Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

# DIPLOMOVÁ PRÁCE

Limitované automaty

2023

Jaroslav Večeřa

Vedoucí práce:
doc. RNDr. Tomáš Masopust,
Ph.D., DSc.

Studijní program: Aplikovaná informatika,
Specializace: Vývoj software

**Bibliografické údaje**

| | |
|---|---|
| Autor: | Jaroslav Večeřa |
| Název práce: | Limitované automaty |
| Typ práce: | diplomová práce |
| Pracoviště: | Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci |
| Rok obhajoby: | 2023 |
| Studijní program: | Aplikovaná informatika, Specializace: Vývoj software |
| Vedoucí práce: | doc. RNDr. Tomáš Masopust, Ph.D., DSc. |
| Počet stran: | 60 |
| Přílohy: | elektronická data v úložišti katedry informatiky |
| Jazyk práce: | český |

**Bibliographic info**

| | |
|---|---|
| Author: | Jaroslav Večeřa |
| Title: | Limited automata |
| Thesis type: | master thesis |
| Department: | Department of Computer Science, Faculty of Science, Palacký University Olomouc |
| Year of defense: | 2023 |
| Study program: | Applied Computer Science, Specialization: Software Development |
| Supervisor: | doc. RNDr. Tomáš Masopust, Ph.D., DSc. |
| Page count: | 60 |
| Supplements: | electronic data in the storage of department of computer science |
| Thesis language: | Czech |

## Anotace

*Diplomová práce se zabývá limitovanými automaty a jazyky, které tyto automaty rozpoznávají. Dále definuje a zjišťuje výpočetní sílu nové verze tohoto modelu – vícepáskových limitovaných automatů.*

## Synopsis

*The diploma thesis deals with the limited automata and the languages they recognize. Furthermore, it defines and explores the computational power of a new version of this model – the multi-tape limited automata.*

**Klíčová slova:** automat; limitovaný automat; regulární jazyky; bezkontextové jazyky; kontextové jazyky;

**Keywords:** automata; limited automata; regular languages; context-free languages; context-sensitive languages;

Děkuji mému vedoucímu doc. RNDr. Tomáši Masopustovi, Ph.D., DSc. za ochotu a dobré rady při konzultacích. Rodině za trpělivost a podporu.

*Odevzdáním tohoto textu jeho autor místopřísežně prohlašuje, že celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.*

# Obsah

# Seznam tabulek

# 1 Úvod

Jedním ze základních modelů klasické teorie automatů je lineárně omezený automat (LBA), který rozpoznává třídu kontextových jazyků (CSL). Omezíme-li počet přepisů pásky, dostáváme model automatu zvaný limitovaný automat. Druhým pohledem by bylo rozšíření nedeterministických dvoucestných automatů o daný počet přepisů $k$. Tento parametr $k$ je součástí konkrétního modelu limitovaného automatu (značíme $k$-LA) a určuje počet přepisů pro každé políčko pásky (nezávisle na počtu přepisů ostatních políček).

Výpočetní síla těchto automatů je známá a budeme se jí věnovat v první části této práce. Z předchozího popisu lze vidět, že třída jazyků, které dokáží limitované automaty rozpoznávat, obsahuje třídu regulárních jazyků, ale není větší než třída kontextových jazyků.

Výpočetní síla konkrétních modelů detailněji zobrazuje následující tabulka.

Tabulka 1: Rozpoznávané jazyky LA

| | 0-LA | 1-LA | 2-LA | 3-LA | 4-LA | $\cdots$ |
|---|---|---|---|---|---|---|
| deterministický | REG | | DCFL | 3-det. | 4-det. | $\cdots$ |
| nedeterministický | REG | | CFL | | | |

Zajímavé je, že pro nedeterministické limitované automaty s žádným, nebo jedním přepisem na políčko, je třída rozpoznávaných jazyků pořád regulární a s libovolným větším konečným počtem přepisů je bezkontextová. Složitější jazyky rozpoznat s konečným počtem přepisů nelze. Rozpoznávané jazyky deterministických limitovaných jazyků jsou zajímavější. Od $k = 2$ tvoří nekonečnou hierarchii počínaje deterministickými bezkontextovými jazyky (DCFL). Tato hierarchie je opět omezena bezkontextovými jazyky a přesnější horní odhad nemáme k dispozici. Třídy $i$-det vysvětlíme později.

Některé z těchto vztahů si později dokážeme za pomoci vzájemných simulací s automaty, u kterých je výpočetní síla dobře známá.

## 1.1 Vícepáskové automaty

Jednou ze základních vlastností, která se zkoumá u Turingových strojů, či lineárně omezených automatů, je ekvivalence s různými verzemi tohoto modelu. Například ekvivalence LBA s vícepáskovým LBA je velice známá konstrukce. Z tohoto faktu dokonce plyne název automatu. Více pásek, případně lineárně dlouhá páska vzhledem k délce vstupu, lze totiž snadno simulovat pomocí jedné pásky délky vstupu. U limitovaných automatů však stejný postup nelze použít kvůli omezení na počet přepisů pásky.

To otevírá celou řadu otázek. V druhé části této práce si formálně představíme vícepáskové limitované automaty a pokusíme se některé z otázek zodpovědět. Nyní můžeme naši tabulku rozšířit o nové modely a doplnit některé údaje.

Tabulka 2: Deterministické vícepáskové limitované automaty

| deterministické | 0-LA | 1-LA | 2-LA | 3-LA | 4-LA | 5-LA |
|---|---|---|---|---|---|---|
| 1 páska | REG | REG | DCFL | 3-det. | 4-det. | 5-det. |
| 2 pásky | $L_{1\text{pal}} \mid a^{2^n}$ | ? | ? | ? | ? | ? |
| 3 pásky | $L_{2\text{pal}} \mid a^p$(prvočísla) | ? | ? | ? | ? | ? |
| 4 pásky | $L_{3\text{pal}}$ | ? | ? | ? | ? | ? |
| 5 pásek | $L_{4\text{pal}}$ | ? | ? | ? | ? | ? |

Tabulka 3: Nedeterministické vícepáskové limitované automaty

| nedeterministické | 0-LA | 1-LA | 2-LA | 3-LA | 4-LA | 5-LA |
|---|---|---|---|---|---|---|
| 1 páska | REG | REG | CFL | CFL | CFL | CFL |
| 2 pásky | ? | $\supseteq$ CFL | ? | ? | ? | ? |
| 3 pásky | ? | ? | ? | ? | ? | ? |
| 4 pásky | ? | ? | ? | ? | ? | ? |
| 5 pásek | ? | ? | ? | ? | ? | ? |

Jazyky $L_{i\text{pal}}$ představíme později a dokážeme si, že odpovídající automaty je skutečně rozpoznávají, tyto jazyky byly navrženy tak, aby je automaty s méně páskami nedokázaly rozpoznat, formální důkaz se však zatím nepodařilo najít.

# 2 Definice

Představme si nyní základní pojmy a značení, které budeme v této práci potřebovat.

**Definice 1**

Konečná posloupnost $b_1, b_2, \ldots, b_m$ se nazývá *podposloupnost* konečné posloupnosti $a_1, a_2, \ldots, a_n$, právě když existuje posloupnost přirozených čísel $k_1 < k_2 < \cdots < k_m$ s $k_m \leq n$ tak, že pro všechna $i$, $1 \leq i \leq m$ je $b_i = a_{k_i}$.

## 2.1 Formální jazyky

**Definice 2**

*Abecedou* myslíme libovolnou neprázdnou konečnou množinu, její prvky nazýváme symboly. Často se značí $\Sigma$.
*Slovem*, nebo *řetězcem* nad abecedou $\Sigma$ myslíme libovolnou konečnou posloupnost symbolů z $\Sigma$, často tuto posloupnost zapisujeme bez oddělovačů, tzn. místo $a, b, c$ píšeme $abc$. Prázdnou posloupnost potom zapisujeme znakem $\epsilon$.

Proměnné označující nějaká slova pak často značíme písmeny $u, v, w$ a délku slova $w$ označujeme $|w|$. Důležitým pojmem je také jazyk, pro jeho definici ale bude potřeba definovat výraz $\Sigma^*$, ten značí množinu všech slov nad abecedou $\Sigma$.

