
3.5 Přehled neinjektivních morfismů s příslušnými faktory, které mají dva a více předků pro $n = 6$. 42
3.6 Přehled počtu repetitivních/nerepetitivních morfismů s příklady morfismů pro $n = 2, 3, 4$. 42
3.7 Přehled počtu repetitivních/nerepetitivních morfismů s příklady morfismů pro $n = 5, 6$. 43
3.8 Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 2$. . 43
3.9 Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 3$. . 44
3.10 Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 4$, $m = 1, 2, 3$. 45
3.11 Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 4, m = 4$. 46
3.12 Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 5, m = 1, 2, 3$. 47
3.13 Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 5, m = 4$. 48

3.14 Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 5, m = 5$. 49
3.15 Přehled cirkulárních morfismů s jejich synchronizačním zpožděním s informací o nejdelším nesynchronizovaném faktoru pro $n = 6$, $m = 1$. 49
3.16 Přehled cirkulárních morfismů s jejich synchronizačním zpožděním s informací o nejdelším nesynchronizovaném faktoru pro $n = 6, m = 2$. 50

Úvod

Kombinatorika na slovech je oblastí spojovanou s mnoha obory matematiky i informatiky a patří mezi stále zkoumané oblasti. L-systémy jsou variantou formálních gramatik využívající přepisovací pravidla. Byly původně vyvinuty pro modelování biologického vývoje, ale později našly využití i ve světě matematiky a informatiky. Jejich speciální variantou jsou D0L-systémy, kde zkratka D0L znamená: deterministický (D), bezkontextový (0) L-systém, tedy že každý symbol má pouze jedno přepisovací pravidlo a přepisovací pravidlo nezáleží na sousedních symbolech. Díky těmto vlastnostem je možné množinu přepisovacích pravidel zachytit jako morfismus. V [2], [3] a [4] byly definovány vlastnosti morfismů a D0L-systémů, konkrétně: pushy, injektivita, repetitivita a cirkularita.

V této práci představíme základní pojmy z oblasti kombinatoriky na slovech i D0L-systémů, které byli čerpány z [1], [2], [3] a [4]. Následně definujeme a vysvětlíme výše zmíněné vlastnosti, včetně návrhu algoritmů vedoucích ke zjištění daných vlastností pro konkrétní morfismus. Všechny tyto algoritmy byly implementovány v jazyce Python tak, aby konvence odpovídala knihovnám programu SageMath [16]. Všechny algoritmy byly otestovány pro vybrané binární morfismy na binární abecedě $\{a, b\}$. Výsledkem práce je tedy přehled implementovaných algoritmů a přehled testovaných binárních morfismů s uvedenými vlastnostmi.

Pokračováním práce by mohlo být testování implementovaných algoritmů na morfismech s větší abecedou či prozkoumání a implementování dalších zajímavých vlastností morfismů a D0L-systémů. Následné zařazení všech implementovaných algoritmů do knihoven programu SageMath.

Kapitola 1

Teorie a základní pojmy

Obsah práce spadá do oblasti kombinatoriky na slovech a zároveň jsou používány některé pojmy z teorie o L-systémech. První část této kapitoly představuje základní pojmy z kombinatoriky na slovech, které budeme později potřebovat. Druhá část obsahuje stručný úvod do L-systémů, speciálně je věnována pozornost D0L-systémům, a třetí část představuje přehled vybraných vlastností D0L-systémů, kterým se věnujeme i algoritmicky.

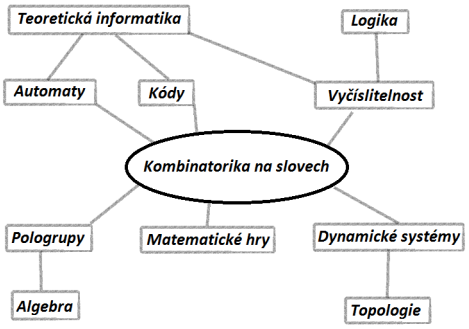
1.1 Kombinatorika na slovech

Kombinatorika na slovech je spojena s mnoha moderními i klasickými obory matematiky. Spojení s kombinatorikou, jež je její součástí, je zřejmé, nicméně i spojení s algebrou je velmi hluboké. [5] Vzájemné vztahy mezi kombinatorikou na slovech s dalšími matematickými odvětvími jsou patrné z obrázku 1.1 :

V této části déle představíme všechny potřebné základní pojmy a značení související s kombinatorikou na slovech a vše je demonstrováno na několika příkladech. Při definování pojmů se opíráme o [1], [2], [3] a [4].

1.1.1 Základní pojmy z kombinatoriky na slovech

- **Abeceda** – je konečná množina symbolů označována $\mathcal{A}$, někdy i $\mathcal{B}$.
- **Písmena** – prvky abecedy, které se obvykle označují buď nezápornými celými čísly 0,1,2,. . .nebo malými písmeny ze začátku abecedy $a, b, c, \ldots$.
- **Jazyk** – volný monoid na $\mathcal{A}$, označujeme $\mathcal{A}^*$.
- **Slova** – prvky jazyka. Konečná nebo doprava nekonečná (definujeme později) posloupnost písmen, kterou označujeme malým písmenem z konce abecedy, obvykle u, v nebo w.



Obrázek 1.1: Ukázka propojení kombinatoriky na slovech s jinými matematickými odvětvími. [5]

- **Prázdné slovo** – unikátní slovo délky 0, které je označováno ε. Prázdné slovo je neutrálním prvkem abecedy.
- **Množina všech konečných neprázdných slov** – označujeme $\mathcal{A}^+$. Platí tedy $\mathcal{A}^* = \mathcal{A}^+ \cup \{\varepsilon\}$.
- **Délka slova** – slovo $w = a_0a_1 \cdots a_{n-1}$, kde $n \in \mathbb{N}$, $a_i \in \mathcal{A}$, je konečné slovo nad abecedou $\mathcal{A}$ délky n a označujeme $|w|$.
- **Součin slov [6]** – je definován pomocí zřetězení slov. Tedy pokud $w = a_0a_1 \cdots a_{n-1}$ a $v = b_0b_1 \cdots b_{m-1}$ jsou slova, pak jejich součin wv je slovo

  $$wv = a_0a_1 \cdots a_{n-1}b_0b_1 \cdots b_{m-1}$$

  délky $n + m$. Součin slov je asociativní, to znamená, že pro všechna $u, v, w \in \mathcal{A}^+$ platí

  $$u(vw) = (uv)w.$$

  Prázdné slovo je vůči operaci zřetězení neutrální, pro libovolné slovo w tedy platí

  $$\varepsilon w = w\varepsilon = w.$$

- **Mocnina slova** – používáme ji ke zjednodušení zápisu: pro všechny $w \in \mathcal{A}^*$ a $n \in \mathbb{N}$ píšeme

  $$w^n = ww \cdots w.$$

- **Nekonečné slovo nad abecedou** $\mathcal{A}$ – je (doprava) nekončená posloupnost písmen z $\mathcal{A}$ označována malým tučným písmenem z konce abecedy, např. $\mathbf{u} = u_0u_1u_2 \cdots$, kde $u_i \in \mathcal{A}$. Pro všechny $w \in \mathcal{A}^*$ píšeme

  $$w^\omega = www \cdots .$$

  Množinu všech nekonečných slov nad abecedou $\mathcal{A}$ označujeme $\mathcal{A}^{\mathbb{N}}$.

- **Faktor, prefix a suffix** – slovo u je faktor w (resp. prefix, suffix) pokud existuje slovo x a y takové, že

$$w = xuy \text{ (resp.} w = uy, w = xu).$$

- **Jazyk slova** – množina všech faktorů slova w. Označujeme $L(w)$.

- **Mazání prefixu a suffixu** – můžeme definovat analogicky jako zřetězení. Nechť $w = v_1uv_2$ je konečné slovo pro nějaké $v_1, u, v_2 \in \mathcal{A}^*$. Potom slovo w bez prefixu v_1 označíme $v_1^{-1}w = uv_2$. Podobně w bez suffixu v_2 je slovo $wv_2^{-1} = v_1u$.

- **Periodické a aperiodické slovo** – nekonečné slovo **u** je *posléze periodické* pokud existují slova $v, w \in \mathcal{A}^*$ taková, že $u = vwww\cdots = vw^\omega$. Pokud navíc $v = \epsilon$, **u** je *periodické*. Každé nekonečné slovo, které není posléze periodické je *aperiodické*.

Příklad 1. *Uvažujme abecedu $\mathcal{A} = \{a, b, c\}$ obsahující tři písmena a, b, c. Příklady slov nad $\mathcal{A}$ jsou řetězce $v_1 = babcca$ délky $|v_1| = 6$ a $v_2 = bcbc$ délky $|v_2| = 4$. Zřetězení slov v_1 a v_2 je slovo $w = v_1v_2 = babccabcbc$ délky $|w| = 10$. Slovo v_2 může být také zapsáno jako $v_2 = (bc)^2$. Všechny se nazývají vlastní, pokud jsou rozdílné od w.*

Příklad 2. *Nechť $\mathcal{A} = \{0, 1\}$ je abeceda. Champernownovo slovo u_C je definováno jako zřetězení binárních reprezentací všech po sobě jdoucích nezáporných celých čísel:*

$$u_C = 011011100101111\cdots .$$

Příklad 3. *Nechť $w = bcca$ je slovo nad abecedou $\mathcal{A} = \{a, b, c\}$. Prefixy slova w jsou b, bc a bcc. Protože prázdné slovo ε může být prefix, suffix nebo také faktor, prefix w jsou také slova ε a $bcca$. Podobně suffixy w jsou slova $\varepsilon, a, ca, ccaabcca$. Jazyk w je množina*

$$L(w) = \{\varepsilon, b, c, a, bc, cc, ca, bcc, cca, bcca\}.$$

Příklad 4. *Očividně, jazyk Champernownova slova u_C (příklad 2) je množina všech konečných slov nad binární abecedou: $L(u_C) = \mathcal{A}^*$.*

Příklad 5. *Slovo $u_1 = 001001001\cdots = (001)^w$ je periodické. Slovo*

$$u_2 = 11110110110\cdots = 11(110)^w$$

je posléze periodické . Champernownovo slovo (příklad 2) $u_C = 011011100\cdots$ je aperiodické.

Příklad 6. *Uvažte slovo $w = bcca$ nad abecedou $\mathcal{A} = \{0, 1, 2\}$. Pokud $v_1 = b$ a $v_2 = ca$, potom $v_1^{-1}w = cca$, $wv_2^{-1} = bc$ a $v_1^{-1}wv_2^{-1} = c$.*

1.1.2 Morfismus

Slova (nekonečná) lze konstruovat různými způsoby. Jeden z velmi jednoduchých příkladů byl prezentován v příkladu 2. Bohatou rodinu nekonečných slov se zajímavými vlastnostmi lze nicméně získat též iterací speciálních zobrazení, tzv. *morfismy.*

- **Morfismus** – na volném monoidu $\mathcal{A}^*$ je zobrazení $\theta : \mathcal{A}^* \to \mathcal{A}^*$ takové, že pro všechny $u, v \in \mathcal{A}^*$ splňuje $\theta(uv) = \theta(u)\theta(v)$. Pro každé slovo $w = a_0 a_1 \cdots a_{n-1}$, $a_i \in \mathcal{A}$, tedy platí $\theta(w) = \theta(a_0)\theta(a_1)\cdots\theta(a_{n-1})$.

  Jakýkoli morfismus je zcela určen jeho obrazy písmen z abecedy; pokud známe slova $\theta(a)$ pro všechny $a \in \mathcal{A}$, morfismus θ je jednoznačně definován a můžeme vytvořit obraz jakéhokoli konečného slova nad $\mathcal{A}$. Tuto skutečnost používáme k rozšíření definice na nekonečná slova.

- **Obraz nekonečného slova** – Nechť $u = u_0u_1u_2\cdots$ je nekonečné slovo nad $\mathcal{A}$. Máme

  $$\theta(u) = \theta(u_0u_1u_2\cdots) = \theta(u_0)\theta(u_1)\theta(u_2)\cdots .$$

  Všimněme si, že morfismy lze definovat obecněji, protože není nutné mít obrazy a předobrazy ve stejné abecedě. Nechť $\mathcal{A}, \mathcal{B}$ jsou dvě abecedy. Morfismus z $\mathcal{A}^*$ do $\mathcal{B}^*$ je zobrazení: $\theta : \mathcal{A}^* \to \mathcal{B}^*$ takové, že pro všechny $u, v \in A^*$ splňuje

  $$\theta(uv) = \theta(u)\theta(v).$$

- **Nevymazávající morfismus** – je takový morfismus, pro který platí $\theta(a) \neq \epsilon$ pro všechna $a \in \mathcal{A}$.

- **Periodický bod morfismu** θ – nazýváme nekonečné slovo **u** nad abecedou $\mathcal{A}$ pokud $\theta^l(u) = u$ pro nějaké $l \in \mathbb{N}$. Speciálně, pokud $l = 1$, říkáme, že **u** je *pevný bod* θ.

Pokud existuje písmeno $a \in \mathcal{A}$ a slovo $u \in \mathcal{A}^+$ takové, že $\varphi(a) = au$ a navíc, pokud $\lim_{n\to\infty} |\varphi^n(a)| = +\infty$, potom φ se nazývá *prodloužitelné* na$a$. Je snadno vidět, že pro všechny $n \in \mathbb{N}$ máme

$$\varphi^n(a) = a\varphi(u)\varphi^2(u)\cdots\varphi^{n-1}(u).$$

Tedy, $\varphi^n(a)$ je prefix $\varphi^{n+1}(a)$ pro všechny $n \in N$ a protože $|\varphi^n(a)|$ jde k nekonečnu když $n \to +\infty$, sekvence $(\varphi(a))_{n\in N}$ konverguje k nekonečnému slovu označenému

$$\varphi^\omega(a) = a\varphi(u)\varphi^2(u)\cdots .$$

Toto nekonečné slovo je *pevným bodem* φ. Nevymazávající morfismus, který je

prodloužitelný na některém písmenu z abecedy je někdy nazýván *substitucí*. Nejznámější substituce na binární abecedě jsou substituce Thue-Morseova a Fibonacciho.

Příklad 7. *Morfismy jsou obvykle definovány svými obrazy písmen. Uvažme morfismus θ_1 nad binární abecedou $\{0,1\}$ definovaný jako*

$$\theta_1(0) = 1,$$

$$\theta_1(1) = 0.$$

Pak můžeme snadno najít obraz faktoru 011 pomocí morfismu θ_1:

$$\theta_1(011) = \theta_1(0)\theta_1(1)\theta_1(1) = 100.$$

Příklad 8. *Periodický bod morfismu θ_1 z příkladu 7 je například slovo $(01)^w = 010101\cdots$ od*

$$\theta^2(010101\cdots) = \theta(101010\cdots) = 010101\cdots .$$

Ukázka pevného bodu je v příkladu 9.

Příklad 9. *Thue-Morseova substituce φ_{TM} je definována jako*

$$\varphi_{TM}(0) = 01,$$

$$\varphi_{TM}(1) = 10.$$

Její pevný bod začínající písmenem 0, t.j. nekonečné slovo

$$u_{TM} = \varphi_{TM}^{\omega}(0) = 0110100110\cdots ,$$

se nazývá Thue-Morseovo slovo.

Příklad 10. *Fibonacciho slovo je jediný pevný bod substituce φ_F definované jako*

$$\varphi_F(0) = 01,$$

$$\varphi_F(1) = 0,$$

takže je to nekonečné slovo

$$u_F = \varphi_F^{\omega}(0) = 010010100100101\cdots .$$

Zdůrazněme, že jeden morfismus může mít několik pevných bodů začínajících různými písmeny. Jelikož mohou tyto pevné body určovat rozdílné jazyky, je k jednoznačné definici jazyků nutné specifikovat morfismus a písmeno. K řešení tohoto problému je použit koncept D0L-systémů jenž je popsán v kapitole 1.2.

Příklad 11. *Uvažme morfismus φ:*

$$\varphi(0) = 012,$$

$$\varphi(1) = 2,$$

$$\varphi(2) = 22,$$

$$\varphi(3) = 303$$

Tento morfismus obsahuje tři pevné body, začínající písmeny $0, 2, 3$, tj. nekonečná slova:

$$\varphi^\omega(0) = 012222\cdots$$

$$\varphi^\omega(2) = 22222222\cdots$$

$$\varphi^\omega(3) = 303012303\cdots .$$

Každé z těchto nekonečných slov má jiný jazyk.

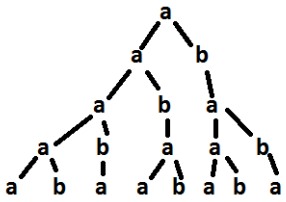
1.2 D0L-systémy

V souvislosti s tématem práce je třeba krátce zmínit i historii D0L-systémů. Základy L-systému položil biolog Aristid Lindenmayer. L-systémy jím byly tehdy vyvinuty jako matematický formalismus pro popis biologického vývoje. Na základě tehdejšího užití lze L-systém definovat jako množinu přepisovacích pravidel, podle kterých systém postupuje z daného počátečního stavu. V Lindenmayerově L-systému byly jednotlivé buňky prezentovány symboly a přepisovací pravidla simulovala jejich dělení. S odstupem času byl díky informatikovi Przemyslawu Prusinkiewiczkému implementován do vývoje počítačové grafiky. V této oblasti byly L-systémy užívány pro generování fraktálů a realistické modelování rostlin.[7]

Přepisovací systémy byly v minulosti podrobeny mnoha studiím. Nejlépe pochopené fungují na řetězcích znaků. Na zvýšeném zájmu o princip přepisovacích pravidel měla nezanedbatelný podíl práce lingvisty Noama Chomského z roku 1957[7], o formálních gramatikách. Zásadní rozdíl mezi prací Chomského a principem L-systémů je odlišný postup v aplikaci produkcí. Chomsky aplikoval produkce postupně, zatímco L-systémy je aplikují paralaleně, nahrazují tedy všechna písmena v daném slově současně. Z tohoto rozdílu je patrná původní biologická motivace L-systémů. [7]

D0L-systémy jsou nejjednodušší třídou L-systémů. Zkratka D0L znamená deterministický (D) bezkontextový (0) L-systém: každý symbol má pouze jedno přepisovací pravidlo a přepisovací pravidla nezávisí na sousedních symbolech. Základní myšlenka D0L-systémů, jak ji v roce 1991 uvedli Lindenmayer a Prusinkiewicz je demonstrována v následujícím příkladu.

Budeme uvažovat o řetězcích skládajících se ze dvou písmen a, b. Pro každé písmeno určíme přepisovací pravidlo $a \to ab$ a $b \to a$. Tedy každé písmeno a bude nahrazeno řetězcem ab a každé písmeno b bude nahrazeno řetězcem a. Proces přepisování vždy musí začít od nějakého počátečního řetězce, který nazýváme *axiom*. Zvolme za axiom písmeno a. Jednotlivé kroky přepisování jsou ilustrovány na obrázku 1.2 [7]:



Obrázek 1.2: Ukázka jednotlivých kroků přepisování podle přepisovacích pravidel $a \to ab$ a $b \to a$ z počátečního axiomu a.

Díky vlastnostem D0L-systémů, tedy determinismu a bezkontextovosti, je možné množinu přepisovacích pravidel zachytit jako morfismus, přičemž axiom představuje počáteční stav. Definovat D0L-systémy lze tedy následovně:

D0L-systém je trojice $D = (\mathcal{A}, \varphi, w)$, kde:

- $\mathcal{A}$ je abeceda,
- φ je morfismus na $\mathcal{A}$,
- w je slovo z $\mathcal{A}^+$ zvané axiom.

V souvislosti s D0L-systémy je třeba uvést několik základních pojmů, jež budou pro další vysvětlení problematiky potřebné. Definice těchto pojmů vychází z [2].

- **Jazyk D0L-systému** D – množina $L(D) = \{\varphi^n(w) | n \in N\}$.
- **Faktorový jazyk D0L-systému** D – množina $S(L(D)) = \{u | u$ je faktor z $\varphi^n(w), n \in \mathbb{N}\}$.
- **Konečný D0L-systém** D – D je konečný, pokud je $L(D)$ je konečný.

- **Omezená a neomezená písmena** – písmeno a je omezené pokud D0L-systém $(\mathcal{A}, \varphi, a)$ je konečný, jinak je neomezené. Množinu všech omezených písmen označujeme jako $\mathcal{A}_0$.
- **PD0L-systém** – nevymazávající D0L-systém. V této práci obvykle používáme PD0L-systém místo D0L-systému, tj. uvažujeme pouze nevymazávající morfismus. Ve skutečnosti je pro každý D0L-systém možné zkonstruovat v jistém smyslu ekvivalentní PD0L-systém s injektivní substitucí viz. [8].
- **Graf prvních písmen** – graf, jehož uzly jsou všechna písmena abecedy a hrana vždy vede písmene a do prvního písmene obrazu a. Díky tomu je možné zjistit, kolik je potřeba kroků k tomu, aby dané písmeno znovu zobrazilo samo na sebe, tedy lze pomocí něj najít periodické body.

Příklad 12. *Uvažujme abecedu $A = \{0, 1\}$, Thue-Morse substituce φ_{TM} (příklad 9.) a slovo 0. Potom systém $D_{TM} = (\mathcal{A}, \varphi_{TM}, 0)$ je příklad D0L-systému. Jazyk D_{TM} je množina*

$$L(D_{TM}) = \{0, 01, 0110, 01101001, 0110100110010110, \cdots\}$$

a faktorový jazyk D_{TM} je množina

$$S(L(D_{TM})) = \{\varepsilon, 0, 1, 00, 01, 10, 11, 001, 010, 011, 100, 101, 110, \cdots\}.$$

1.3 Vlastnosti morfismů a D0L-systémů

V této části práce jsou představeny vybrané vlastnosti morfismů a D0L-systémů. Veškeré definice, tvrzení, lemmata i věty se opírají o [3], [2], [4]. Všechny uvedené vlastnosti jsou vysvětleny a demonstrovány na příkladech. Vlastnostem uvedeným v této části se později věnujeme algoritmicky a dostatečný prostor je jim věnován i ve výsledcích prováděných experimentů. Jedná se o injektivitu, pushy, repetitivitu a cirkularitu se synchronizačním zpožděním.

1.3.1 Injektivita

Definice 1. *Nechť $\mathcal{U}$ je podmnožina $\mathcal{A}^*$. Morfismus φ nad $\mathcal{A}$ je injektivní na $\mathcal{U}$ pokud pro všechna $u, v \in \mathcal{U}$ platí následující implikace:*

$$\Big(\varphi(w) = \varphi(v)\Big) \Rightarrow w = v.$$

Možností volby množiny $\mathcal{U}$ je více. Můžeme například uvažovat o injektivitě na faktorovém jazyce $S(L(D))$ příslušného D0L-systému $D = (\mathcal{A}, \varphi, a)$. O této injektivitě uvažujeme v průběhu implementovaných algoritmů i výsledných

experimentech. Dále můžeme uvažovat o injektivitě na $\mathcal{A}^*$. Tato podmínka je přísnější. Navíc platí, že každý morfismus injektivní na $\mathcal{A}^*$ je injektivní i na $S(L(G))$. V opačném případě toto tvrzení neplatí.[4]

Definice 2. *Platí, že D0L-systém $D = (\mathcal{A}, \varphi, w)$ je injektivní pokud pro všechny $w, v \in S(L(D))$, $\varphi(w) = \varphi(v) \Rightarrow w = v$.*

Je tedy zřejmé, že pokud φ je injektivní, potom je injektivní i D. Obráceně to však není pravda.

Příklad 13. *Uvažme $\varphi : a \to abc, b \to bc, c \to a$, potom φ není injektivní, protože $\varphi(cb) = \varphi(a)$, ale $D = (\{a, b, c\}, , a)$ je injektivní, protože $cb \notin S(L(D))$.*

Není obtížné rozhodnout, zda je daný morfismus injektivní na $\mathcal{A}^*$, a to díky tvrzení uvedenému v [4]. Složitější je oproti tomu určení, zda je daný morfismus injektivní na $S(L(D))$. Není zatím znám žádný jiný algoritmus než použití hrubé síly. Hrubou silou je v tomto kontextu myšleno postupné procházení všech faktorů a jejich kontrola – tedy že má každý z nich pouze jeden předobraz, aby byla splněna podmínka injektivity. Existují-li pro nějaký faktor dva a více předobrazů, pak je zadaný D0L-systém neinjektivní. Nalezení takového faktoru však může být záležitostí časově velmi náročnou, neboť může být libovolně dlouhý.

1.3.2 Pushy

Definice 3. *D0L-systém $D = (\mathcal{A}, \varphi, w)$ je pushy, pokud pro všechna $n \in \mathbb{N}$ existuje slovo $v \in S(L(D))$ délky n, které obsahuje pouze omezená písmena. V opačném případě říkáme, že D není pushy.*

Příklad 14. *Uvažme D0L-systém $D = (\{a, b, c\}, \varphi_D, a)$ kde φ_D je definováno pomocí přepisovacích pravidel $a \to bcc$, $b \to ab$, $c \to c$. Písmeno c je omezené a písmena a a b jsou neomezená. Protože faktorový jazyk $S(L(D))$ neobsahuje faktory c^k, $k \geq 3$, systém D je non-pushy.*

Uvažme D0L-systém $E = (\{a, b, c, d\}, \varphi_E, a)$ kde φ_E je definováno pomocí $a \to ab$, $b \to cd$, $c \to d$, $d \to c$. Písmena b, c, d jsou omezená neboť $\{\varphi^n(b) | n \in N\} = \{cd, dc\}$, $\{\varphi^n(c) | n \in N\} = \{d\}$ a $\{\varphi^n(d) | n \in N\} = \{c\}$. Jazyk $L(E)$ začíná jako

$$a \to ab \to abcddc \to abcddccd \to abcddccddc \to \cdots .$$

Je jednoduché ověřit, že faktorový jazyk $S(L(E))$ obsahuje faktory $(cddc)^n$ pro všechny $n \in \mathbb{N}$ a proto je systém E pushy.

Ověření, zda je konkrétní D0L-systém pushy, lze provést též pomocí grafů neomezených písmen.[2]

Definice 4. *Nechť $G = (\mathcal{A}, \varphi, w)$ je D0L-systém. Graf neomezených písmen z G z prava, je orientovaný graf označený UR_G a definovaný takto:*

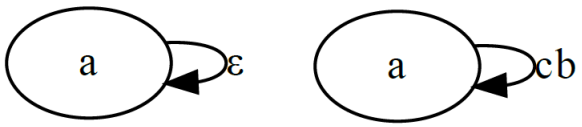
(i) množina vrcholů je $V(UR_G) = \mathcal{A} \setminus \mathcal{A}_0$,

(ii) existuje orientovaná hrana z a do b s označením $u \in \mathcal{A}_0^*$, pokud existuje $v \in \mathcal{A}^*$ takové, že $\varphi(a) = vbu$ pro nějaké $u \in \mathcal{A}^*$.

Graf neomezených písmen z G z leva UL_G je definován analogicky: jediný rozdíl je, že role v a u v definici orientované hrany se vymění.

Platí, že D0L-systém G je pushy tehdy a jen tehdy, když jeden z grafů UR_G a UL_G obsahuje cyklus, který obsahuje hranu označenou jinak než ε. Pokud mají všechny hrany ve všech cyklech označení ε, není daný D0L-systém pushy.

Příklad 15. *Na obrázku 1.3 vidíme neomezené grafy pro D0L-systém $G = (\{a, b, c\}, \varphi_G, a)$, kde φ_G je definováno pomocí přepisovacích pravidel $a \to abc$, $b \to c$ a $c \to c$. Vzhledem k tomu, že jedna hrana cyklu grafu UR_G má označení cb (tedy jiné než ε), je tento D0L-systém pushy.*



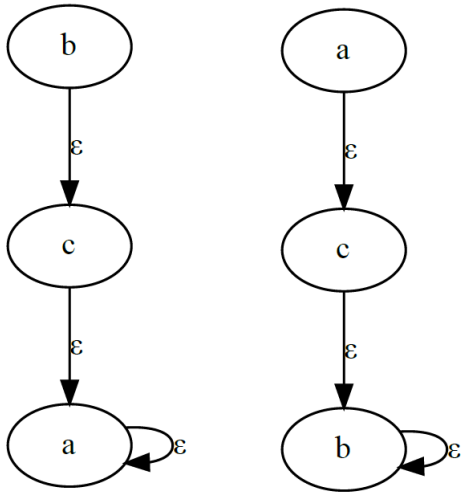
Obrázek 1.3: Ukázka grafů neomezených písmen: UL_G (vlevo) a UR_G (vpravo) pro pushy D0L-systém.