**Definice 3**

*Reverzem* slova $w = a_1 a_2 \dots a_n$ se rozumí slovo $w^{\mathrm{R}} = a_n \dots a_2 a_1$.
*Zřetězením* dvou řetězců $w_1 = a_1 a_2 \dots a_m$ nad abecedou $\Sigma_1$ a $w_2 = b_1 b_2 \dots b_n$ nad abecedou $\Sigma_2$ rozumíme řetězec $w = a_1 a_2 \dots a_m b_1 b_2 \dots b_n$ nad abecedou $\Sigma_1 \cup \Sigma_2$. Zřetězení značíme jako $w_1 \cdot w_2$, ale znak „$\cdot$“ často vynecháváme. Zřetězení můžeme rozšířit na $n$-násobné zřetězení slova, tedy výrazem $x^n$ označujeme prázdné slovo, pokud $n = 0$, jinak $x \cdot x^{n-1}$.

Tedy například $01^3 0^2 1^0$ znamená slovo 011100.

Někdy je třeba mluvit o určité části slova, konkrétně začátek, prostředek, či konec slova, proto definujeme následující pojmy.

**Definice 4**

Slovo $x$ nazýváme *prefixem* slova $z$, pokud pro nějaké slovo $y$ platí $z = xy$.
Slovo $y$ nazýváme *sufixem* slova $z$, pokud pro nějaké slovo $x$ platí $z = xy$.
Slovo $y$ nazýváme *infixem* slova $u$, pokud pro nějaká slova $x, z$ platí $u = xyz$.

**Definice 5**

*Jazyk* nad abecedou $\Sigma$ je libovolná množina $L \subseteq \Sigma^*$. Jazyky obvykle značíme symbolem $L$, případně $L_i$ a podobně.

V našich příkladech bude jazyk často nad binární abecedou ($\Sigma = \{0, 1\}$). To ale není žádné omezení, protože každé slovo nad jinou abecedou se dá pomocí slov nad $\Sigma$ přirozeně zakódovat.

**Definice 6**

*Homomorfismus* $h$ abecedy $\Sigma$ je zobrazení $h(a)$ přiřazující každému symbolu $a \in \Sigma$ řetězec nad zvolenou abecedou $\Pi$. Homomorfismus můžeme rozšířit na slova takto:

$$h(a) = \begin{cases} \epsilon & \text{když } a = \epsilon \\ h(b)h(w) & \text{když } a = bw \end{cases}.$$

Na jazyk poté rozšíříme homomorfismus jako množinu homomorfních obrazů všech jeho slov.

## 2.2 Gramatiky

Protože se v tomto textu budeme zabývat převážně nekonečnými jazyky, vyvstává otázka, jak je konečným způsobem reprezentovat. Ačkoliv neexistuje způsob, jak

konečným způsobem reprezentovat kterýkoliv jazyk, podstatná část jazyků lze reprezentovat pomocí takzvaných *gramatik*.

**Definice 7**

Gramatika $G$ je čtveřice $(N, \Sigma, P, S)$, kde

- $N$ je neprázdná konečná množina. Její prvky nazýváme *neterminály*.
- $\Sigma$ je neprázdná konečná množina. Její prvky nazýváme *terminály*. Přitom platí $N \cap \Sigma = \emptyset$. Množina všech symbolů gramatiky (tedy $N \cup \Sigma$) se obvykle značí $V$.
- $P \subseteq V^*NV^* \times V$ je konečná množina pravidel. Pravidlo $(a, b) \in P$ také zapisujeme jako $a \to b$.
- $S$ je *počáteční neterminál*, tedy $S \in N$.

**Definice 8**

Ke každé gramatice $G$ můžeme přiřadit relaci *odvození v jednom kroku* $\Rightarrow_G \subseteq V^* \times V^*$ tak, že $(\alpha, \beta) \in \Rightarrow_G$ právě když existuje pravidlo $(\gamma, \delta) \in P$ a slova $w_1, w_2 \in V^*$ taková, že $w_1\gamma w_2 = \alpha$ a $w_1\beta w_2 = \beta$.

Pokud je $G$ zřejmé z kontextu, píšeme místo $\Rightarrow_G$ pouze $\Rightarrow$.

**Definice 9**

Relaci *odvození v k krocích* ${}^k\Rightarrow$ definujeme jako

$${}^k\Rightarrow = \begin{cases} \text{Relace identity} & \text{pro } k = 0 \\ \overset{k-1}{\Rightarrow} \circ \Rightarrow & \text{pro } k \in \mathbb{N} \end{cases}$$

Relaci *odvození* potom jako reflexivní a tranzitivní uzávěr relace $\Rightarrow$, nebo také

$$\Rightarrow^* = \bigcup_{i=0}^{\infty} \overset{i}{\Rightarrow}$$

Relace odvození nám nyní dává možnost zjistit všechny takzvané *větné formy*, to znamená všechny řetězce nad $V$, které lze odvodit z počátečního neterminálu $S$. Jazyk *generovaný* gramatikou $G$ potom definujeme jako množinu všech jeho větných forem neobsahujících neterminály a značíme jej $\mathrm{L}(G)$. Přesněji

$$\mathrm{L}(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}.$$

### 2.2.1 Chomského hierarchie

*Chomského hierarchie* rozděluje gramatiky do čtyř základních druhů podle jejich popisné síly. Nás bude zajímat především typ 2, ovšem pro úplnost uvedeme všechny:

**Typ 0** Gramatika z Definice 7 je gramatikou typu 0. Na pravidla se nekladou žádné další podmínky.

**Typ 1** Gramatika typu 1, nebo také kontextová gramatika (CSG) pro všechna pravidla $\alpha \to \beta$ splňuje $|\alpha| \leq |\beta|$ s výjimkou pravidla $S \to \epsilon$, pokud není neterminál $S$ obsažen na pravé straně žádného pravidla. V každém kroku odvození vytvoří novou větnou formu, která je stejně dlouhá, nebo delší. Proto se jí také někdy říká *nezkracující*.

**Typ 2** Gramatika typu 2, nebo také bezkontextová gramatika (CFG) má všechna pravidla ve tvaru $A \to \beta$, kde $|beta| \geq 1$ s výjimkou pravidla $S \to \epsilon$, pokud není neterminál $S$ obsažen na pravé straně žádného pravidla.

**Typ 3** Gramatika typu 3, nebo také regulární gramatika má všechna pravidla ve tvaru $A \to aB$ nebo $A \to a$ opět s případnou výjimkou pravidla $S \to \epsilon$.

Z definice je vidět, že každý další typ je vytvořený dalším omezením na tvar pravidel gramatiky, tedy tyto gramatiky jsou slabší v popisu jazyků.

Dále můžeme zadefinovat příslušné třídy jazyků.

**Definice 10**

*Kontextový (bezkontextový, regulární) jazyk* je jazyk $L$, pro který existuje kontextová (bezkontextová, regulární) gramatika $G$ taková, že platí $\mathrm{L}(G) = L$.

## 2.3 Automaty

Dalším prostředkem, jak konečným způsobem zapsat některé nekonečné jazyky jsou automaty. V dalších kapitolách se nejvíce budeme věnovat *zásobníkovým automatům* (PDA), ale i *lineárně omezeným automatům* (LBA). Tyto automaty si podrobněji představme.

### 2.3.1 Základní automaty

Dále budeme $\mathrm{Fin2}^A$ značit množinu všech konečných podmnožin $A$.

**Definice 11**

Nedeterministický zásobníkový automat je sedmice $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde

- $Q$ je konečná neprázdná množina. Její prvky nazýváme stavy.
- $\Sigma$ je konečná neprázdná množina. Nazýváme ji vstupní abeceda.
- $\Gamma$ je konečná neprázdná množina. Nazýváme ji pásková abeceda.
- $\delta \colon Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathrm{Fin2}^{Q \times \Gamma^*}$ nazýváme *přechodová funkce*.

- $q_0 \in Q$ je počáteční stav.
- $Z_0 \in \Gamma$ je počáteční zásobníkový symbol.
- $F \subseteq Q$ je množina koncových stavů.

Výpočet zásobníkového automatu si můžeme představit tak, že stroj má konečnou paměť, ve které si pamatuje stav z $Q$, dále vstupní slovo, které čte směrem zleva doprava a nevrací se při tom, a potom zásobníkovou strukturu, kde si může během výpočtu ukládat a číst symboly z $\Gamma$. Přechodová funkce $\delta$ obsahuje pravidla, která říkají v každém kroku, co má automat dělat. Například pravidlo $((q_0, a, Z_0), (q_1, Z_1 Z_2 Z_3))$ znamená, že pokud na vstupu čteme znak $a$ a na vrcholu zásobníku je znak $Z_0$, nahradíme jej řetězcem znaků $Z_1 Z_2 Z_3$ a přejdeme do stavu $q_1$.

Abychom mohli definovat jazyk akceptovaný zásobníkovým automatem, potřebujeme podobně jako v případě gramatiky jakési „konfigurace“. U gramatik jimi byly větné formy, které se postupně odvozovaly pomocí relace $\Rightarrow_G$. Zde bude celkový stav výpočtu reprezentovat nezpracovaná část vstupního slova, stav z $Q$ a obsah zásobníku. Relaci přechodu mezi těmito konfiguracemi pak definujeme podle přechodové funkce $\delta$.

**Definice 12**

*Konfigurací* nazveme libovolný prvek $(p, w, \alpha)$ z $Q \times \Sigma^* \times \Gamma^*$.
Relaci *kroku výpočtu* $\vdash$ definujeme jako

$$(p, aw, Z\alpha) \vdash (q, w, \beta\alpha) \Leftrightarrow \exists (q, \beta) \in \delta(p, a, Z) \quad \text{pro } a \in \Sigma \cup \{\epsilon\}.$$

Reflexivní a tranzitivní uzávěr relace kroku výpočtu značíme $\overset{*}{\vdash}$.

Všimněme si, že $a \in \Sigma \cup \{\epsilon\}$ umožňuje provádět takzvané epsilon kroky, kdy automat provádí nějaké operace se zásobníkem, nečte u toho však žádné další znaky ze vstupu. Dále $\alpha$ může být jeden symbol, více symbolů, nebo i žádný symbol, tímto způsobem lze odstraňovat symboly ze zásobníku.

Jazyk $\mathrm{L}(A)$ *rozpoznávaný* zásobníkovým automatem $A$ potom definujeme pomocí relace $\overset{*}{\vdash}$, a to tak, že slovo $w$ automat přijímá, pokud jej celé přečte a přitom se dostane do koncového stavu.

$$\mathrm{L}(A) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \overset{*}{\vdash} (q_f, \epsilon, \alpha)\}, \quad \text{kde } q_f \in F, \alpha \in \Gamma^*.$$

Tímto způsobem přijímáme *koncovým stavem*, lze však přijímat i prázdným zásobníkem. Tento způsob si ukazovat nebudeme, lze však dokázat, že pro každý PDA přijímající koncovým stavem existuje PDA přijímající prázdným zásobníkem (a naopak) tak, že jsou jazykově ekvivalentní. To znamená, že se rovnají jejich přijímané jazyky.

Lze ukázat, že ke každému PDA lze sestrojit jazykově ekvivalentní bezkontextovou gramatiku, to znamená, že zásobníkové automaty rozpoznávají přesně třídu bezkontextových jazyků.

Všimněme si skutečnosti, že v každém kroku je obecně různý počet následujících konfigurací. Výpočetní větev tedy není jednoznačně určena. Z definice ale vidíme, že automat přijímá, pokud *existuje* nějaká výpočetní větev vedoucí do koncového stavu. Tedy pokud jiná větev do koncového stavu nevede, neznamená to automaticky, že automat daný vstup zamítá. Tím se však liší *deterministické* zásobníkové automaty (DPDA), které můžeme neformálně vnímat jako speciální případ PDA, kdy je obraz každé trojice v přechodové funkci množina velikosti přesně jedna. Tím je každý výpočet jednoznačný (deterministický).

Formálně DPDA definujeme stejně jako PDA, ale $\delta$ bude ve tvaru

$$\delta\colon Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to Q \times \Gamma^*,$$

navíc zakážeme nedeterministickou volbu mezi $\epsilon$-přechodem a přechodem pod symbolem. Tedy:

$$\forall q \in Q\ \forall a \in \Sigma\ \forall c \in \Gamma\colon \delta(q, \epsilon, c) \text{ je definovaná} \Rightarrow \delta(q, a, c) \text{ není definovaná.}$$

Odpovídající třída jazyků se nazývá *deterministické bezkontextové jazyky* (značíme DCFL) a je podmnožinou bezkontextových jazyků. Existují i gramatiky, které generují tuto třídu, tu však v dalším textu nebudeme potřebovat.

Lineárně omezené automaty jsou výrazně silnější. Sice na rozdíl od zásobníkových automatů nemají k dispozici zásobník, mohou ale vstupní slovo přepisovat a k takto zapsaným údajům se vracet. Oproti nejsilnějšímu modelu (*Turingovým strojům*) však mají k dispozici na zápis pouze místo o velikosti samotného vstupu, případně jeho $k$-násobku.

**Definice 13**

Lineárně omezeným automatem je šestice $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde

- $Q$ je neprázdná konečná množina stavů
- $F \subseteq Q$,
- $\Gamma$ je abeceda páskových symbolů
- $\Sigma \subseteq \Gamma$ je konečná abeceda vstupních symbolů
- $\triangleright, \triangleleft \in \Gamma_k$ značí konce pásky kolem vstupního slova a
- $\delta\colon Q \times \Gamma \to 2^{Q \times \Gamma \times \{-1,1\}}$ je přechodová funkce.

Navíc vyžadujeme aby pro každé $q, p \in Q$ platilo:

$$\forall (p, c, d) \in \delta(q, \triangleright)\colon\ d = 1 \tag{1}$$

$$\forall (p, c, d) \in \delta(q, \triangleleft)\colon\ d = -1 \tag{2}$$

Jelikož jsou LBA restrikcí Turingova stroje, je výpočet LBA definován stejně. Automat přechází mezi jednotlivými konfiguracemi. Formálně je každá konfigurace $(q, s, n)$ prvkem množiny $Q \times \Gamma^* \times \mathbb{N}_0$, kde $q$ je aktuální stav, $s$ je obsah pásky a $n$ je pořadí snímaného políčka zleva. *Počáteční* konfigurací automatu $\mathcal{A}$ je $(q_0, \triangleright w \triangleleft, 1)$ a *přijímající* konfigurací je každá konfigurace $(q_{acc}, \triangleright w \triangleleft, n)$, *zamítající* je konfigurace ve tvaru $(q_{rej}, \triangleright w \triangleleft, n)$.

Výpočet limitovaného automatu $\mathcal{A}$ sestává z přechodů mezi konfiguracemi. Definujeme proto následující relaci přechodu $\vdash_{\mathcal{A}} \subseteq (Q \times \Gamma^* \times \mathbb{N})^2$:

$$(q, \triangleright c_1 c_2 \dots c_i \dots c_n \triangleleft, i) \vdash_{\mathcal{A}} (p, \triangleright c_1 c_2 \dots c \dots c_n \triangleleft, i + d) \quad \Leftrightarrow \quad (p, c, d) \in \delta(q, c_i),$$

přitom $q \neq q_{\text{rej}}$.

Pokud je z kontextu zřejmý automat $\mathcal{A}$, píšeme pouze $\vdash$. Jestliže je $\vdash$ pro konfiguraci $k$ nedefinovaný ($k \nvdash$), říkáme, že $\mathcal{A}$ v $k$ zastavuje.

Opět můžeme definovat reflexivní a tranzitivní uzávěr relace $\vdash$. Automat poté přijímá slovo $w$, jestliže existuje přijímající konfigurace $(q_{\text{acc}}, w_2, j)$ splňující $(q_0, w, 1) \overset{*}{\vdash} (q_{\text{acc}}, w_2, j)$.

Jazyk přijímaný automatem $\mathcal{A}$ je množina všech přijímaných slov a značíme ji $\mathrm{L}(\mathcal{A})$.

### 2.3.2 Limitované automaty

Limitované automaty poprvé představil T. Hibbard [1] pod názvem „scan-limited automata“. Hibbard limitované automaty původně definoval pomocí přepisovacích systémů obsahující pravidla následujících typů:

$$pX \to qY, \qquad pX \to Yq, \qquad Xp \to Yq, \qquad Xp \to qY, \qquad p \to q.$$

Ekvivalentní definici vytvořili Pighizzini a Pisoni v práci [2].

**Definice 14**

$k$-limitovaným automatem (značíme $k$-LA) pro dané $k \in \mathbb{N}_0$ je šestice $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde

- $Q$ je konečná množina stavů,
- $F \subseteq Q$,
- $\Gamma$ je konečná abeceda páskových symbolů taková, že $\Gamma = \Gamma_0 \cup \Gamma_1 \cup \dots \cup \Gamma_k$, $\forall i, j, 0 \leq i \leq k, 0 \leq j \leq k\colon \quad \Gamma_i \cap \Gamma_j \neq \emptyset \;\Rightarrow\; i = j$,
- $\Sigma = \Gamma_0$ je konečná abeceda vstupních symbolů,
- $\triangleright, \triangleleft \in \Gamma_k$ značí konce pásky kolem vstupního slova a
- $\delta\colon Q \times \Gamma \to 2^{Q \times \Gamma \times \{-1,1\}}$ je přechodová funkce.