Příklad 16. *Na obrázku 1.4 vidíme neomezené grafy pro D0L-systém $G = (\{a, b, c\}, \varphi_G, a)$, kde φ_G je definováno pomocí přepisovacích pravidel $a \to abc$, $b \to cb$ a $c \to ab$. Vzhledem k tomu, že všechny hrany grafu UR_G mají označení ε, není tento D0L-systém pushy.*

1.3.3 Repetitivita a neomezená repetitivita

Definice 5. *D0L-systém G je repetitivní, pokud pro každé $k \in \mathbb{N}$ existuje neprázdné slovo w takové, že w^k je faktor, tedy $w^k \in S(L(G))$. Pokud navíc existuje slovo w^k pro které platí, že $w^k \in S(L(G))$ pro všechna $k \in \mathbb{N}^+$, říkáme že G je silně repetitivní. Podle [9] je každý repetitivní D0L-systém silně repetitivní.*

Definice 6. *D0L-systém G je neomezeně repetitivní pokud existuje $w \in S(L(G))$ takové, že $w^k \in S(L(G))$ pro všechna k a w obsahuje minimálně jedno neomezené písmeno.*



Obrázek 1.4: Ukázka grafů neomezených písmen: UL_G (vlevo) a UR_G (vpravo) pro non-pushy D0L-systém.

O neomezené repetitivitě je možné rozhodnout pomocí algoritmu na zkonstruování injektivních zjednodušení (více v [10], [11] a [2]) společně s algoritmem pro nalezení všech konečných repetic v G [2]. Navíc podle [3] je každé injektivní G svým vlastním injektivním zjednodušením. Zároveň platí, že pokud je podmínka neomezené repetitivity splněna pro nějaké injektivní zjednodušení, je splněna pro všechna injektivní zjednodušení - viz tvrzení 10 v [3]. Proto v případě, že budeme uvažovat pouze injektivní D0L-systémy, stačí nám k rozhodnutí o neomezené repetitivitě pouze algoritmus, který je popsán v kapitole 2.4.
Důležitá věta pro další závěry je odvozena z Tvrzení 10 v [3] a z Věty 1 v [2].

Věta 7. *Nechť G je repetitivní D0L-systém, potom platí jedno z následujících:*

(i) G je pushy,

(ii) G je neomezeně repetitivní.

1.3.4 Cirkularita a synchronizační zpoždění

V literatuře je možné najít dva mírně rozdílné pohledy na cirkularitu. Oba tyto pohledy lze však vyjádřit pomocí pojmu interpretace.[3]

Definice 8. *Nechť $G = (\mathcal{A}, \varphi, w)$ je PD0L-systém. Potom interpretace slova $u \in S(L(G))$ je trojice (p, v, s), kde v je slovo z $\mathcal{A}^+$, $p, s \in \mathcal{A}^*$ a zároveň platí $\varphi(v) = pus$.*

Příklad 17. *Uvažme PD0L-systém $G = (\{a, b, c\}, \varphi_G, a)$, kde φ_G je definováno pomocí přepisovacích pravidel $a \to abc$, $b \to bc$ a $c \to c$. Potom interpretace faktoru bc jsou: (a, a, ε) a $(\varepsilon, b, \varepsilon)$.*

Definice 9. *O dvou interpretacích (p, v, s) a (p', v', s') říkáme, že jsou synchronizované na pozici k, pokud se na pozici k potkávají hranice mezi obrazy dvou písmen v obou interpretacích (označme $a|b$).*

Definice 10. *Nechť $G = (\mathcal{A}, \varphi, w)$ je PD0L-systém. Slovo $u \in S(L(G))$ má synchronizační bod na pozici k, kde $0 \leq k \leq |u|$, pokud všechny jeho interpretace jsou synchronizované na pozici k.*

Příklad 18. *Uvažme PD0L-systém $D = (\{a, b, c\}, \varphi_D, a)$, kde φ_D je definováno pomocí přepisovacích pravidel: $a \to abc$, $b \to cab$ a $c \to bbc$. Například slovo bccab má tyto dvě interpretace: (a, ab, ε) a (b, cb, ε). Pro obě interpretace platí, že hranice mezi obrazy dvou písmen jsou následující: $bc|cab|$. Hranice tedy existuje pro $k = 2$ a $k = 5$. Tedy slovo bccab má dva synchronizační body $(2, 5)$. Oproti tomu slovo abc má například tyto dvě interpretace: $(\varepsilon, a, \varepsilon)$ a (c, bb, ab). První interpretace má hranice $|abc|$ a druhá interpretace má hranice $ab|c$. Tyto dvě interpretace nemají hranici pro stejné k a proto zde synchronizační bod neexistuje.*

Následující definice cirkularity je převzata z [12].

Definice 11. *Nechť $G = (\mathcal{A}, \varphi, w)$ je PD0L-systém a nechť (p, v, s) a (p', v', s') jsou dvě interpretace neprázdného slova $u \in S(L(G))$ s $v = v_1 \cdots v_n$, $v' = v'_0 \cdots v'_m$ a $u = u_1 \cdots u_l$. Řekneme, že G je cirkulární se synchronizačním zpožděním $D > 0$, pokud kdykoli máme*

$$|\varphi(v_1 \cdots v_i)| - |p| > D$$

a

$$|\varphi(v_{i+1} \cdots v_n)| - |s| > D$$

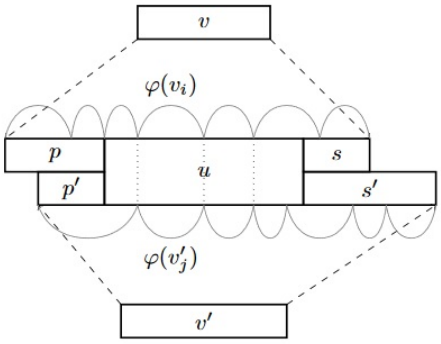
pro nějaké $1 \leq i \leq n$, potom máme $1 \leq j \leq m$ takové, že

$$|\varphi(v_1 \cdots v_{i-1})| - |p| = |\varphi(v'_1 \cdots v'_{j-1})| - |p'|$$

a $v_i = v'_j$.

Tato definice říká, že daný D0L-systém je cirkulární, pokud existuje dostatečně dlouhé slovo, které má jedinečný předobraz, s výjimkou nějakého prefixu a suffixu kratšího než konstanta D. Dále si můžeme všimnout, že pokud G obsahuje libovolně dlouhá slova se dvěma různými předobrazy (tj. pro jakékoli $n > 0$ existují slova v a $u \in S(L(G))$ delší než n s $\varphi(v) = \varphi(u)$), pak nemůže být cirkulární.

Dále můžeme cirkulární D0L-systém, který je injektivní, definovat pomocí synchronizačního bodu následovně vit [4] nebo [15]

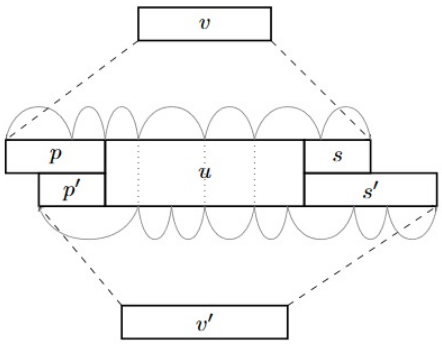


Obrázek 1.5: Dvě interpretace z definice cirkularity s $v_i = v'_j$.[3]

Definice 12. *D0L-systém $G = (\mathcal{A}, \varphi, w)$ je cirkulární na $\mathcal{U}$, pokud φ je injektivní na $\mathcal{U}$ a existuje konstanta $Z \in \mathbb{N}$ taková, že každé slovo $v \in \mathcal{U}$ kde $|v| \geq 2Z$ má alespoň jeden synchronizační bod. Nejmenší Z, pro které je tato podmínka splněna se nazývá synchronizační zpoždění.*

Situace ohledně rozhodnutí, co zvolit za množinu $\mathcal{U}$, je shodná se situací v případě injektivity. Pro naši práci bylo za $\mathcal{U}$ zvoleno $S(L(G))$.

Cirkularita může být samozřejmě definována i pro neinjektivní D0L-systémy, avšak rozhodnutí o cirkularitě se v tomto případě stává velmi komplikovaným problémem a proto v naší práci i algoritmech uvažujeme pouze injektivní D0L-systémy.



Obrázek 1.6: Dvě interpretace synchronizované na pozicích znázorněných tečkovanými čarami.[3]

1.3.4.1 Vztah mezi cirkularitou a neomezenou repetitivitou

Nyní uvedeme definici slabé cirkularity, kterou potřebujeme pro formulaci věty o charakterizaci injektivního cirkulárního D0L-systému pomocí neomezené repetitivity. Nicméně pro injektivní D0L-systémy je slabá cirkularita to samé jako cirkularita, neplatí to však pro neinjektivní případy.

Definice 13. *PD0L-system G se nazývá slabě cirkulární, pokud existuje konstanta $D > 0$, taková že každé v z $S(L(G))$ je delší než $2D$ má synchronizační bod.*

Jak je uvedeno výše, pokud je G injektivní, slabá cirkularita je ekvivalentní cirkularitě. Jak ukazuje následující příklad, toto neplatí pro neinjektivní případ.

Příklad 19. *Uvažme D0L-systém $G_1 = (\{a, b, c\}, \varphi_1, a)$ s neinjektivním $\varphi_1 : a \to abca, b \to bc, c \to bc$. Tento systém není cirkulární, protože pro všechna $m \in \mathbb{N}$ slovo $(bc)^{2m}$ má dva rozdílné předobrazy $(bc)^m$ a $(cb)^m$. Avšak odpovídající interpretace mají synchronizační body pro $m > 1$ na pozicích $2k$ pro všech $0 < k < m$. Kromě toho lze snadno zkontrolovat, že G_1 je slabě cirkulární. Takže cirkularita implikuje slabou cirkularitu, ale obráceně to není pravda.*

Věta 14. *Každý PD0L-systém, který není slabě cirkulární, je repetitivní.*

K důkazu této věty jsou v [3] použita dvě lemata, která zde uvedeme. Následující lemma a jeho důkaz je založen na myšlenkách v důkazu věty 4.35 v [13]. Důkaz věty 13 i obou následujících lemmat je možné najít v [3].

Lemma 15. *Nechť $G = (\mathcal{A}, \varphi, w)$ je PD0L systém. Pokud existuje sekvence $\epsilon(k)$ s $\lim_{k\to\infty} \varepsilon(k) = +\infty$ a pokud pro každé $k \in \mathbb{N}$ existují dvě neprázdná slova u a $v \in S(L(G))$ obsahující neomezené písmeno, tak, že jsou splněny následující podmínky*

(i) $|u| = k$;

(ii) existují dvě celá čísla m a n taková, že $m > n$ a písmena a a b taková, že pro každé $i \in \{m, n\}$ slovo $\varphi^i(u)$ je faktor $\varphi^i(v)$ a $\varphi^i(v)$ je faktor $\varphi^i(aub)$, navíc $\frac{|\varphi^i(u)|}{|\varphi^i(a)|} > \epsilon(k)$ nebo $\frac{|\varphi^i(u)|}{|\varphi^i(b)|} > \epsilon(k)$; a

(iii) pro každé $i \in \{m, n\}$ faktor $\varphi^i(u)$ nemá synchronizační bod: dvě nesynchronizované interpretace jsou $(\varepsilon, \varphi^{i-1}(u), \varepsilon)$ a $(p_i, \varphi^{i-1}(v), s_i)$,

potom je D0L-systém repetitivní.

Lemma 16. *V každém PD0L-systému existuje konstanta C taková, že všechny faktory nad omezenými písmeny delší než C mají synchronizační bod.*

V předchozí části jsme prokázali, že jakýkoli PD0L-systém, který není slabě cirkulární, je repetitivní. Další věta dává charakterizaci injektivního cirkulárního D0L-systému.

Věta 17. *Injektivní D0L-systém $G = (\mathcal{A}, \varphi, w)$ není cirkulární právě tehdy, když je neomezeně repetitivní.*

Důkaz. ($\Rightarrow$) Protože injektivní morfismus je zároveň nevymazávající (PD0L), tak z věty 14 víme, že pokud není slabě cirkulární, pak je repetitivní. Dále podle věty 7 platí, že je-li repetitivní, tak je buď pushy nebo neomezeně repetitivní. Předpokládejme tedy, že je pushy a není neomezeně repetitivní. Pak by platilo, že všechny repetice u^l, kde $u \in S(L(G))$ jsou pouze nad omezenými písmeny, tedy že $u \in \mathcal{A}_0^+$. Podle Lemmatu 15 a 16 však platí, že pokud máme dostatečně dlouhé nesynchronizované faktory, které obsahájí stále delší repetice, tak tyto repetice nemohou být nad omezenými písmeny. Proto tedy morfismus musí být neomezeně repetitivní a tím vzniká spor.

($\Leftarrow$): Tvrzení 10 v [3] říká, že existuje kladné celé číslo l a písmeno a, takové že $(\varphi^l)^\infty(a) = w^\omega$ pro nějaké $\mathcal{A}^+$. V [2] je dokázáno, že slovo w může být vybráno tak, aby obsahovalo písmeno a pouze na začátku. Z toho vyplývá, že $\varphi^l(w) = w^k$ pro nějaké $k > 1$. Protože φ je injektivní, musíme mít $\varphi(p) \neq w$ pro všechny prefixy p z w. To znamená, že pro všechna $n \in \mathbb{N}$ slovo w^{nk} má dvě nesynchronizované interpretace $(\epsilon, w^n, \epsilon)$ a $(w, w^{(n+1)}, w^{(k-1)})$ a tedy není cirkulární.