Navíc pro všechny $(q, \gamma, m) \in \delta(p, \sigma)$, kde $p, q \in Q$, $\sigma \in \Gamma_h$, $\gamma \in \Gamma_{h'}$ a $m \in \{-1, 1\}$, vyžadujeme:

- $h = k \quad \Rightarrow \quad \sigma = \gamma \wedge h = h'$, (3)
- $h < k \wedge m = 1 \quad \Rightarrow \quad h' = \min\left(\lceil \frac{h}{2} \rceil \cdot 2 + 1, k\right)$ a (4)
- $h < k \wedge m = -1 \quad \Rightarrow \quad h' = \min\left(\lceil \frac{h+1}{2} \rceil \cdot 2, k\right)$ (5)

a aby pro každé $q, p \in Q$ platilo:

$$\forall (p, c, d) \in \delta(q, \triangleright): \ \ d = 1 \tag{6}$$

$$\forall (p, c, d) \in \delta(q, \triangleleft): \ \ d = -1 \tag{7}$$

**Definice 15**
Každý $k$-limitovaný automat pro libovolné $k \in \mathbb{N}_0$ nazveme limitovaný automat.

Zde vidíme oproti LBA výrazné změny zejména v páskové abecedě a v přechodové funkci. Páskové abeceda $\Gamma$ je rozdělená na abecedy $\Gamma_0, \Gamma_1, \ldots \Gamma_k$, které tak tvoří disjunktní pokrytí $\Gamma$. Jednotlivé abecedy $\Gamma_i$ mají význam množiny symbolů, které lze na políčko napsat při jeho $i$-tém přepisu.

Jakmile tedy hlava projde přes políčko, ve kterém je symbol z abecedy $\Gamma_i, i < k$, přepíše se obsah políčka na symbol z abecedy $\Gamma_{i+1}$, případně $\Gamma_{i+2}$, pokud hlava na daném políčku změnila svůj směr a zároveň $i < k - 1$. Při změně směru hlavy totiž počítáme průchod políčka za dva. To je způsobeno chováním, které vyplývá z definice limitovaných automatů pomocí přepisovacích systémů. Hlava přepisovacího systému se totiž nachází vždy mezi jednotlivými znaky vstupu a změna směru tak implikuje opětovný průchod posledně přepsaným znakem.

Tato definice také zaručuje, že obsah všech buněk napravo od pozice hlavy bude vždy z množiny $\Gamma_x$, kde $x$ je sudé (je-li $x < k$) a obsah buněk nalevo od hlavy bude z množiny $\Gamma_x$, kde x je liché (je-li $x < k$).

Jak je vidět v úvodu této podkapitoly, hlava těchto systémů se nenachází na pozici symbolů, ale mezi nimi. Pokud se tedy změní směr pohybu hlavy, ve skutečnosti to znamená, že hlava přešla pole na druhou stranu a zpět.

Tato omezení jsou zaručena prostřednictvím přechodové funkce $\delta$.

- Podmínka (3) říká, že pokud je symbol z $\Gamma_k$, nelze již přepsat, proto také $h = h'$.
- Podmínka (4) říká, že pokud se hlava přesunula směrem vpravo, bude nový symbol zapsaný na pásku z $\Gamma_{h'}$, kde $h'$ je nejmenší větší liché číslo. Případně $k$, pokud je menší.

- Podmínka (5) říká, že pokud se hlava přesunula směrem vlevo, bude nový symbol zapsaný na pásku z $\Gamma_{h'}$, kde $h'$ je nejmenší větší sudé číslo. Případně $k$, pokud je menší.

Z toho také vidíme, že pokud je nějaký symbol z $\Gamma_i$, kde $i < k$ a hlava automatu je nalevo od tohoto symbolu, $i$ musí být sudé, pokud je napravo $i$, musí být liché.

Jednou z podmínek je také $\triangleright, \triangleleft \in \Gamma_k$, to znamená, že znaky na hranicích vstupu nelze přepsat a díky (6) a (7) je tak nelze hlavou překročit.

Výpočet stroje definujeme velice podobně jako u lineárně omezených automatů. Konfigurace je opět $(q, s, n) \in Q \times \Gamma^* \times \mathbb{N}_0$, kde $q$ je aktuální stav, $s$ je obsah pásky a $n$ je pořadí snímaného políčka zleva. *Počáteční* konfigurací automatu $\mathcal{A}$ je $(q_0, \triangleright w \triangleleft, 1)$ a *koncovou* konfigurací je každá konfigurace $(q_f, \triangleright w \triangleleft, n)$ splňující $q_f \in F$. Zde hovoříme o koncové konfiguraci, nikoliv o akceptující konfiguraci, význam je však stejný.

Výpočet limitovaného automatu $\mathcal{A}$ sestává z přechodů mezi konfiguracemi. Definujeme proto následující relaci přechodu $\vdash_{\mathcal{A}} \subseteq (Q \times \Gamma^* \times \mathbb{N})^2$:

$$(q, \triangleright c_1 c_2 \dots c_i \dots c_n \triangleleft, i) \vdash_{\mathcal{A}} (p, \triangleright c_1 c_2 \dots c \dots c_n \triangleleft, i + d) \quad \Leftrightarrow \quad (p, c, d) \in \delta(q, c_i)$$

Jestliže je $\vdash$ pro konfiguraci $k$ nedefinovaný ($k \nvdash$), říkáme, že $\mathcal{A}$ v $k$ zastavuje.

Výpočtem limitovaného automatu rozumíme (ne)konečnou posloupnost konfigurací $(c_i)_{i=1}^n$, kde pro všechny $i$ platí $1 \leq i < n \colon c_i \vdash c_{i+1}$. Slovo $w$ je přijímané limitovaným automatem (LA) $\mathcal{A}$, jestliže $\mathcal{A}$ v nějakém výpočtu $(c_i)_{i=1}^n$ zastaví v akceptujícím stavu (neboli jestliže $n \in \mathbb{N}$ a $c_n$ je koncová konfigurace). Jazyk přijímaný automatem $\mathcal{A}$ je množina všech přijímaných slov a značíme ji $\mathrm{L}(\mathcal{A})$.

Poznámka 16

Některé zdroje definují akceptování stroje stejně jako v Definici 14, v práci [3] se ale tyto definice liší. Zde automat může překročit hranici danou dvěma zarážkami ($\triangleleft$, $\triangleright$), a to v případě, že zároveň přechází do akceptujícího stavu, tím také vstupní slovo přijímá. Jinými slovy není vyžadováno $c = \triangleleft$ v podmínce (6) a $c = \triangleright$ v (7).

Obě definice jsou však ekvivalentní. Akceptování pomocí koncového stavu můžeme v automatu přijímajícím kombinací stavu a porušením hranic vstupu simulovat tak, že pro každý koncový stav $q_f \in F$ přidáme přechody

$$(q_f, c', 1) \in \delta(q_f, c), \quad \text{pro všechny } c \in \Gamma,$$

kde $c'$ je z odpovídající množiny $\Gamma_x$. Akceptování pomocí porušení hranice vstupu lze simulovat strojem podle Definice 14 přidáním nového koncového stavu, do kterého se automat dostane vždy, když v původním automatu porušil hranici a byl přitom v nějakém koncovém stavu. Přitom zde samozřejmě tuto hranici porušit nesmí, hlava se tedy otočí.

**Definice 17**

Limitovaný automat $\mathcal{A}$ je deterministický, jestliže platí

$$\forall q \in Q \; \forall c \in \Gamma : |\delta(q, c)| \leq 1.$$

Dále budeme u deterministických LA místo $\delta(q, c) = \{(q', c', d)\}$ psát pouze $\delta(q, c) = (q', c', d)$, jelikož množina bude vždy jednoprvková.