Tvrzení 18. *V předchozí větě nemůžeme vynechat předpoklad injektivity a nahradit cirkularitu slabou cirkularitou: vezměme v úvahu D0L-systém G_1 z příkladu 19. Podmínky tvrzení 10 v [3] jsou splněny pro $l = 1$ a písmeno b s $w = bc$, ale odpovídající D0L-systém je slabě cirkulární.*

Jelikož existenci l a a splňující podmínku tvrzení 10 z [3] lze prokázat jednoduchým a rychlým algoritmem popsaným v [14], je možné navrhnout zjednodušený algoritmus rozhodující o cirkularitě. Tedy, že pokud je zadaný D0L-systém injektivní, není pushy a zároveň není neomezeně repetitivní, pak je cirkulární. Tento zjednodušený algoritmus pro zjištění cirkularity využíváme i v implementaci.

Kapitola 2

Algoritmy a popis implementace

V této kapitole jsou popsány implementované algoritmy pro jednotlivé vlastnosti D0L-systému popsané v kapitole 1.2. Pro každou vlastnost algoritmus nejprve slovně popíšeme a ukážeme na příkladu. Dále pro každou vlastnost uvedeme pseudokód algoritmu implementovaného v jazyce Python. Tyto algoritmy budeme dále používat pro experimenty a následné vyhodnocení výsledků.

Jak již bylo v práci dříve použito, pro jednoduší formulaci budeme morfismus pro jednotlivá písmena nazývat jako přepisovací pravidla a aplikaci morfismu budeme nazývat jako použití pravidla pro dané písmeno.

2.1 Nalezení faktorů dané délky

Jedná se o algoritmus, který pro zadaný D0L-systému nalezne faktory požadované délky n včetně informací o tom, z jakých předobrazů jednotlivé faktory vznikly. Informaci o předobrazu ukládáme ve tvaru $u : (v, t)$, kde $a \in S(L(G))$ je nově nalezený faktor, $b \in \mathcal{A}^+$ představuje faktor, v jehož obrazu po aplikování morfismu byl hledaný faktor nalezen, a $t \in \mathbb{N}_0$ reprezentuje pozici, na které nově nalezený faktor v obrazu předchozího faktoru začíná. Například tedy v pravidle $a \to ba$, najdeme faktor ba a uložíme pro něj $ba : (a, 0)$.

2.1.1 Popis algoritmu na nalezení faktorů dané délky s ukázkou běhu algoritmu

Algoritmus slovně popíšeme a každý jeho krok rovnou ukážeme na vybraném D0L-systému. Nechť $G = (\mathcal{A}, \varphi, w)$ je D0L-systém, kde $\mathcal{A} = (a, b, c)$, $\varphi(a) = abc, \varphi(b) = bc, \varphi(c) = a$, $w = a$ a $n = 4$.

1) Označíme si písmena, jejichž pravidla (uložím písmena na levé straně pravidla) mají na pravé straně pouze jeden znak a nazveme je pravidlem délky 1,

   - Vyberu pravidlo délky 1: c,

2) najdeme faktory délky 2 přímo v pravých stranách pravidel,

   - Najdu faktory délky 2 přímo v pravidlech: ab: (a,0), bc: (a,1), (b,0)

3) použitím pravidel na faktory z bodu 2) zkontrolujeme, zda v jejich obrazech nenajdeme nový faktor délky 2, či zda nenajdu nový předobraz již nalezených faktorů,

   -
   ```
   - ab -> (ab)cb(c) => cb: (ab,2)
   - cb -> ab(c) => ab: (cb,0)
   - bc -> (b)ca => ca: (bc,1)
   - ca -> aa(bc) => aa: (ca,0)
   - aa -> (ab)ca(bc) => ca: (aa,2)
   ```

4) všechny poslední nalezené faktory (nyní délky 2) uložíme jako aktuální faktory (ty které v daném kroku zpracovávám),

   -
   ```
   - ab: (a,0), (cb,0)
   - bc: (a,1), (b,0)
   - cb: (ab,2)
   - ca: (bc,1), (aa,2)
   - aa: (ca,2)
   ```

5) použijeme pravidla na aktuální faktory a uložíme vzniklé obrazy (slova),

   -
   ```
   - ab -> abcbc
   - bc -> bca
   - cb -> abc
   - ca -> aabc
   - aa -> abcabc
   ```

6) v uložených slovech najdeme faktory délky o 1 větší než jsou ty aktuální,

   -
   ```
   - abc: (ab,0), (cb,0), (ca,1), (aa,0), (aa,3)
   - bcb: (ab,1)
   - cbc: (ab,2)
   - bca: (bc,0), (aa,1)
   - aab: (ca,0)
   - cab: (aa,2)
   ```

7) z nově nalezených faktorů, vybereme ty které mají na všech pozicích kromě první a poslední pouze pravidla délky 1 (ad1),

   - bcb, bca

8) na tyto faktory aplikujeme pravidla a zkontrolujeme, zda se neobjeví nový faktor či nový předobraz k nějakému již nalezenému faktoru,

- ```
  - bcb -> (b)cab(c) => cab: (bcb,1)
  - bca -> (b)caa(bc) => caa: (bca,1)
  ```

9) kroky 5-8 opakujeme dokud poslední nalezené faktor nemají délku rovnou n.

5. Na aktuální faktory použijeme pravidla:

   - ```
     - abc => abcbca
     - bcb => bcabc
     - cbc => abca
     - bca => bcaabc
     - aab => abcabcbc
     - cab => aabcbc
     - caa => aabcabc
     ```

6) v uložených slovech najdeme faktory délky o 1 větší než jsou ty aktuální,

   - ```
     - abcb: (abc,0), (aab,3), (cab,1)
     - bcbc: (abc,1), (aab,4), (cab,2)
     - cbca: (abc,2)
     - bcab: (bcb,0), (aab,1), (caa,2)
     - cabc: (bcb,1), (aab,2), (caa,3)
     - abca: (cbc,0), (aab,0), (caa,1)
     - bcaa: (bca,0)
     - caab: (bca,1)
     - aabc: (bca,2), (cab,0), (caa,0)
     ```

7) Vybereme faktory, které mají uprostřed pouze pravidla délky 1:
   - žádné nejsou

8) Použijeme pravidla na vybrané faktory a zjistíme zda nenajdeme nové: Vzhledem k tomu, že v kroku 7 se nenašly žádné faktory, není na co pravidla použít.

9) Vzhledem k tomu, že poslední nalezené faktory mají délku 4, nalezli jsme faktory požadované délky $n = 4$. Proto zde algoritmus končí.

2.1.2 Pseudokód algoritmu na nalezení faktorů dané délky

```
input: morphism, start letter, lenght of factor = n
output: factors len <= n with detail of their anccestors

one_letters = rules whose right side is only one letter


bigrams = find all factors of lenght 2 in rules
all_bigrams = use rules to bigrams and check if there is not
```
```

```
    yet found factor of lenght 2 or new ancestor for any bigram

actual_factors = all_bigrams

for i in range (3 to n + 1)
    for factor in actual_factors:
        for letter in factor:
            use rules to letters and save the words

    in all words:
        find new_factors of lenght i with their origin ancestor

    in new_factors:
        choose short_factors -> those which have on position
            {1 to i-1} only letters in list one_letters

    for factor in short_factors:
        use rules to factor -> save if you find new factor
            or new ancestor to any factor

    actual_factors = union new_factors and short_factors
    all_factors.aad(actual_factors)

return all_factors
```

2.2 Injektivita

Zde popíšeme a na příkladu ukážeme běh algoritmu, který rozhodne, zda je zadaný D0L-systém neinjektivní či (pravděpodobně) injektivní. V první příkladě je vše demonstrováno na neinjektivním D0L-systému, v druhém příkladě na D0L-systému pravděpodobně injektivním.

2.2.1 Popis algoritmu na rozhodnutí o injektivitě s ukázkou běhu algoritmu

2.2.1.1 Neinjektivní příklad

Nechť $G = (\mathcal{A}, \varphi, w)$ je D0L-systém, kde $\mathcal{A} = (a, b, c, d, e)$, $\varphi(a) = abcd, \varphi(b) = c, \varphi(c) = c, \varphi(d) = e, (e) = b$ a $w = a$.

1) V prvním kroku spustíme algoritmus pro nalezení faktorů dané délky,

- ```
  ab: (a, 0)
  bc: (a, 1), (ec, 0)
  cd: (a, 2)
  ```
  ```

```
dc: (ab, 3)
cc: (cc, 0), (cb, 0), (bc, 0)
ce: (cd, 0)
ec: (dc, 0)
cb: (ce, 0)
abc: (a, 0)
bcd: (a, 1)
cdc: (ab, 2)
dcc: (abc, 3)
cce: (bcd, 0)
cec: (cdc, 0)
ecc: (dcc, 0)
ccb: (cce, 0)
cbc: (cec, 0)
bcc: (ecc, 0)
ccc: (cbc, 0), (ccc, 0), (bcc, 0), (ccb, 0)
```

2) z nalezených faktorů a jejich předobrazů vybereme ty, které má smysl zkoumat, tzn. ty, které začínají na počáteční pozici (pozice 0) svého předobrazu alespoň ve dvou případech,

   - ```
     cc: (cc, 0), (cb, 0), (bc, 0)
     ccc: (cbc, 0), (ccc, 0), (bcc, 0), (ccb, 0)
     ```

3) nyní zkoumáme, zda má nějaký z vybraných faktorů i přesný konec jako jeho předobraz,

   - cc:
     1) $|c|c|$ - OK
     2) $|c|b|$ - OK
     3) $|b|c|$ - OK

4) pokud najdeme faktor, který má takových přesných předobrazů 2 a více, je D0L-systém neinjektivní, pokud takový faktor nenajdeme, je pravděpodobně injektivní. (Pravděpodobně proto, že takový faktor může existovat mezi delšími faktory, které nebyly prohledávány).

   - nalezli jsme faktor, který má 2 a více přesných předků, a proto tento D0L-systém je neinjektivní a není třeba zkoumat další faktory.

2.2.1.2 Injektivní příklad

Nechť $G = \mathcal{A}, \varphi, w)$ je D0L-systém, kde $\mathcal{A} = (a, b, c)$, $\varphi(a) = abc, \varphi(b) = c, \varphi(c) = b$ a $w = a$.

1) V prvním kroku spustíme algoritmus pro nalezení faktorů dané délky,
     ```

- ```
  ab: (a,0)
  bc: (a,1), (cb,0)
  cc: (ab,2), (bb,0)
  bb: (cc,0)
  cb: (bc,0)
  abc: (ab,0)
  bcc: (ab,1), (cbb,0)
  ccb: (abc,2), (bbc,0)
  bbc: (ccb,0)
  cbb: (bcc,0)
  ```

2) z nalezených faktorů a jejich předobrazů vybereme ty, které má smysl zkoumat, tzn. ty, které začínají na počáteční pozici (pozice 0) svého předobrazu alespoň ve dvou případech,

   - žádné takové jsme nenalezli

3) nyní zkoumáme, zda má nějaký z vybraných faktorů i přesný konec jako jeho předobraz,

   - vzhledem k tomu, že v bodu 2 nebyly žádné faktory nalezeny, není zde co zkoumat

4) pokud najdeme faktor, který má takových přesných předobrazů 2 a více, je D0L-systém neinjektivní, pokud takový faktor nenajdeme, je pravděpodobně injektivní. (Pravděpodobně proto, že takový faktor může existovat mezi delšími faktory, které nebyly prohledávány).

   - nenalezli jsme faktor, který má 2 a více přesných předků a proto tento D0L-systém je pravděpodobně injektivní.

2.2.2 Pseudokód algoritmu na injektivitu

```
input: morphism, output of find_factor, max factor lenght
to injektiv decide (m)

output: injektiv/non-injektiv

factors = output of use algorithm for find_factor

for factor in factors:
    go throw all factor ancestors:
        if start position in ancestor = 0:
            count +=1
    if count >= 2 (number of ancestors with start on position 0):
        good_start = factor (save the factor to final list)
```
```

```
if good_start is empty:
    output is INJEKTIV

for factor in good_start:
  for ancestor in factor_ancestors:
      if start position in ancestor = 0:
          word = rule use to letters in ancestor
          if word = item:
              image += 1 (exact ancestor = generate only this factor)
    if image >= 2: (found factor with 2 or more exact ancestors)
        output is NON-INJEKTIV

if any factor doesnt have 2 or more exact ancestor:
    output is INJEKTIV
```

2.3 Pushy

2.3.1 Popis algoritmu pushy s ukázkou běhu algoritmu

Zde popíšeme algoritmus, který pro zadaný D0L-systém rozhodne, zda je či není pushy. Běh algoritmu je demonstrován na dvou případech, kdy v prvním příkladě je D0L-systém non-pushy a v druhém příkladě je D0L-systém pushy.

2.3.1.1 Non-pushy příklad

Nechť $G = (\mathcal{A}, \varphi, w)$ je D0L-systém, kde $\mathcal{A} = (a, b)$, $\varphi(a) = ab, \varphi(b) = ba$ a $w = a$.

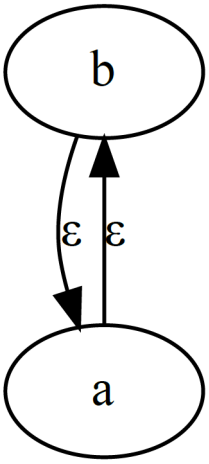
1) Vybereme a uložíme neomezená písmena,

   - vybereme neomezená písmena: a,b

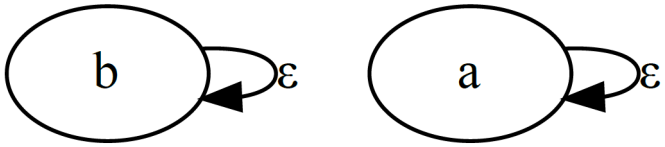
2) pro vybraná neomezená písmena vytvoříme grafy neomezených písmen zleva i zprava, viz obrázky 2.1 a 2.2.

3) prozkoumáme hrany ve vytvořených grafech. Pokud je nějaké hrana označena jinak než ε, morfismus je pushy. Pokud jsou všechny hrany označeny pouze ε, morfismus je non-pushy.