Příklad 18

Abychom si vysvětlili přepisování symbolů na příkladu, vytvoříme triviální automat, který nebude dělat nic užitečného, jen bude s hlavou chodit střídavě po prvních dvou políčkách. Nechť $\mathcal{A} = (\{q_0, q_1\}, \{a\}, \{a, a_1, a_2\}, \delta, q_0, \{\}), \Gamma_1 = \{a_1\}$, $\Gamma_2 = \{a_2\}$, $\Gamma_3 = \{a_3, \triangleleft, \triangleright\}$ je 2-limitovaný automat, kde

Vyžadováno Definicí 14, vztahy (6) a (7).
$$\begin{cases} \delta(q, \triangleright) = (q, \triangleright, 1) & q \in Q \\ \delta(q, \triangleleft) = (q, \triangleleft, -1) & q \in Q \end{cases}$$

Pravidla pro symboly z $\Sigma$
$$\begin{cases} \delta(q_0, a) = (q_1, a_1, 1) \\ \delta(q_1, a) = (q_0, a_2, -1) \end{cases}$$

Pravidla pro symboly z $\Gamma_1$
$$\begin{cases} \delta(q_0, a_1) = (q_1, a_3, 1) \\ \delta(q_1, a_1) = (q_0, a_2, -1) \end{cases}$$

Pravidla pro symboly z $\Gamma_2$
$$\begin{cases} \delta(q_0, a_2) = (q_1, a_3, 1) \\ \delta(q_1, a_2) = (q_0, a_3, -1) \end{cases}$$

Pravidla pro symboly z $\Gamma_3$
$$\begin{cases} \delta(q_0, a_3) = (q_1, a_3, 1) \\ \delta(q_1, a_3) = (q_0, a_3, -1) \end{cases}$$

Všimněme si, že první dvě pravidla spíše zapisujeme jako parametrizovanou šablonu, než konkrétní pravidla. Například to první si můžeme vzhledem k $q \in Q$ představit jako dvě pravidla $\delta(q_0, \triangleright) = (q_0, \triangleright, 1)$ a $\delta(q_1, \triangleright) = (q_1, \triangleright, 1)$. Těchto zápisů budeme dále využívat.

Dále si všimněme, že první dvě šablony pravidel jsou vyžadovány definicí limitovaného automatu. Máme sice volnost v určení stavu, do kterého hlava ze zarážky vstoupí, nesmíme ji však překročit. V tomto konkrétní případě zvolíme zanechání stejného stavu.

Nyní můžeme simulovat $\mathcal{A}$ na slově $aa$. Víme, že automat začíná ve stavu $q_0$ na prvním políčku pásky. Čteme symbol $a$, který je z $\Sigma$, tedy víme, že symbol

ještě nikdy nebyl přepsán. Podle odpovídajícího přepíšeme $a$ na $a_1$ a přesuneme se vpravo do stavu $q_1$. Zapsaný symbol je z množiny $\Gamma_{h'}$, kde

$$h' = \min\left(\left\lceil\frac{0}{2}\right\rceil \cdot 2 + 1, 3\right) = 1.$$

V dalším kroku vybereme podle aktuální konfigurace pravidlo, dle kterého jde hlava vlevo do stavu $q_0$ a přitom přepisuje symbol $a$ na $a_2$. Zapsaný symbol je z množiny $\Gamma_{h'}$, kde

$$h' = \min\left(\left\lceil\frac{0+1}{2}\right\rceil \cdot 2, 3\right) = 2.$$

V dalším kroku přepisujeme symbol $a_1$ na $a_3$ a

$$h' = \min\left(\left\lceil\frac{1}{2}\right\rceil \cdot 2 + 1, 3\right) = 3.$$

V dalších krocích bude hlava dál oscilovat na prvních dvou políčkách. Jejich obsah ale už zůstane $a_3$, protože pro další $h$ bude operace minima vždy vybírat číslo $k = 3$. Část pásky, kterou již nelze přepisovat, budeme nazývat *zamrzlá*.

Předchozí simulace odpovídá naší intuitivní představě o dvojitém průchodu políčka při změně směru hlavy. Vidíme zde také, že $\delta$ je určená správně a všechna $h'$ odpovídají. Pokud bychom simulovali stejným automatem slovo $w = a$, bude se hlava otáčet až na zarážkách a symbol $a$ se bude přepisovat vždy pouze na $\Gamma$ s následujícím indexem.

# 3 Výpočetní síla limitovaných automatů

0-LA automaty mají tu vlastnost, že hlava se při výpočtu může hýbat libovolně po pásce (v rozsahu vstupu), nesmí ale žádnou buňku přepsat. Fakticky tak dostáváme dvoucestný nedeterministický/deterministický automat (2-NFA/2-DFA). O těchto automatech bylo dokázáno, že existuje vzájemná simulace s konečnými automaty. Nula limitované automaty (deterministická i nedeterministická verze) tedy rozpoznávají právě regulární jazyky.

Zajímavé je, že přidáním možnosti jednoho přepisu se síla těchto automatů nijak nemění. 1-LA automaty jsou sice obecnější, stále však existuje simulace 1-LA pomocí konečného automatu. Tímto převodem se zabývá práce [2].

Jak dokázal Hibbard již ve své původní práci o LA [1], třída jazyků přijímaných LA je totožná s třídou bezkontextových jazyků. Hibbard zde uvádí důkaz pro LA definované pomocí přepisovacích systémů. Poté G. Pighizzini v práci [3] představil důkaz pro LA definované pomocí Turingových strojů (tedy podle Definice 14). Důkaz spočívá ve vzájemné simulaci 2-LA a PDA. Protože PDA rozpoznávají právě třídu bezkontextových jazyků, je zřejmé, že i 2-LA rozpoznávají tuto třídu.

Přidáváním počtu přepisů se ale dále výpočetní síla nedeterministických limitovaných automatů nezvětší. To lze dokázat přímou simulací mezi $x$-LA, kde $x \geq 2$ a PDA. Tato simulace byla představena v [4].

Deterministické (dva a více) limitované automaty potom tvoří nekonečnou hierarchii tříd jazyků mezi deterministickými bezkontextovými a bezkontextovými jazyky. Tyto jazyky nazýváme $k$-deterministické ($k$-det) dle příslušného parametru $k$. Pomocí jednoduché konstrukce [1] umíme pro $x \geq 3$ vytvořit jazyk přijímaný $x$-LA, který není přijímaný $(x-1)$-LA.

Žádný deterministický LA však nerozpoznává všechny bezkontextové jazyky. Uvažme následující příklad.

Příklad 19

Mějme jazyk $L = \{a^i b^i \mid i \in \mathbb{N}_0\}$. Tento jazyk lze jednoduše rozpoznat 2-limitovaným automatem $\mathcal{A}$, který nalezne první znak „b“ a poté střídavě označuje odpovídající znaky „a“ směrem doleva a nové znaky „b“ směrem doprava. Formálně definujme $\mathcal{A}$ takto: $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde $Q = \{q_0, f_a, f_b, q_{\text{check}}, q_{\text{rej}}\} \cup F$, $\Sigma = \{a, b\}$, $\Gamma_1 = \{\dot{a}, \dot{b}\}$, $\Gamma_2 = \{\ddot{a}, \ddot{b}\}$, $F = \{q_{\text{acc}}\}$ a $\delta$ je definována následovně:

$$\text{Hledání prvního „b“} \quad \begin{cases} \delta(q_0, a) = (q_0, \dot{a}, 1) \\ \delta(q_0, b) = (f_a, \ddot{b}, -1) \\ \delta(q_0, \triangleleft) = (q_{\text{check}}, \triangleleft, -1) \end{cases}$$

$$\begin{matrix} \text{Hledání} \\ \text{odpovídajícího „a“} \\ \text{k nalezenému „b“} \end{matrix} \quad \begin{cases} \delta(f_a, x) = (f_a, x, -1) & x \in \{\ddot{a}, \ddot{b}\} \\ \delta(f_a, \dot{a}) = (f_b, \ddot{a}, 1) \\ \delta(f_a, \triangleright) = (f_{\text{rej}}, \triangleright, 1) \end{cases}$$

$$\begin{matrix} \text{Hledání následujícího} \\ \text{„b“} \end{matrix} \quad \begin{cases} \delta(f_b, x) = (f_b, x, 1) & x \in \{\ddot{a}, \ddot{b}\} \\ \delta(f_b, b) = (f_a, \ddot{b}, -1) \\ \delta(f_b, \triangleleft) = (q_{\text{check}}, \triangleleft, -1) \end{cases}$$

$$\begin{matrix} \text{Kontrola nepárových} \\ \text{„a“} \end{matrix} \quad \begin{cases} \delta(q_{\text{check}}, x) = (q_{\text{check}}, x, -1) & x \in \{\ddot{a}, \ddot{b}\} \\ \delta(q_{\text{check}}, a) = (q_{\text{rej}}, \dot{a}, -1) \\ \delta(q_{\text{check}}, \triangleright) = (q_{\text{acc}}, \triangleright, 1) \end{cases}$$

Zde je vidět jeden z postupů výpočtu typických pro LA. Automat nalezne „střed“ rozpoznávaného vzoru (v Příkladu 19 je to první znak b) a poté pracuje střídavě na obou stranách. Využíváme tak toho, že hlava prochází přes buňky, kde už jsme užitečné informace zapsali.