   - hrany grafu zprava jsou: ε, ε hrany grafu zleva jsou: ε, $\varepsilon \rightarrow$ všechny hrany na grafech jsou ε, proto je zadaný D0L-systém non-pushy.



Obrázek 2.1: Graf neomezených písmen zprava.



Obrázek 2.2: Graf neomezených písmen zleva.

2.3.1.2 Pushy příklad

Nechť $G = (\mathcal{A}, \varphi, w)$ je D0L-systém, kde $\mathcal{A} = (a, b, c, d)$, $\varphi(a) = abcd, \varphi(b) = c, \varphi(c) = b, \varphi(d) = bcbc$ a $w = a$.

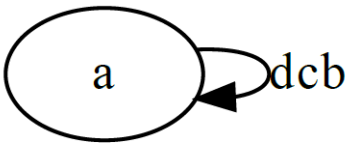
1) Vybereme a uložíme neomezená písmena,

   - vybereme neomezená písmena: a

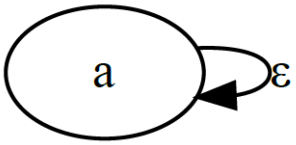
2) pro vybraná neomezená písmena vytvoříme grafy neomezených písmen zleva i zprava, viz obrázky 2.3 a 2.4.

3) prozkoumáme hrany ve vytvořených grafech. Pokud je nějaké hrana označena jinak než ε, morfismus je pushy. Pokud jsou všechny hrany označeny pouze ε, morfismus je non-pushy.

   - hrany grafu zprava jsou: *dcb* hrany grafu zleva jsou: $\varepsilon \rightarrow$ na hranách se objevuje i něco jiného než ε, proto je zadaný D0L-systém pushy.



Obrázek 2.3: Graf neomezených písmen zprava.



Obrázek 2.4: Graf neomezených písmen zleva.

2.3.2 Pseudokód algoritmu pushy

```
input: morphism
output: puhsy/non-pushy

get the list of unbounded letters

left_graph = make unbounded graph from left
right_graph = make unbounded graph from right

for edge in left_graph, right_graph
    control name of edges
    if some edge != epsilon
        output = PUSHY
    else
        output = NON-PUSHY
```

2.4 Repetitivita

Tento algoritmus ověří, zda je zadaný D0L-systém repetitivní. Pro rozhodnutí o repetitivitě využíváme algoritmus uvedený v [2]. Algoritmus je, stejně jako algoritmy v předchozí části práce, nejprve popsán a následně demonstrován na příkladu.

2.4.1 Popis algoritmu na repetitivitu s ukázkou běhu algoritmu

Nechť $G = (\mathcal{A}, \varphi, w)$ je D0L-systém, kde $\mathcal{A} = (a, b, c)$, $\varphi(a) = ab, \varphi(b) = a, \varphi(c) = bca$ a $w = a$.

1) Vybereme pouze neomezená písmena.
   - Vybereme neomezená písmena: a, b, c
2) Pomocí grafu prvních písmen najdeme periodické body.
   - Vybereme periodické body: a, b a uložím jejich l (počet potřebných zobrazení, než se písmeno zobrazí znovu samo na sebe): $l_a = 2$, $l_b = 2$
3) Vybereme písmena, která jsou neomezená a zároveň jsou periodickým bodem a pro tyto písmena provádíme následující kroky.
   - Vyberu neomezená písmena, která jsou zároveň periodickým bodem: a, b
4) Najdeme nejmenší $k \leq (velikost)A$ takové, že $(\varphi^l)^k(a)$ obsahuje nejméně 2 výskyty jednoho neomezeného slova.
   - a) $\varphi^2(0) = 0- > 12- > 0120 => k = 1$ (obsahuje nejméně 2 výskyty neomezeného písmene
     b) $\varphi^2(1) = 1- > 0- > 12 => k = 1$ (obsahuje nejméně 2 výskyty neomezeného písmene
5) Pokud $(\varphi^l)^k(a)$ neobsahuje 2 výskyty a, tak potom $(\varphi^l)^k(a)$ není periodické. Jinak označíme v nejdelší prefix $(\varphi^l)^k(a)$ obsahující a pouze jako první písmeno.
   - a) $(\varphi^2)^1(0) = 0120 =>$ obsahuje 2 výskyty $0 => v = 012$
     b) $(\varphi^2)^1(1) = 12 =>$ neobsahuje 2 výskyty $1 =>$ 1 není periodické
6) $(\varphi^l)^k(a)$ je periodické pouze tehdy, když $\varphi^l(v) = v^m$ pro nějaké $m \geq 2$.
   - ověřím, zda $\varphi^2(012) = v^m$: $\varphi^2(012) = 120120 \rightarrow 012012012012 \Rightarrow \varphi^2(012) = v^4 \Rightarrow m = 4 \Rightarrow 0$ je periodické

7) Aby byl celý DOL systém repetitivní, musí být periodická všechna písmena, jinak není D0L-systém repetitivní.

   - Protože 1 není periodické, není zadaný D0L systém repetitivní.

2.4.2 Pseudokód algoritmu na repetitivitu

```
input morphism
output repetitiv/non repetitiv

is_periodic(morphism, unbounded_letters)
    first_letter <- find first letter for each rule
    l <- find l for all letter = how many steps are needed
        to go from letter again in the same letter
    good_letter <- letter which is unbounded & l exist

    for letter in good_letter
        find k and word -> k is index how many repetitions
            is needed to have two times of unbounded letter
            in the generate word
        if k exist:
            in saved word find prefix where the letter is only
                on the first position
        else:
            letter <- 'not-periodic'

        if prefix exist
            m <- m is index when it applies only if use rules
                l-times on prefix = prefix m-times

        if m exist
            letter <- 'periodic'
        else:
            letter <- 'not-periodic'
    if all letters are periodic:
        output is REPETITIV
    else:
        output is NON-REPETITIV
```

2.5 Cirkularita

Jak bylo definováno na konci kapitoly o cirkularitě, pro jednoduchost můžeme k rozhodnutí o cirkularitě využít další již dříve uvedené vlastnoti. Konkrétně

injektivitu, pushy a repetitivitu. Algoritmus na cirkularitu je tedy velmi jednoduchý.

2.5.1 Popis algoritmu na cirkularitu s ukázkou běhu algoritmu

Nechť $G = (\mathcal{A}, \varphi, w)$ je D0L-systém, kde $\mathcal{A} = (a, b)$, $\varphi(a) = ab, \varphi(b) = ba$ a $w = a$.

1) Rozhodneme, zda je zadaný DOL-systém injektivní.
   - Zadaný D0L-systém je injektivní.
2) Rozhodneme, zda je zadaný D0L-systém pushy.
   - Zadaný D0L-systém je non-pushy.
3) Rozhodneme, zda je zadaný D0L-systém repetitivní.
   - Zadaný D0L-systém je nerepetitivní.
4) Podle předchozích výsledků rozhodneme o cirkularitě. Pokud je D0L-systém injektivní, non-pushy a neperiodický, pak je cirkulární. Jinak není cirkulární.
   - Zadaný D0L-systém je tedy injektivní, non-pushy a nerepetitivní a je tedy cirkulární.

2.5.2 Pseudokód algoritmu na cirkularitu

```
input result of: pushy, injektiv, repetitiv
output cirkular/non-cirkular

if pushy == non-pushy
    if injektiv == injektiv
        if periodic == non-repetitiv
            result <- CIRKULAR
else
    result <- NON-CIRKULAR
```

2.6 Synchronizační zpoždění

Pokud je zadaný D0L-systém cirkulární, pak tento algoritmus hledá u jednotlivých faktorů synchronizační body a pomocí nich pak najde synchronizační zpoždění.

2.6.1 Popis algoritmu na nalezení synchronizačního zpoždění a ukázka běhu algoritmu

Nechť $G = (\mathcal{A}, \varphi, w)$ je D0L-systém, kde $\mathcal{A} = (a, b)$, $\varphi(a) = ab, \varphi(b) = ba$ a $w = a$.

1) V prvním kroku algoritmu spustíme algoritmus pro nalezení faktorů dané délky,

   - ```
     ba: (aa,1), (b,0)
     ab: (a,0), (bb,1)
     aa: (ba,1)
     bb: (ab,1)
     baa: (ba,0)
     aab: (ba,1)
     abb: (ab,0)
     bba: (ab,1)
     aba: (aa,0), (bb,1)
     bab: (aa,1), (bb,0)
     baab: (ba,0)
     aaba: (baa,1)
     abab: (aa,0)
     babb: (aab,1)
     abba: (ab,0)
     bbab: (abb,1)
     baba: (bb,0)
     abaa: (bba,1)
     bbaa: (aba,1)
     aabb: (bab,1)
     ```

2) Postupně procházíme všechny faktory délky 2 až n a v jednom kroku vždy prohledáváme faktory jedné délky,

   - nejprve procházíme faktory délky 2: ba, ab, aa, bb

3) Pro každý faktor rozhodneme, zda je či není synchronizovaný. Pokud má faktor pouze jeden předobraz (interpretaci), můžeme o něm rovnou říct, že je synchronizovaný. Pokud má však předobrazů (interpretací) více, je potřeba prozkoumat, zda jsou předobrazy (interpretace) synchronizované na nějaké pozici k či nikoli. Viz definice, kdy jsou dvě interpretace synchronizované.

   - U každého rozhodneme, zda je či není synchronizovaný:
     - aa: synchronizovaný (pouze jeden předek)
     - bb: synchronizovaný (pouze jeden předek)
     ```

- *ba*: z předka aa máme hranice $b|a$, z předka b máme hranice $|ba| \rightarrow$ hranice u předků nejsou stejné, tedy tento faktor není synchronizovaný
- *ab*: z předka a máme hranice $|ab|$, z předka bb máme hranice $a|b \rightarrow$ hranice u předků nejsou stejné, tedy tento faktor není synchronizovaný

4) Pokud jsou všechny faktory dané délky synchronizované tak synchronizační zpoždění má hodnotu délky těchto synchronizovaných faktorů. Pokud všechny faktory synchronizované nejsou, je třeba jít kontrolovat faktory o 1 delší a opakovat znovu krok 2 a 3.

- všechny faktory nejsou synchronizované, proto musíme pokračovat s faktory délky o 1 větší

  2) nyní procházíme faktory délky 3: $baa, aab, abb, bba, aba, bab$
  3) U každého rozhodneme, zda je či není synchronizovaný:
     - *baa*: synchronizovaný (pouze jeden předek)
     - *aab*: synchronizovaný (pouze jeden předek)
     - *abb*: synchronizovaný (pouze jeden předek)
     - *bba*: synchronizovaný (pouze jeden předek)
     - *aba*: z předka aa máme hranice $ab|a|$, z předka bb máme hranice $a|ba \rightarrow$ hranice u předků nejsou stejné, tedy tento faktor není synchronizovaný
     - *bab*: z předka aa máme hranice $b|ab|$, z předka bb máme hranice $|ba|b \rightarrow$ hranice u předků nejsou stejné, tedy tento faktor není synchronizovaný
  4) všechny faktory nejsou synchronizované, proto musíme pokračovat s faktory délky o 1 větší
  2) nyní procházím faktory délky 4: *baab*, *aaba*, *abab*, *babb*, *abba*, *bbab*, *baba*, *abaa*, *bbaa*, *aabb*
  3) U každého rozhodnu, zda je či není synchronizovaný:
     - *baab*: synchronizovaný (pouze jeden předek)
     - *aaba*: synchronizovaný (pouze jeden předek)
     - *abab*: synchronizovaný (pouze jeden předek)
     - *babb*: synchronizovaný (pouze jeden předek)
     - *abba*: synchronizovaný (pouze jeden předek)
     - *bbab*: synchronizovaný (pouze jeden předek)
     - *baba*: synchronizovaný (pouze jeden předek)
     - *abaa*: synchronizovaný (pouze jeden předek)
     - *bbaa*: synchronizovaný (pouze jeden předek)
     - *aabb*: synchronizovaný (pouze jeden předek)

4) všechny faktory délky 4 jsou synchronizované a proto velikost synchronizačního zpoždění je 4.

2.6.2 Pseudokód algoritmu na nalezení synchronizačního zpoždění

```
input: morphism, all_factors, n
output: synchronization delay or info that it wasnt found

for s in range(2, n+1):
    for factor in all_factors:
        if factor have only one ancestor:
            factor is synchro
        else:
            put factor in list to find borders

    for factor in list to find borders:
        find borders in factor from all ancestor

    for factor in list with borders:
        if all borders of factor are same:
            factor is synchro
        else:
            factor is no-synchro

    if all factor are sychro:
        output = Synchronization delay is s.

    else:
        s += 1

if no synchronization point wasnt found:
    output = Synchronization delay was not found.
```

Kapitola 3

Experiment

Experiment jsme prováděli na algoritmech implementovaných v jazyce Python testovaných na zadaných morfismech. Výsledky byly vyhodnoceny tak, aby vytvořili ucelený přehled vlastností pro vybraná data. Všechny funkce v implementovaných algoritmech mají komentáře odpovídající konvencím SageMath viz [16], aby bylo v budoucnu možné tyto algoritmy do SageMath zařadit.

3.1 Zadání experimentu

Cílem experimentu bylo systematicky projít všechny binární morfismy a u každého z nich rozhodnout o implementovaných vlastnostech:

- Je injektivní? Pokud ne, uložit faktory do dané délky, které mají 2 a více předobrazů.
- Je pushy? Pokud ano, uložit příslušné grafy.
- Je neomezeně repetitivní?
- Je cirkulární? Pokud ano, najít synchronizační zpoždění a vypsat nejdelší nesynchronizované faktory.

3.2 Generování dat

Binární morfismy jsou morfismy nad abecedou $\{a, b\}$. Vzhledem k tomu, že záměnou a a b získáme jazyk se stejnými vlastnostmi, lze předpokládat, že obraz písmene a je delší nebo stejně dlouhý jako obraz písmene b a pokryjeme tak stejně všechny morfismy. Generování požadovaných morfismů ukazuje následující pseudokód:

```
input: alphabeth={a,b}
for n  in (2, 3. 4, ..., x)
```

```
for m in (1, 2, 3, ..., n)
    for prod in product( alphabet, repeate = n+m )
        first_rule = prod[:n]
        second_rule = prod[:-m]
        rules.add(a -> first_rule, b -> second_rule)
        in file{(n, m).in} write rules
```

Data vygenerujeme pro každé n od 2 do x a pro každé m od 1 do n pomocí operace *product*: operace kartézský součin množiny $\{a, b\}$ samo se sebou, parametr repeat (viz pseudokód) určuje, kolikrát má být operace provedena. Výslednými daty jsou všechna slova nad $\{a, b\}$ délky $n + m$ a jejich počet je 2^{n+m}. Tyto vygenerované řetězce rozdělím tak, že prvních n znaků bude tvořit obraz a a posledních m znaků bude tvořit obraz b. Například pro $n = 3$ a $m = 1$ jde tedy o řetězce: *aaaa*, *aaab*, *aaba*, *aabb*, *abaa*, *abab*, *abba*, *abbb*, *baaa*, *baab*, *baba*, *babb*, *bbaa*, *bbab*, *bbba*, *bbbb* a morfismy vypadají následovně:

```
a->aaa,b->a
a->aaa,b->b
a->aab,b->a
a->aab,b->b
a->aba,b->a
a->aba,b->b
a->abb,b->a
a->abb,b->b
a->baa,b->a
a->baa,b->b
a->bab,b->a
a->bab,b->b
a->bba,b->a
a->bba,b->b
a->bbb,b->a
a->bbb,b->b
```

Morfismy jsou ukládány do souborů, jejichž pojmenování udává velikost n a m. Předchozí uvedené morfismy jsou tedy uloženy v souboru (3, 1).in.

Pro náš experiment bylo n nastaveno v rozsahu $(2, 6)$ a m bylo nastaveno v rozsahu $(1, 6)$. V případě zájmu o vygenerování morfismů pro větší n stačí v programu změnit počáteční a koncovou hodnotu n (tedy rozsah provádění prvního for cyklu ve výše uvedeném pseudokódu). Pro všechny testované morfismy byl jako axiom (počáteční stav) nastaven znak a.

3.3 Generování výsledků

Pro všechna vygenerovaná data jsme spustili implementované algoritmy a ukládali jsme jejich výsledky. Nejprve je vždy spuštěn algoritmus na vygenerování

všech **faktorů dané délky**. Délku hledaných faktorů jsme pro začátek nastavili na hodnotu 100.

Z vlastností jako první zkoumáme, zda je zadaný morfismus **pushy**. Pokud ano, uložíme příslušné grafy. V opačném případě uložíme pouze výsledek.

Další zkoumanou vlastností je **injektivita** a to na základě výstupu algoritmu na nalezení faktorů dané délky. Pokud zjistíme, že je zadaný morfismus neinjektivní, uložíme si faktory, které do aktuální procházené délky měly 2 a více předobrazů a další vlastnosti již nezkoumáme. Důvodem je, že všechny ostatní vlastnosti jsou v práci definované pro injektivní morfismy a pro neinjektivní případy se jejich zjišťování stává netriviálním případem. Pro algoritmus injektivity je nastavena délka faktorů 100, pokud faktor zapříčiňující neinjektivitu neobjevíme mezi těmito faktory, předpokládáme, že je morfismus injektivní a pokračujeme ve zkoumání dalších vlastností.

Je-li tedy vstupní morfismu injektivní, jako další vlastnost zkoumáme **repetitivitu**. Zde algoritmus pouze rozhodne zda je vstupní morfismus repetitivní či nikoli.

Jako poslední nám zbývá prozkoumat **cirkularitu**. Zda je zadaný morfismus cirkulární vyhodnocujeme na základně předchozích výsledků, viz algoritmus zmíněný na konci kapitoly 1.3.4. Vstupní morfismus tedy musí být nonpushy, injektivní a nerepetitivní. Poté víme, že je i cirkulární. Pokud některou z uvedených vlastností nesplňuje, řekneme že není cirkulární a u necirkulárního morfismu nehledáme synchronizační zpoždění a program ukončíme. Algoritmus pro nalezení synchronizačního zpoždění tedy spouštíme pouze pro cirkulární morfismy. Tento algoritmus najde velikost synchronizačního zpoždění a vypíše nejdelší nesynchronizované faktory. I tento algoritmus využívá pro vstup faktory nalezené algoritmem na hledání faktorů dané délky, tedy nejprve má k dispozici faktory maximálně délky 100. V případě, že by nebylo synchronizační zpoždění do délky 100 nalezeno, program si sám vygeneruje faktory větší délky.

Postup generování výsledků zde popsaný znázorňuje následující pseudokód:

```
for i in (2,6):
    for y in (2, i+1):
        do_file(i, y)

do_file(i, y):
    for rule_set in file:
        get_all_factors()
        out = {
                rule_set = rul_set
                pushy: is_pushy()
                injektiv: is_injektiv()
                }
        if not out[injektiv]:
```

```
            out[rules] = get_non_injektiv_rules()
        if out[injektiv]:
            out[repetitiv] = is_repetitiv()
            out[cirkular] = is_cirkular()
            if out[cirkular]:
                out[synchro] = is_synchro()
                out[factors] = get_unsynchro_factors()
        if out[pushy]:
            out[graf_l] = get_graphs(L)
            out[graf_r] = get_graphs(R)
        output.append(out)
    in file{(i, y).out} write output
```

K ukládání výsledků byl použit balíček JSON viz [17]. Pro přehledné zobrazení dat a jejich vyhodnocení byl použit pandas viz [18]. K vykreslení grafů pushy morfismů byl využit nástroj Graphvicz viz [19]. Data byla zpracovávána v Jupyter notebooku.

3.4 Vyhodnocení výsledků

Celkově bylo otestováno 2984 různých binárních morfismů. Tabulky uvedené níže poskytují přehled zjištěných vlastností i s konkrétními morfismy.

3.4.1 Souhrn všech vlastností

Tabulka 3.4.1 uvádí počet morfismů s jednotlivými vlastnostmi mezi všemi testovanými morfismy.

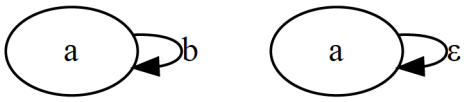
Vlastnost	**Splňuje**	**Nesplňuje**	**Nebylo zjišťováno**
Pushy	88	2986	0
Injektivita	2926	58	0
Repetitivita	2464	462	58
Cirkularita	398	2528	58

Tabulka 3.1: Přehled počtu morfismů s jednotlivými vlastnostmi.

Speciální pozornost si zaslouží morfismus uvedený v kapitole 1.1.1 - Příklad 9., a to Thue-Morseově substituci. Jedná se o morfismus jehož přepisovací pravidla jsou $a \to ab$, $b \to ba$. Vlastnosti tohoto morfismu jsou: non-pushy, injektivní, nerepetitivní a cirkulární se synchronizačním zpožděním 4.

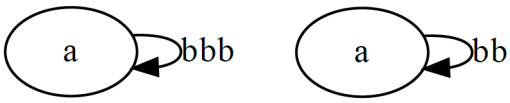
3.4.2 Pushy

U morfismů, které mají vlastnost pushy jsme vykreslili příslušné grafy neomezených písmen. Vzhledem k tomu, že se jedná pouze o binární morfismy a že obraz a je vždy stejně dlouhý nebo delší než obraz b, pak pushy morfismus může vzniknout pouze v případě, že obraz b je b (pravidlo $b \to b$) a obraz a končí nebo začíná na b. Tuto myšlenku potvrzují i výsledky našeho experimentu. Výsledné grafy neomezených písmen souvisí s tvarem pravidla a, neboť jiné neomezené písmeno v binárním morfismu kde $b \to b$ nenajdeme. Na hraně neomezeného grafu zleva vždy najdeme takový počet b, kterým pravidlo a začíná než narazí na první a. Na hraně neomezeného grafu zprava vždy najdeme takový počet b, který následuje za posledním výskytem a. Příklady pushy morfismů a jejich grafů znázorňují obrázky 3.1, 3.2 a 3.3.



Obrázek 3.1: Grafy neomezených písmen zleva (levý) a zprava (pravý) pro morfismus $a \to ba$, $b \to b$.

Obrázek 3.2: Grafy neomezených písmen zleva (levý) a zprava (pravý) pro morfismus $a \to abbbb$, $b \to b$.



Obrázek 3.3: Grafy neomezených písmen zleva (levý) a zprava (pravý) pro morfismus $a \to bbbabb$, $b \to b$.

3.4.3 Injektivita

Nyní zde uvedeme přehled všech neinjektivních morfismů a u každého z nich uvedeme faktory, které mají do dané délky dva a více obrazů.

Velikost n, m	**Morfismus**	**Faktory s 2 a více přesnými předobrazy**
$n = 2$, $m = 1$	$a \to bb$, $b \to b$	bb: $(a, 0)$, $(bb, 0)$
$n = 2$, $m = 2$	$a \to ab$, $b \to ab$	ab: $(b, 0)$, $(a, 0)$
	$a \to ba$, $b \to ba$	ba: $(b, 0)$, $(a, 0)$
	$a \to bb$, $b \to bb$	bb: $(b, 0)$, $(bb, 1)$, (a, 0)
$n = 3$, $m = 3$	$a \to aab$, $b \to aab$	aab: $(a, 0)$, $(b, 0)$
	$a \to aba$, $b \to aba$	aba: $(a, 0)$, $(b, 0)$
	$a \to abb$, $b \to abb$	abb: $(a, 0)$, $(b, 0)$
	$a \to baa$, $b \to baa$	baa: $(b, 0)$, $(a, 0)$
	$a \to bab$, $b \to bab$	bab: (b, 0), (a, 0)
	$a \to bba$, $b \to bba$	bba: $(b, 0)$, $(a, 0)$

Tabulka 3.2: Přehled neinjektivních morfismů s příslušnými faktory, které mají dva a více předků pro $n = 2, 3$.

Velikost n, m	**Morfismus**	**Faktory s 2 a více přesnými předobrazy**
$n = 4$, $m = 2$	$a \to abab$, $b \to ab$	$ababab$: $(ba, 0)$, $(aba, 2)$, $(ab, 0)$
	$a \to baba$, $b \to ba$	$bababa$: $(ba, 0)$, $(aba, 2)$, $(ab, 0)$
$n = 4$, $m = 4$	$a \to aaab$, $b \to aaab$	$aaab$: $(a, 0)$, $(b, 0)$
	$a \to aaba$, $b \to aaba$	$aaba$: $(a, 0)$, $(b, 0)$
	$a \to aabb$, $b \to aabb$	$aabb$: $(a, 0)$, $(b, 0)$
	$a \to abaa$, $b \to abaa$	$abaa$: $(a, 0)$, $(b, 0)$
	$a \to abab$, $b \to abab$	$abab$: $(a, 0)$, $(ab, 2)$, $(ba, 2)$, $(b, 0)$
	$a \to abba$, $b \to abba$	$abba$: $(a, 0)$, $(b, 0)$
	$a \to abbb$, $b \to abbb$	$abbb$: $(a, 0)$, $(b, 0)$
	$a \to baaa$, $b \to baaa$	$baaa$: $(b, 0)$, $(a, 0)$
	$a \to baab$, $b \to baab$	$baab$: $(b, 0)$, $(a, 0)$
	$a \to baba$, $b \to baba$	$baba$: $(b, 0)$, $(ab, 2)$, $(ba, 2)$, $(a, 0)$
	$a \to babb$, $b \to babb$	$babb$: $(b, 0)$, $(a, 0)$
	$a \to bbaa$, $b \to bbaa$	$bbaa$: $(b, 0)$, $(a, 0)$
	$a \to bbab$, $b \to bbab$	$bbab$: $(b, 0)$, $(a, 0)$
	$a \to bbba$, $b \to bbba$	$bbba$: $(b, 0)$, $(a, 0)$

Tabulka 3.3: Přehled neinjektivních morfismů s příslušnými faktory, které mají dva a více předků pro $n = 4$.

Velikost n, m	**Morfismus**	**Faktory s 2 a více přesnými předobrazy**
$n = 5$, $m = 5$	$a \to aaaab$, $b \to aaaab$	$aaaab$: $(a, 0)$, $(b, 0)$
	$a \to aaaba$, $b \to aaaba$	$aaaba$: $(a, 0)$, $(b, 0)$
	$a \to aaabb$, $b \to aaabb$	$aaabb$: $(a, 0)$, $(b, 0)$
	$a \to aabaa$, $b \to aabaa$	$aabaa$: $(a, 0)$, $(b, 0)$
	$a \to aabab$, $b \to aabab$	$aabab$: $(a, 0)$, $(b, 0)$
	$a \to aabba$, $b \to aabba$	$aabba$: $(a, 0)$, $(b, 0)$
	$a \to aabbb$, $b \to aabbb$	$aabbb$: $(a, 0)$, $(b, 0)$
	$a \to abaaa$, $b \to abaaa$	$abaaa$: $(a, 0)$, $(b, 0)$
	$a \to abaab$, $b \to abaab$	$abaab$: $(a, 0)$, $(b, 0)$
	$a \to ababa$, $b \to ababa$	$ababa$: $(a, 0)$, $(b, 0)$
	$a \to ababb$, $b \to ababb$	$ababb$: $(a, 0)$, $(b, 0)$
	$a \to abbaa$, $b \to abbaa$	$abbaa$: $(a, 0)$, $(b, 0)$
	$a \to abbab$, $b \to abbab$	$abbab$: $(a, 0)$, $(b, 0)$
	$a \to abbba$, $b \to abbba$	$abbba$: $(a, 0)$, $(b, 0)$
	$a \to abbbb$, $b \to abbbb$	$abbbb$: $(a, 0)$, $(b, 0)$
	$a \to baaaa$, $b \to baaaa$	$baaaa$: $(b, 0)$, $(a, 0)$
	$a \to baaab$, $b \to baaab$	$baaab$: $(b, 0)$, $(a, 0)$
	$a \to baaba$, $b \to baaba$	$baaba$: $(b, 0)$, $(a, 0)$
	$a \to baabb$, $b \to baabb$	$baabb$: $(b, 0)$, $(a, 0)$
	$a \to babaa$, $b \to babaa$	$babaa$: $(b, 0)$, $(a, 0)$
	$a \to babab$, $b \to babab$	$babab$: $(b, 0)$, $(a, 0)$
	$a \to babba$, $b \to babba$	$babba$: $(b, 0)$, $(a, 0)$
	$a \to babbb$, $b \to babbb$	$babbb$: $(b, 0)$, $(a, 0)$
	$a \to bbaaa$, $b \to bbaaa$	$bbaaa$: $(b, 0)$, $(a, 0)$
	$a \to bbaab$, $b \to bbaab$	$bbaab$: $(b, 0)$, $(a, 0)$
	$a \to bbaba$, $b \to bbaba$	$bbaba$: $(b, 0)$, $(a, 0)$
	$a \to bbabb$, $b \to bbabb$	$bbabb$: $(b, 0)$, $(a, 0)$
	$a \to bbbaa$, $b \to bbbaa$	$bbbaa$: $(b, 0)$, $(a, 0)$
	$a \to bbbab$, $b \to bbbab$	$bbbab$: $(b, 0)$, $(a, 0)$
	$a \to bbbba$, $b \to bbbba$	$bbbba$: $(b, 0)$, $(a, 0)$

Tabulka 3.4: Přehled neinjektivních morfismů s příslušnými faktory, které mají dva a více předků pro $n = 5$.

Velikost n, m	**Morfismus**	**Faktory s 2 a více přesnými předobrazy**
$n = 6$	$a \to ababab$, $b \to ab$	$ababab$: (ba, 0), (aba, 4), (ab, 0), (aba, 2)
$m = 2$	$a \to bababa$, $b \to ba$	$bababa$: (ba, 0), (aba, 4), (ab, 0), (aba, 2)

Tabulka 3.5: Přehled neinjektivních morfismů s příslušnými faktory, které mají dva a více předků pro $n = 6$.

Z tabulek je patrné, že pro morfismy, ve kterých je délka $n = m$ jsou neinjektivními morfismy pouze ty, kdy je obraz a i b stejný. V případech, kdy je n sudé a $m = 2$, jsou neinjektivní morfismy takové, kde obraz a je mocninou obrazu b. Speciálně pro $n = 2$ a $m = 1$ je neinjektivní i morfismus $a \to bb$, $b \to b$. Další morfismy pro $m = 1$ nejsou injektivní na $\mathcal{A}^*$, my však v algoritmech vše zkoumáme na $S(L(G))$ a na této množině jsou tyto morfismy injektivní a tak je správně vyhodnotí i náš algoritmus.

3.4.4 Repetitivita

Zde uvedeme přehled toho, kolik repetitivních/nerepetetivních morfismů jsme nalezli pro jednotlivé hodnoty n, m a vždy uvedeme příklad. Tato vlastnost nebyla vůbec vyhodnocována pro neinjektivní morfismy.

Velikost n, m	**Repetitivní**	**Příklad repetitivní**	**Nerepetitivní**	**Příklad nerepetitivní**
$n = 2$, $m = 1$	3	$a \to aa$, $b \to a$	4	$a \to ab$, $b \to b$
$n = 2$, $m = 2$	10	$a \to bb$, $b \to aa$	3	$a \to ab$, $b \to ba$
$n = 3$, $m = 1$	9	$a \to bba$, $b \to b$	6	$a \to bab$, $b \to b$
$n = 3$, $m = 2$	23	$a \to aba$, $b \to ab$	9	$a \to aba$, $b \to ba$
$n = 3$, $m = 3$	49	$a \to aab$, $b \to abb$	9	$a \to abb$, $b \to baa$
$n = 4$, $m = 1$	20	$a \to aabb$, $b \to a$	12	$a \to bbaa$, $b \to b$
$n = 4$, $m = 2$	49	$a \to abaa$, $b \to aa$	13	$a \to abba$, $b \to ab$
$n = 4$, $m = 3$	102	$a \to aaab$, $b \to aba$	26	$a \to abaa$, $b \to abb$
$n = 4$, $m = 4$	217	$a \to aaaa$, $b \to abba$	25	$a \to abab$, $b \to aaaa$

Tabulka 3.6: Přehled počtu repetitivních/nerepetitivních morfismů s příklady morfismů pro $n = 2, 3, 4$.

Velikost n, m	**Repetitivní**	**Příklad repetitivní**	**Nerepetitivní**	**Příklad nerepetitivní**
$n = 5$, $m = 1$	8	$a \to aaaaa$, $b \to a$	16	$a \to baaab$, $b \to b$
$n = 5$, $m = 2$	96	$a \to aaaba$, $b \to ba$	32	$a \to abaab$, $b \to ba$
$n = 5$, $m = 3$	215	$a \to aaaaa$, $b \to bbb$	41	$a \to ababb$, $b \to aaa$
$n = 5$, $m = 4$	418	$a \to abaab, b \to abbb$	94	$a \to abaaa$, $b \to aabb$
$n = 5$, $m = 5$	932	$a \to abbbb$, $b \to babbb$	62	$a \to abbbb$, $b \to aaabb$
$n = 6$, $m = 1$	76	$a \to bababa$, $b \to b$	52	$a \to ababaa$, $b \to b$
$n = 6$, $m = 2$	202	$a \to aaaaab$, $b \to ba$	52	$a \to ababab$, $b \to aa$

Tabulka 3.7: Přehled počtu repetitivních/nerepetitivních morfismů s příklady morfismů pro $n = 5, 6$.

Z přehledu je zřejmé, že počet repetitivních morfismů se zvyšuje v závislosti na součtu $n + m$. Čím vyšší je součet, tím větší počet repetitivních morfismů byl nalezen. Dále můžeme usuzovat, že počet repetitivních morfismů se zvyšuje vzhledem k menšímu rozdílu mezi n a m, tedy například jejich počet pro $(4, 3)$ je větší než pro $(6, 1)$, přestože součet $n + m$ je v obou případech 7.

3.4.5 Cirkularita

V této sekci ukážeme cirkulární morfismy včetně jejich synchronizačního zpoždění (v tabulkách použijeme zkartku SD = synchronization delay) s informací o tom, jaký byl nejdelší nesynchronizovaný faktor.

Velikost n, m	**Morfismus**	**SD**	**Nejdelší nesynchronizované faktory**
$n = 2$, $m = 1$	$a \to ab$, $b \to a$	2	N/A
	$a \to ba$, $b \to a$	2	N/A
$n = 2, m = 2$	$a \to ab$, $b \to aa$	3	aa: $(ba, 1)$, $(b, 0)$
	$a \to ab$, $b \to ba$	4	aba: $(aa, 0)$, $(bb, 1)$, bab: $(aa, 1)$, $(bb, 0)$
	$a \to bb$, $b \to ba$	3	bb: $(ab, 1)$, $(a, 0)$

Tabulka 3.8: Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 2$.

Velikost n, m	**Morfismus**	**SD**	**Nejdelší nesynchronizované faktory**
$n = 3$, $m = 1$	$a \to aba$, $b \to a$	2	N/A
	$a \to abb$, $b \to a$	2	N/A
	$a \to bbb$, $b \to b$	3	bb: $(bb, 0)$, $(a, 1)$, $(a, 0)$
$n = 3$, $m = 2$	$a \to aba$, $b \to aa$	4	aaa: $(ba, 0)$, $(ab, 2)$
	$a \to aba$, $b \to ba$	2	N/A
	$a \to abb$, $b \to aa$	5	$aaaa$: $(bba, 1)$, $(bb, 0)$
	$a \to abb$, $b \to ab$	3	ab: $(a, 0)$, $(b, 0)$
	$a \to abb$, $b \to ba$	4	bba: $(ab, 2)$, $(aa, 1)$
	$a \to baa$, $b \to ba$	3	ba: $(a, 0)$, $(b, 0)$
	$a \to bab$, $b \to ba$	4	bab: $(ba, 0)$, $(bb, 0)$, $(a, 0)$
	$a \to bba$, $b \to ba$	2	N/A
	$a \to bbb$, $b \to ba$	4	bbb: $(ab, 1)$, $(a, 0)$
$n = 3$, $m = 3$	$a \to aba$, $b \to aab$	5	$aaba$: $(ba, 0)$, $(aa, 2)$
	$a \to abb$, $b \to aaa$	7	$aaaaaa$: $(bba, 1)$, $(bb, 0)$
	$a \to abb$, $b \to aab$	3	ab: $(b, 1)$, $(a, 0)$
	$a \to abb$, $b \to aba$	3	ba: $(ab, 2)$, $(b, 1)$, $(aa, 2)$
	$a \to abb$, $b \to baa$	4	aab: $(ba, 2)$, $(bb, 1)$, bba: $(ab, 2)$, $(aa, 1)$
	$a \to bab$, $b \to baa$	3	ab: $(a, 1)$, $(bb, 2)$, $(ba, 2)$
	$a \to bba$, $b \to baa$	3	ba: $(a, 1)$, $(b, 0)$
	$a \to bba$, $b \to bab$	5	$bbab$: $(bb, 2)$, $(ab, 0)$
	$a \to bbb$, $b \to baa$	7	$bbbbbb$: $(aa, 0)$, $(aab, 1)$

Tabulka 3.9: Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 3$.

Pro další hodnoty n, m (tedy pro dvojice $(4, 1) - (6, 2)$) začíná být počet cirkulárních morfismů již větší, proto nebudeme uvádět všechny morfismy, ale vybereme jen ty, jejichž synchronizační zpoždění je větší nebo rovno než součet $n + m$.

Z výsledků je patrné, že hodnota synchronizačního zpoždění je ve většině případů celkem malá, což je v souladu s předpokladem. Objevují se zde i vyšší hodnoty, v žádném případě však nebylo třeba navyšovat délku nalezených faktorů nad délku zadanou při vstupu. Mezi morfismy, které jsme otestovali, nepřesáhla hodnota synchronizačního zpoždění hodnotu 30, nejvyšší nalezené synchronizační zpoždění bylo 29. Z analýzy výsledků by se dalo usuzovat, že velikost synchronizačního zpoždění souvisí s délkou obrazů, neboť se zvyšujícím se n, m se začínají objevovat vyšší hodnoty synchronizačního zpoždění.