Pokud by ale tento střed nebyl nijak vyznačen, deterministická verze ho obecně naleznout nedokáže, protože by potom počet páskových symbolů nebo stavů byl závislý na délce vstupu a u nekonečných jazyků tedy nekonečný. Příkladem takových jazyků (kde automat při průchodu nerozpozná střed tohoto vzoru, který je potřeba ověřit) jsou

$$L_1 = \{w \mid w = w^{\text{R}}\}, \qquad \text{(R zde značí reverz slova)}$$
$$L_2 = \{ww' \mid w, w' \in \{a, b\}^*, \, w \text{ neobsahuje } b \text{ a } w' \text{ obsahuje } b\}.$$

Tím je také dokázáno, že tato nekonečná hierarchie tzv. $k$-deterministických jazyků je vlastní podtřídou CFL. Ukážeme také, že 2-LA rozpoznávají přesně třídu deterministických bezkontextových jazyků.

## 3.1 Konverze CFG na 2-LA

Každý bezkontextový jazyk lze zapsat jako homomorfismus průniku Dyckova jazyku nad $k$ páry symbolů s regulárním jazykem. Dyckovým jazykem $D_k$ zde myslíme jazyk správných uzávorkování symbolů $(_i$ a $)_i$ pro všechny $i \in \{1, \ldots, k\}$. Toto tvrzení formuluje tzv. Chomsky–Schützenbergerova věta, dále jej pak zesiluje věta 3 z materiálu [5].

**Věta 20**
*Jazyk $L \subseteq \Sigma^*$ je bezkontextový právě tehdy, když existují $k, l \geq 1$, regulární jazyk $R \subseteq \Omega_{j,k}^*$ a homomorfizmus $h \colon \Omega_{j,k} \to \Sigma$, splňující $L = h(D_{j,k} \cap R)$.*

$D_{j,k}$ zde značí upravený Dyckův jazyk definovaný nad abecedou

$$\Omega_{\{1,\ldots,j\},\{1,\ldots,k\}} = \{\ (_x \mid x \in \{1, \ldots, j\}\} \cup \{\ )_x \ \mid x \in \{1, \ldots, j\}\} \cup \{1, \ldots, k\},$$

gramatikou

$$\begin{aligned} S &\to (_x S)_x \qquad && x \in \{1, \ldots, j\} \\ S &\to SS \\ S &\to c \qquad && c \in \{1, \ldots, k\} \\ S &\to \epsilon \end{aligned}$$

Každý bezkontextový jazyk $L$ pak můžeme rozpoznat složením tří komponent:

- Nedeterministický stroj $T$, který pro vstup $x$ zapíše na pásku $h^{-1}(x)$.
- Stroj $M_{D_{j,k}}$, který rozpoznává $D_{j,k}$.
- Automat $A$ rozpoznávající regulární jazyk $R$.

Protože inverzní homomorfismus k homomorfismu $h$ musí mít také obrazy délky jedna, půjde jednoduše vytvořit 2-limitovaný automat simulující $T$. Pro automat $A$ je situace obdobná. Zbývá vyřešit, jak sestrojit 2-LA $M_{D_{j,k}}$ rozpoznávající $D_{j,k}$.

Můžeme zvolit strategii, kdy opakovaně vyhledáváme první neoznačenou uzavírací závorku zleva $)_a$, přitom všechny nepárové symboly $c$ označíme $\hat{c}$ a při dalších průchodech je budeme ignorovat. Poté obrátíme směr hlavy doleva (tím závorku označíme např. $\hat{)}_a$ ) a hledáme jakoukoliv neoznačenou závorku. Protože všechny uzavírací závorky směrem od aktuální pozice již musely být označeny, musí být takto nalezený symbol buď otevírací závorka $(_b$, nebo začátek pásky.

Pokud nalezneme začátek pásky, není výraz správně uzávorkovaný, pokud $(_b$, musí platit $a = b$, jinak neodpovídá typ závorek. Potom můžeme celý proces opakovat, dokud nenalezneme konec pásky.

Poté je třeba zkontrolovat, jestli nezbyly žádné neoznačené otevírací závorky. Všechny uzavírací závorky a neutrální symboly jsou označeny hned při prvním průchodu daným políčkem, proto nám na těchto místech stačí dva přepisy. Přes každou otevírací závorku hlava projde jednou při hledání některé uzavírací závorky a až poté je označena. Nikdy však neprocházíme doleva přes závorku, kterou neoznačíme, a pro kterou již není označena uzavírací závorka. Proto nám na všech políčkách pásky jistě stačí 2 přepisy, a proto lze jazyk rozpoznat pomocí 2-LA.

Zdá se tedy, že všechny tři stroje $T$, $A$ i $M_{D_{j,k}}$ by šlo zkombinovat do jednoho 2-LA. Nejde však nejprve vygenerovat řetězec $x_2 = h^{-1}(x)$ a poté zkontrolovat, jestli $x_2 \in D_{i,j}$ a $x_2 \in R$, protože bychom průchodem pásky vypotřebovali jeden průchod pro každé políčko. Využít však lze Věty 20. Víme, že homomorfismus $h$ zobrazuje řetězce délky jedna na řetězce délky jedna. Mohli bychom tedy prostě spustit $M_{D_{j,k}}$ přímo na vstup $x$, přitom však za běhu simulovat $T$ i $A$. Automat $A$ lze jednoduše simulovat pomocí stavu našeho stroje a inverzní homomorfismus bychom nedeterministicky tvořili vždy při prvním průchodu daným políčkem. Obsah políčka by potom byl sice nedeterministicky vytvořený, ale zůstal by stejný po dobu celého výpočtu a přitom nepotřebujeme vyšší počet přepisů políček.

Tato konverze je jednoduchá, nezachovává však determinismus. Existuje ale i konverze, z níž plyne, že pro každý deterministický bezkontextový jazyk existuje deterministický 2-limitovaný automat, který jej rozpoznává.

Celý tento postup je detailně popsán v [6].

## 3.2 Simulace 2-LA pomocí PDA

Ukažme si nyní simulaci z [3], převádějící libovolný 2-LA na (nedeterministický) zásobníkový automat.

Před samotným popisem algoritmu si zaveďme některá označení a pojmy.

*Přechodová tabulka* je libovolná relace $T \subseteq Q \times \{-1, 1\} \times Q \times \{-1, 1\}$. Přechodovou tabulku asociovanou se slovem $w \in \Gamma_2^*$ definujme jako $\tau_w$, kde $(q, d', p, d'') \in \tau_w$, jestliže v $M$ existuje výpočetní cesta tak, aby hlava $M$ vstupovala na segment pásky obsahující slovo $w$ ve stavu $q$ ze strany $d'$ a vystoupila z něj ve stavu $p$ na stranu $d''$. Obecnou přechodovou tabulku budeme dále značit $T$.

*Složení přechodových tabulek* je operace $\cdot$ mezi dvěma tabulkami $\tau_w$ a $\tau_z$, má však smysl pouze pokud uvažujeme, že $\tau_w$ a $\tau_z$ popisují sousední segmenty na pásce se slovy $w, z \in \Gamma_2^*$. Výslednou tabulku $\tau_{wz} = \tau_w \cdot \tau_z$ lze potom obdržet jednoduše z tabulek tak, aby byl zachován význam čtveřic vysvětlený v předchozím bodu.

Pokud reprezentujeme tabulky pomocí binárních matic, můžeme pro libovolné dvě tabulky $T_1$ a $T_2$ definovat složení jako násobení odpovídajících matic a značíme jej $T_1 \cdot T_2$

*Funkce exit* popisuje případ, kdy máme opět dva sousední segmenty na pásce $s_l$ a $s_r$, které již obsahují pouze symboly z $\Gamma_2$, tentokrát nás však zajímá případ, kdy se hlava nachází mezi oběma segmenty. Nejprve uvažme, že hlava vstupuje do levého segmentu zprava, zajímá nás množina dvojic $(p, d)$ z $Q \times \{-1, 1\}$ tak, že hlava opustí celý segment $s_l s_r$ ve stavu $p$ směrem $d$. Tuto množinu je možné obdržet pouze z přechodových tabulek $T_l$ a $T_r$ pro oba segmenty. Budeme ji tedy značit $\text{exit}(T_l, q, -1, T_r)$. Analogicky můžeme značit množinu pro případ, že hlava vstupuje zleva do pravého segmentu $s_r$ – $\text{exit}(T_l, q, 1, T_r)$

*funkce nSelect(S)* vrátí nedeterministický výběr jednoho prvku z $S$, přitom pokud je $S$ prázdná, automat zamítá.