Velikost n, m	**Morfismus**	**SD**	**Nejdelší nesynchronizované faktory**
$n = 4$, $m = 1$	$a \to abab$, $b \to a$	8	$abababa$: $(aa, 0)$, $(aab, 2)$
$n = 4$, $m = 2$	$a \to abbb$, $b \to aa$	7	$aaaaaa$: $(bbba, 1)$, $(bbb, 0)$
	$a \to abbb$, $b \to ba$	6	$bbbab$: $(abb, 2)$, $(aa, 1)$
	$a \to baba$, $b \to ab$	7	$bababa$: $(aa, 0)$, $(aa, 2)$
	$a \to bbab$, $b \to ba$	13	$babbabbabbab$: $(ababb, 1)$, $(baba, 0)$
$n = 4$, $m = 3$	$a \to abaa$, $b \to aab$	6	$aaaba$: $(aba, 3)$, $(aa, 2)$
	$a \to abab$, $b \to aab$	10	$ababababa$: $(baa, 1)$, $(aab, 0)$
	$a \to abab$, $b \to aba$	10	$ababababa$: $(aab, 2)$, $(aab, 0)$
	$a \to abab$, $b \to bab$	6	$babab$: $(ba, 0)$, $(ba, 2)$
	$a \to abba$, $b \to bab$	16	$bababbababbabab$: $(abbab, 2)$, $(babba, 0)$
	$a \to abbb$, $b \to aaa$	10	$aaaaaaaaa$: $(bbba, 1)$, $(bbb, 0)$
	$a \to baba$, $b \to aba$	10	$ababababa$: $(baa, 0)$, $(baa, 2)$
	$a \to baba$, $b \to baa$	10	$ababababa$: $(baa, 2)$, $(aab, 1)$
	$a \to baba$, $b \to bab$	6	$babab$: $(ab, 2)$, $(ab, 0)$
	$a \to bbbb$, $b \to baa$	9	$bbbbbbb$: $(aa, 0)$, $(aab, 1)$

Tabulka 3.10: Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 4$, $m = 1, 2, 3$.

Velikost n, m	**Morfismus**	**SD**	**Nejdelší nesynchronizované faktory**
	$a \to abaa$, $b \to aaab$	6	$aaaba$: (ba, 0), (aa, 2)
	$a \to abab$, $b \to aaaa$	20	$abababababababababa$: ($aaaaa$, 0), ($aaaaab$, 2)
	$a \to abab$, $b \to abaa$	14	$abababababaa$: ($aaab$, 2), ($aaab$, 0)
	$a \to abab$, $b \to baaa$	15	$abababababab$: ($aaaa$, 2), ($aaaa$, 0)
$n = 4$	$a \to abbb$, $b \to aaaa$	13	$aaaaaaaaaaaa$: ($bbba$, 1), (bbb, 0)
$m = 4$	$a \to abbb$, $b \to abab$	14	$bababababab$: ($bbba$, 1), ($abbb$, 3)
	$a \to abbb$, $b \to baba$	15	$bababababaa$: ($bbbb$, 2), ($bbbb$, 0)
	$a \to baba$, $b \to baaa$	14	$abababababaa$: ($baaa$, 3), ($aaab$, 1)
	$a \to babb$, $b \to baba$	14	$bababababab$: ($bbba$, 0), ($bbba$, 2)
	$a \to bbbb$, $b \to baaa$	13	$bbbbbbbbbbb$: (aaa, 0), ($aaab$, 1)
	$a \to bbbb$, $b \to baba$	20	$bababababababababab$: ($bbbbb$, 0), ($bbbbba$, 2)

Tabulka 3.11: Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 4, m = 4$.

Velikost n, m	**Morfismus**	**SD**	**Nejdelší nesynchronizované faktory**
$n = 5$, $m = 1$	$a \to abaaa$, $b \to b$	6	$aaaba$: $(aa, 3)$, $(aba, 2)$
	$a \to abbaa$, $b \to b$	6	$aabba$: $(aa, 4)$, $(abba, 3)$
	$a \to baabb$, $b \to a$	6	$bbaab$: $(aa, 4)$, $(abba, 3)$
$n = 5$, $m = 2$	$a \to abaab$, $b \to ba$	7	$abaaba$: $(aa, 0)$, $(bba, 1)$, $babaab$: $(aa, 4)$, $(bba, 0)$
	$a \to abbab$, $b \to ba$	7	$abbaba$: $(aa, 0)$, $(abb, 3)$, $babbab$: $(abb, 2)$, $(aa, 4)$
	$a \to abbbb$, $b \to aa$	9	$aaaaaaaa$: $(bbbba, 1)$, $(bbbb, 0)$
	$a \to abbbb$, $b \to ba$	7	$bbbbab$: $(abb, 2)$, $(aa, 1)$
$n = 5$, $m = 3$	$a \to abbbb$, $b \to aaa$	13	$aaaaaaaaaaaa$: $(bbbba, 1)$, $(bbbb, 0)$
	$a \to babba$, $b \to abb$	8	$bbabbab$: $(bbb, 1)$, $(baa, 2)$
	$a \to bbaab$, $b \to baa$	11	$bbaabbaabb$: $(abaa, 4)$, $(aba, 0)$
	$a \to bbbbb$, $b \to baa$	11	$bbbbbbbbbb$: $(aa, 0)$, $(aab, 1)$

Tabulka 3.12: Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 5, m = 1, 2, 3$.

Velikost n, m	**Morfismus**	**SD**	**Nejdelší nesynchronizované faktory**
	$a \to ababb$, $b \to aaaa$	9	$aaaaaaaa$: $(bba, 1)$, $(bb, 0)$
	$a \to ababb$, $b \to abab$	12	$bababababab$: $(bba, 1)$, $(abba, 4)$
	$a \to abbaa$, $b \to abab$	9	$abababab$: $(bba, 2)$, $(bb, 0)$
	$a \to abbaa$, $b \to baab$	9	$aabbaaba$: $(bba, 1)$, $(aab, 4)$
	$a \to abbba$, $b \to abab$	13	$abababababab$: $(bbba, 2)$, $(bbb, 0)$
	$a \to abbbb$, $b \to aaaa$	17	$aaaaaaaaaaaaaaaa$: $(bbbba, 1)$, $(bbbb, 0)$
	$a \to abbbb$, $b \to abab$	18	$bababababababab$: $(abbbb, 4)$, $(bbbba, 1)$
	$a \to abbbb$, $b \to baba$	19	$babababababababa$: $(bbbbb, 0)$, $(bbbbb, 2)$
$n = 5$	$a \to baaab$, $b \to abab$	9	$abababab$: $(abb, 3)$, $(bb, 0)$
$m = 4$	$a \to baaab$, $b \to baba$	9	$babababa$: $(bba, 2)$, $(bb, 0)$
	$a \to baaba$, $b \to baba$	8	$abababa$: $(aba, 4)$, $(ab, 2)$
	$a \to babab$, $b \to baba$	12	$bababababab$: $(bba, 2)$, $(bba, 0)$
	$a \to babbb$, $b \to abab$	12	$bababababab$: $(bbb, 1)$, $(abbb, 4)$
	$a \to babbb$, $b \to baba$	18	$bababababababab$: $(bbbba, 0)$, $(bbbba, 2)$,
	$a \to bbaab$, $b \to baba$	12	$bababababab$: $(bbba, 2)$, $(bbb, 0)$
	$a \to bbaba$, $b \to baba$	12	$bababababab$: $(abb, 1)$, $(abba, 3)$
	$a \to bbbab$, $b \to baba$	12	$bababababab$: $(bbba, 2)$, $(bbb, 0)$
	$a \to bbbbb$, $b \to baaa$	16	$bbbbbbbbbbbbbbb$: $(aaa, 0)$, $(aaab, 1)$
	$a \to bbbbb$, $b \to baba$	24	$bababababababababababab$: $(bbbbbba, 2)$, $(bbbbbb, 0)$

Tabulka 3.13: Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 5, m = 4$.

Velikost n, m	**Morfismus**	**SD**	**Nejdelší nesynchronizované faktory**
	$a \to ababb$, $b \to aaaaa$	11	$aaaaaaaaaa$: $(bba, 1)$, $(bb, 0)$
$n = 5$	$a \to abbbb$, $b \to aaaaa$	21	$aaaaaaaaaaaaaaaaaaaa$: $(bbbba, 1)$, $(bbbb, 0)$
$m = 5$	$a \to bbbbb$, $b \to baaaa$	21	$bbbbbbbbbbbbbbbbbbb$: $(aaaab, 1)$, $(aaaa, 0)$
	$a \to bbbbb$, $b \to babaa$	11	$bbbbbbbbb$: $(aa, 0)$, $(aab, 1)$

Tabulka 3.14: Přehled cirkulárních morfismů s jejich synchronizačním zpožděním včetně ukázky nejdelšího nesynchronizovaného faktoru pro $n = 5, m = 5$.

Velikost n, m	**Morfismus**	**SD**	**Nejdelší nesynchronizované faktory**
	$a \to abaaaa$, $b \to b$	7	$aaaaba$: $(aa, 3)$, $(aba, 2)$
	$a \to abaaba$, $b \to b$	10	$abaabaaba$: $(aa, 0)$, $(aa, 3)$
	$a \to ababaa$, $b \to b$	7	$aababa$: $(aba, 4)$, $(aa, 5)$
	$a \to ababab$, $b \to a$	12	$abababababa$: $(aa, 0)$, $(aab, 2)$
$n = 6$	$a \to ababba$, $b \to b$	9	$babbabab$: $(abba, 4)$, $(aba, 1)$
$m = 1$	$a \to abbaaa$, $b \to b$	7	$aaabba$: $(aa, 4)$, $(abba, 3)$
	$a \to abbaba$, $b \to b$	9	$bababbab$: $(aba, 4)$, $(abba, 2)$
	$a \to abbabb$, $b \to a$	17	$abbabbabbabbabba$: $(aaab, 3)$, $(aaa, 0)$
	$a \to abbbaa$, $b \to b$	7	$aabbba$: $(abbba, 4)$, $(aa, 5)$
	$a \to babaab$, $b \to a$	9	$abaababa$: $(abba, 4)$, $(aba, 1)$
	$a \to bababa$, $b \to a$	12	$abababababa$: $(baa, 0)$, $(aa, 1)$
	$a \to babbbb$, $b \to a$	7	$bbbbab$: $(aa, 3)$, $(aba, 2)$

Tabulka 3.15: Přehled cirkulárních morfismů s jejich synchronizačním zpožděním s informací o nejdelším nesynchronizovaném faktoru pro $n = 6$, $m = 1$.

Velikost n, m	**Morfismus**	**SD**	**Nejdelší nesynchronizované faktory**
	$a \to abaaba$, $b \to ab$	13	$abaabaabaaba$: $(aa, 0)$, $(aaba, 3)$
	$a \to ababab$, $b \to aa$	17	$abababababababab$: $(aaa, 2)$, $(aaa, 0)$
	$a \to abbaab$, $b \to ba$	15	$baabbaabbaabba$: $(abab, 2)$, $(baba, 0)$
	$a \to abbabb$, $b \to aa$	29	$abbabbabbabbabbabbabbabbabba$: $(aaaaab, 3)$, $(aaaaa, 0)$
	$a \to abbabb$, $b \to ba$	10	$abbabbabb$: $(aa, 0)$, $(aa, 3)$
$n = 6$	$a \to abbbbb$, $b \to aa$	11	$aaaaaaaaaa$: $(bbbbb, 0)$, $(bbbbba, 1)$
$m = 2$	$a \to abbbbb$, $b \to ba$	8	$bbbbbab$: $(aa, 1)$, $(abb, 2)$
	$a \to baabaa$, $b \to ba$	13	$abaabaabaaba$: $(aab, 2)$, $(baa, 1)$
	$a \to baabba$, $b \to ab$	15	$abbaabbaabbaab$: $(abab, 2)$, $(baba, 0)$
	$a \to bababb$, $b \to ab$	27	$babbababbababbababbababbab$: $(ababbabba, 5)$, $(abbabbab, 2)$
	$a \to bbabab$, $b \to ba$	27	$babbababbababbababbababbab$: $(babbabba, 0)$, $(abbabbaba, 3)$
	$a \to bbbabb$, $b \to ba$	23	$babbbabbbabbbabbbabbba$: $(ababab, 2)$, $(bababa, 0)$

Tabulka 3.16: Přehled cirkulárních morfismů s jejich synchronizačním zpožděním s informací o nejdelším nesynchronizovaném faktoru pro $n = 6, m = 2$.

Závěr

Zadáním práce bylo seznámit se s teorií D0L-systémů a s potřebnými pojmy z kombinatoriky na slovech. Dále z doporučené literatury nastudovat algoritmy pro zjišťování vybraných vlastností D0L-systémů, zejména cirkularitu, repetitivitu, a pushy D0L-systémy. Dalším úkolem bylo pokusit se co nejefektivněji implementovat tyto algoritmy v jazyce Python, ideálně v takovém formátu, aby bylo možné je zařadit do knihoven programu SageMath. Posledním úkolem bylo seznámit se s injektivitou D0L-systémů a pokusit se navrhnout a implementovat algoritmus (brute-force) pro její ověření.

Všechny potřebné pojmy z oblasti kombinatoriky na slovech i z teorie D0L-systémů jsme představili a ukázali na příkladech. Všechny požadované vlastnosti jsme nastudovali a v práci podrobně vysvětlili. Stejně tak byly nastudovány jednotlivé vlastnosti, které jsme následně implementovali v jazyce Python. V jazyce Python jsme též implementovali námi navržený brute-force algoritmus pro zjištění injektivity D0L-systémů. Všechny tyto algoritmy jsme v práci vysvětlili, ukázali na příkladu a popsali pomocí pseudokódu. Všechny implementované funkce v programech v Pythonu jsme opatřili komentáři, odpovídajícími konvenci pro knihovny programu SageMath. K finálnímu zařazení do knihoven programu SageMath však nedošlo, z důvodu nutnosti provedení dodatečných změn konceptu algoritmů, jež by značně přesahovalo rozsah práce.

Implementované algoritmy byly otestovány pro vybrané binární morfismy a na základě výsledků experimentu byl vytvořen přehled vybraných morfismů s jejich vlastnostmi.

Pokračováním práce by mohlo být otestování implementovaných algoritmů na dalších typech morfismů či implementování dalších zajímavých vlastností. Další vývoj by mohl být směřován právě k úpravám všech algoritmů na přesnou konvenci knihoven SageMath, aby mohly být následně do těchto knihoven zařazeny.

Literatura

[1] KLOUDA, K.: Non-standard numeration systems and combinatorics on words, disertační práce ČVUT, 2010.

[2] KLOUDA K.; STAROST, Š.: *An Algorithm Enumerating All Infinite Repetitions in a D0L System*, Journal of Discrete Algorithms, 33, 2015, p.130–138.

[3] KLOUDA, K.; STAROSTA, Š.: *Characterization of circular D0L systems*, Theoretical Computer Science, 790, 2019, p.131-137.

[4] MEDKOVÁ, K.: *Bispeciální faktory v jazycích S-adických sytémů*, diplomová práce FJFI, 2016.

[5] CENTRE, T.; CHOFFRUT, C.; KARHUMAKI, J.*Combinatorics of words*, 1997, doi:10.1007/978-3-642-59136-5_6.

[6] KABÁTOVÁ, R.: *Základy kombinatoriky na slovech*, b.akalářská práce Masarykova Univerzita - Přírodovědecká fakulta, 2009.

[7] An Introduction to Lindenmayer System, [cit. 16.9.2020] Dostupné z: http://www1.biologie.uni-hamburg.de/b-online/e283/lsys.html

[8] SÉÉBOLD,P.: *Sequences generated by infinitely iterated morphisms*. Discrete Appl. Math., 11(3), 1985, p. 255-264.

[9] EHRENFEUCHT, A.; ROZENBERG, G.: *Repetition of subwords in D0L languages* .Inform. Comput., 53, 1983, p.13–35.

[10] EHRENFEUCHT, A.; ROZENBERG, G.: *Elementary homomorphisms and a solution of the D0L sequence equivalence problem*. Theoretical Computer Science, 7(2), 1978, p.169-183.

[11] KOBAYASHI,Y.; OTTO, F.: *Repetitiveness of languages generated by morphisms.*Theoret. Comput. Sci., 240(2), 2000, P. 337–378.

[12] MIGNOSI, F.; SÉÉBOLD, P.: *If a D0L language is k-power freethen it is circular*. ICALP '93: Proceedings of the 20th International Colloquim on Automata, Languages and Programming, London - UK, 1993. Sprige-Verlag, p.507–518

[13] KŮRKA, P.: *Topological and Symbolic Dynamics*. Societé Math ematique de France, 2003.

[14] LANDO, B.: Periodicity and ultimate periodicity of D0L systems. Theoret. Comput. Sci., 82(1), 1991, p.19–33.

[15] CASSAIGNE, J.: *An algorithm to test if a given circular HD0L-language avoids a pattern*. IFIP World Computer Congress '94, Elsevier, 1994, p.459–464.

[16] SAGE DEVELOPERS´ GUIDE, v.9.1, [cit. 16.9.2020] Dostupné z: https://doc.sagemath.org/html/en/developer/coding_basics.html

[17] JSON encoder and decoder, [cit. 16.9.2020] Dostupné z: https://docs.python.org/3/library/json.html

[18] PANDAS 1.1.2 documentation, [cit. 16.9.2020] Dostupné z: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_json.html

[19] API REFERENCE - GRAPHVIZ 0.14.1 documentation, [cit. 16.9.2020] Dostupné z: ?https://graphviz.readthedocs.io/en/stable/api.html

Příloha **A**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `DP_Štěpánková_Anežka_2020.tex` ... zdrojová forma práce ve formátu LaTeX
  - `pictures` ..... obrázky k práci
  - `impl`
    - `***.py` ..... zdrojové kódy implementace
    - `how_to.txt` ..... návod na spuštění zdrojových kódů
- `text` ..... text práce
  - `DP_Štěpánková_Anežka_2020.pdf` ..... text práce ve formátu PDF
  - `ZZP.pdf` ..... zadání práce ve formátu PDF