*funkce read()* značí přečtení jednoho znaku z pásky a posunutí čtecí hlavy doprava (zásobníkový automat nemůže posunovat hlavou opačným směrem).

Pomocí těchto funkcí lze sestrojit algoritmus popisující chování PDA $M'$ simulujícího chování libovolného 2-limitovaného automatu $M$ s přechodovou funkcí $\delta$. Předpokládejme také, že zásobník automatu je prázdný a hlava je na pozici 1. Pro jednoduchost také nejprve předpokládejme, že vstup obsahuje ukončovací symboly $\triangleright$ a $\triangleleft$, později tento předpoklad odstraníme.

```
1  push(τ▷)
2  q ← q0
3  repeat
4     a ←read()
5     (q, X, d) ←nSelect(δ(q, a))
6     if d = 1
7        push(X)
8     else
9        T ← τX
10       repeat
11          Y ←pop()
12          if Y je přechodová tabulka
13             (q, d) ←nSelect(exit(Y, q, −1, T))
14             T ← Y · T
15          else
16             (q, Z, d) ←nSelect(δ(q, Y))
17             if d = 1
18                (q, d) ←nSelect(exit(τZ, q, 1, T))
19             T ← τZ · T
20       until d = 1
21       push(T)
22 until a = ◁
23 ACCEPT
```

Algoritmus pracuje ve dvou módech – normální a zpětný mód. V normálním módu pracuje, pokud čte automat ještě nenavštívený symbol (řádek 7), ve zpětném módu (řádek 9 – 21) pracuje v ostatních případech, protože se ale hlava PDA nemůže vracet zpět na již přečtené symboly, musí pracovat pouze s informacemi uloženými na zásobníku.

Na zásobník jsou v normálním módu ukládány symboly $Y_m, \dots Y_1$, kde $m \geq 0$ a každý $Y_j$ je buď symbol z $\Gamma_1$, nebo přechodová tabulka.

V jedné iteraci hlavního cyklu se nejprve načte jeden symbol (tedy každá iterace začíná normálním módem) a nedeterministicky se vybere přechod podle delta funkce stroje $M$. Poté můžeme podle směru, ve kterém se pohne hlava, poznat, jestli automat zůstane v normálním módu. V normálním módu totiž musí platit, že páska napravo od aktuálního políčka pásky ještě nebyla navštívena a políčka nalevo už byla alespoň jednou navštívena.

V normálním módu se tedy pouze uloží na zásobník jeden symbol z $\Gamma_1$, nebo $\Gamma_2$, záleží na směru $d$.

Ve zpětném módu se pracuje s aktuální přechodovou tabulkou $T$ popisující chování na již „zamrzlé“ části pásky od polohy, na které je aktuálně hlava simulovaného stroje $M$ až po polohu přečteného symbolu $a$. Tuto tabulku můžeme rovnou inicializovat přechodovou tabulkou pro symbol $X$, jelikož na tomto symbolu se směr hlavy $M$ otočil, takže už nepůjde přepsat.

Následuje vnitřní cyklus, který končí až v momentě, kdy pomocí přechodové tabulky zjistíme směr výstupu ze segmentu pásky $d = 1$. V tomto případě se totiž hlava $M$ dostane opět na další ještě nepřečtený symbol a $M'$ tedy vstupuje do normálního módu. V cyklu vždy automat vyjme ze zásobníku jeden symbol $Y$. Pokud je tento symbol přechodová tabulka, znamená to, že máme tabulku $Y$ pro segment pásky přímo sousedící se segmentem popsaným pomocí tabulky $T$ a hlava se nachází mezi nimi, přičemž postupuje doleva.Nedeterministicky tedy vybereme výslednou „větev“ výpočtu pomocí množiny $\texttt{exit}(Y, q, 1, T)$ a obě přechodové tabulky můžeme spojit, protože hlava se teď nachází vně celého segmentu.

V případě, že symbol $Y$ vyňatý ze zásobníku byl symbol z $\Gamma_1$, je třeba nedeterministicky zjistit nový stav, směr hlavy a nově přepsaný symbol $Z$. Pokud se ale směr hlavy otočí opět doprava, znamená to, že se nachází mezi dvěma zamrzlými segmenty a musíme opět vybrat stav a směr pomocí `exit`. Následně aktualizujeme tabulku $T$.

Pokud je po iteraci hlavního cyklu v $a$ načtená pravá zarážka, znamená to, že automat přes ni prošel a proto přijímá. Připomeňme, že v poznámce 16 jsme diskutovali vzájemnou ekvivalenci tohoto způsobu přijímání se způsobem s koncovými stavy.

Zbývá nám odstranit předpoklad s koncovými značkami kolem vstupního slova automatu $M'$. Uvědomme si, že levá zarážka ve vstupu vůbec nemusí být, protože automat začíná číst až na políčku s indexem 1. Pro fungování zpětného módu je ovšem nutné zachovat přechodovou tabulku $\tau_{\triangleright}$. Pravou zarážku už však beze změny kódu odstranit nelze, před porušením pravé zarážky totiž může automat dál procházet celou pásku, po jejím odstranění by však automat nepoznal,

že vstup končí. Můžeme ale nahradit řádek 4 za nedeterministickou volbu mezi přečtením dalšího symbolu a přiřazením $a \leftarrow \triangleleft$. V prvním případě kód probíhá dál s rozdílem, že pokud už automat přečetl celý vstup, zamítá.

Tento případ lze simulovat i pomocí obecnější metody převádějící $k$-LA i pro $k > 2$, tato obecnější simulace však nezachovává, a ani nemůže zachovávat, determinismus. Deterministické $k$-limitované automaty jsou slabší než zásobníkové automaty, proto je předchozí simulace důležitá, deterministický případ zde simulujeme prostým nahrazením nedeterministického výběru `nSelect()` deterministickým, případně zamítnutím, pokud žádný přechod neexistuje.

## 3.3 Simulace k-LA pomocí PDA

Důkaz o ekvivalenci $k$-limitovaných automatů se zásobníkovými automaty byl představen už Hibbardem a to induktivně pomocí ekvivalence $k$-limitovaného automatu s $k-1$-limitovaným a následně ekvivalence 2-limitovaného automatu a PDA.

V materiálu [4] byl představen algoritmus pro přímou simulaci. Využívá princip takzvané crossing posloupnosti. Tu můžeme přiřadit ke každé hranici mezi dvěma sousedními buňkami a je definována posloupností všech stavů, do kterých automat přejde při překročení této hranice. Pro dvě crossing posloupnosti můžeme zjistit, jestli je můžeme přiřadit oběma hranicím buňky s obsahem $a \in \Sigma$ v některém akceptujícím výpočtu. Bohužel simulace $k$-LA nejde vyřešit pouhým nedeterministickým hádáním crossing posloupností a kontrolou jejich kompatibility na hranicích každé buňky, protože crossing posloupnosti v těchto strojích obecně nemusí být konstantně dlouhé, mohou mít například lineární velikost vzhledem k délce vstupu. To znamená, že nemáme dostatek místa na jejich uložení.

Pro $k$-limitované automaty využijeme pouze počátečních částí crossing posloupností, a to zhruba velikosti $k$. Pro simulaci výpočtu buněk s dále neměnným obsahem postačí popis pomocí tabulek přechodů představených v předchozí simulaci.

Označujme teď simulovaný automat $M$ a simulující automat $M'$. Dále předpokládejme, že $M'$ místo jazyka $L$ přijímá $\triangleright L \triangleleft$ (tento předpoklad později odstraníme), že $M'$ začíná výpočet s prázdným zásobníkem na symbolu $\triangleright$, a že $M$ přijímá v libovolném stavu pomocí překročení pravé zarážky a na konci výpočtu má na pásce všechny symboly z $\Gamma_k$. Zamrzlé pásky na konci výpočtu lze jednoduše dosáhnout umělým $k$-násobným průchodem pásky.

Během výpočtu lze obsah pásky v závislosti na přechodech popsat pomocí crossing posloupností na obou stranách buňky a znaku z $\Gamma_k$ obsaženého v buňce po $k$ průchodech. Definujme formálně množinu $\texttt{active}(a)$ všech takových možností.

**Definice 21**

Nechť $a \in \Sigma$, potom $\texttt{active}(a) \subseteq Q^* \times \Sigma \times Q^*$, kde pro každou trojici

$(s_1 s_2 \dots s_l, b, t_1 t_2 \dots t_r)$ s $s_1, \dots, s_l, t_1, \dots, t_r \in Q,\ l \geq 1, r \geq 0,\ b \in \Gamma_k$ tak, že existuje $h \in \mathbb{N},\ 1 \leq h \leq k$, stavy $p_1, p_2, \dots, p_{2h}$, symboly $\gamma_0, \gamma_1, \dots, \gamma_{h-1} \in \Gamma \setminus \Gamma_k,\ \gamma_h \in \Gamma_k$ a směry $d_0 = -1,\ d_1, d_2, \dots, d_h \in \{-1, 1\}$ a platí:

(i) $\gamma_0 = a, \gamma_h = b,\ \forall j \in \{1, \dots, h\} \colon (p_{2i}, \gamma_i, d_i) \in \delta(p_{2i-1}, \gamma_{i-1})$,

(ii) $s_1 s_2 \dots s_l$ je podposloupnost $p_1 p_2 \dots p_{2h}$ všech $p_j$ splňujících $d_{\lfloor j/2 \rfloor} = -1$,

(iii) $t_1 t_2 \dots t_r$ je podposloupnost $p_1 p_2 \dots p_{2h}$ všech $p_j$ splňujících $d_{\lfloor j/2 \rfloor} = 1$.

Z technických důvodů dodefinujeme množiny pro zarážky:

- $\texttt{active}(\triangleright) = \{(q_0, \triangleright, t) \mid (q_0, \triangleright, 1) \in \delta(q_0, \triangleright)\}$,
- $\texttt{active}(\triangleleft) = \{(s, \triangleleft, t) \mid (s, \triangleleft, 1) \in (s, \triangleleft)\} \cup \{(st, \triangleleft, \epsilon) \mid (t, \triangleleft, -1) \in \delta(s, \triangleleft)\}$

Trojice z $\texttt{active}(a)$ definují prvních $h$ přechodů hlavy přes buňku, přičemž je obsah buňky následně zamrzlý, a to i když $h < k$. To může být způsobeno tím, že hlava mění svůj směr přímo na buňce a tím její obsah podle definice přepíše, jako by přes ni prošla dvakrát.

Pokud hlava opustila buňku ve směru $d_{i-1}$, víme že při případném opětovném vstupu do buňky musí hlava přijít opět ze strany $d_{i-1}$. Pro $i \in \{2, \dots, h\}$ tedy víme, že přechod $i$ nastává ze stavu $p_{2i-1}$ ze směru $d_{i-1}$ a přechází do stavu $p_{2i}$ směrem $d_i$. Pro $i = 1$ víme, že hlava musí přijít z levé strany, proto dodefinujeme $d_0 = 0$. Z toho plyne, že pokud $d_i = 1$, pak $p_{2i}$, případně $p_{2i+1}$, pokud existuje, patří do posloupnosti $t_1 t_2 \dots t_r$. Zbytek stavů patří do posloupnosti $s_1 s_2 \dots s_l$. Levá posloupnost značí stavy při přechodu levé hranice buňky a pravá posloupnost stavy přechodu pravé hranice. Navíc pro stavy $s_i$ s lichým indexem platí, že jsou dosaženy při vstupu do buňky z levé strany a se sudým indexem při opuštění buňky směrem vlevo. Pro stavy $t_i$ s lichým indexem zase platí, že jich automat dosáhne při opouštění buňky vpravo a sudé indexy jsou dosaženy při opětovném vstupu zprava. Každá buňka musí být v akceptujícím výpočtu navštívena alespoň jednou, nicméně všechny průchody buňkou, které přepisují její obsah, mohou vést doleva, je tedy možné $r = 0$. Protože $l + r = 2h$, musí být $l + r$ sudé. Máme tedy dvě možnosti:

- $l$ i $r$ jsou sudé. Snadným pozorováním zjistíme, že potom poslední průchod přepisující buňku směřuje vlevo a vstupuje do stavu $s_l$ a
- $l$ a $r$ jsou liché. Analogickým pozorováním dostáváme poslední aktivní průchod vedoucí vpravo do stavu $t_r$.

Dále potřebujeme rozšířit některé pojmy představené v předchozí simulaci.

- Přechodové tabulky popisující pomocí čtveřic chování na zamrzlém segmentu pásky. Zde rozšíříme definici o speciální případ tabulky pro prázdné slovo $\tau_\epsilon$. Tabulka $\tau_\epsilon$ proto popisuje případ, kdy hlava přechází na sousední buňku, tedy
$$\tau_\epsilon = \{(p, d_1, q, d_2) \mid p = q \ \wedge \ d_1 = -d_2\}.$$
Složení tabulek definujeme stejně jako v předchozí simulaci.

- Rovněž budeme potřebovat i už představované množiny $\texttt{exit}(T_l, q, 1, T_r)$ pro výčet možností, s jakými může celý segment popisovaný tabulkami $T_l$ a $T_r$ hlava opustit, nachází-li se na pravé straně levého segmentu.
- $\texttt{nSelect}(S)$ rozšíříme vzory $\alpha$ zapisované ve tvaru $(p, d_1, q, d_2)$, kde $p, q \in S \cup \{\cdot\}$ a $d_1, d_2 \in \{-1, 1, \cdot\}$. Znak $\cdot$ zde představuje libovolnou hodnotu daného typu. Zápisem $\texttt{nSelect}(S|_\alpha)$ tak rozumíme pouze prvky $\texttt{nSelect}(S)$ odpovídající vzoru $\alpha$.

Nyní jsme schopni popsat algoritmus simulace. Rozdělíme jej na dva algoritmy pro větší přehlednost.

```
1  (q_0, ▷, t_1 t_2 ... t_r) ← nSelect(active(▷))
2  T ← τ_▷
3  repeat
4     push(t_1 t_2 ... t_r T)
5     T ← τ_ϵ
6     a ←read()
7     (s_1 s_2 ... s_l, b, t_1 t_2 ... t_r) ← nSelect(active(a))
8     s ←pop()
9     if s ≠ s_1
10       REJECT
11    for j ← 1 to ⌊(l − 1)/2⌋
12       connectLeft(s_2j, s_2j+1)
13    if l is odd
14       T ← T · τ_b
15    else
16       s ←nSelect(S)
17       t ←nSelect(S)
18       connectLeft(s_l, s)
19       T ← T · τ_b
20       connectLeft(s, t)
21       r++
22       t_r ← t
23    if peek() is table
24       T' ←pop()
25       T ← T' · T
26 until a = ◁
27 if stack is empty
28    ACCEPT
29 else
30    REJECT
```

Řádky 1 a 2 inicializují algoritmus. Do $(q_0, \triangleright, t_1 t_2 \dots t_r)$ se přiřadí jedna z trojic z $\texttt{active}(\triangleright)$ a jako aktuální tabulka $T$ se použije přechodová tabulka pro znak $\triangleright$.

V každé iteraci vnějšího cyklu se přečte nový symbol, simulujeme tím první návštěvu daného políčka pásky. Poté se simulují kroky spojené s výpočtem na již navštívených políčkách, ty je však nutné simulovat pouze na zásobníku.

Na začátku iterace jsou nejprve na zásobník vloženy nezpracované stavy crossing posloupnosti z minulé iterace a aktuální tabulka přechodů. Tato tabulka

označuje zamrzlý segment na pásce počínaje pozicí aktuálně čteného znaku $a$ konče nejlevější hranicí mezi buňkami tak, že mezi začátkem a koncem tohoto segmentu neleží žádná hranice s nezpracovanými stavy. Protože jsme na zásobník uložili novou crossing posloupnost, je nutné upravit i tabulku $T$, která teď popisuje prázdný segment. Podle definice `active`($a$) sice může být $r = 0$, nicméně tato crossing sekvence je z předešlé iterace vždy upravena tak, že $r$ je liché, proto je zásobník přidáno alespoň jedno $t_i$. Lichost $r$ vždy zabezpečuje else větev na řádku 15.

Po přečtení nového symbolu $a$ náhodně vygenerujeme trojici z `active`($a$). Protože na vrcholu zásobníku (díky lichosti $r$) musí být stav po prvním přechodu z nejpravější přečtené nezamrzlé buňky směrem vpravo, můžeme zkontrolovat, jestli je shodný se stavem, ve kterém se nachází automat při prvním dosažení aktuální buňky. To je zařízeno na řádcích 8–10. Na řádcích 11 a 12 můžeme vidět kontrolu ostatních stavů levé crossing posloupnosti. To zařizuje algoritmus `connectLeft`($q,\ q'$), pro zatím předpokládejme, že pokud byly crossing posloupnosti vygenerovány špatně a neshodují se s nezpracovanými stavy na zásobníku, automat zamítá, dále také tento algoritmus může zjednodušovat obsah pásky. Pokud všechna volání `connectLeft`($q,\ q'$) projdou bez zamítnutí, znamená to, že jsme zpracovali lichý počet stavů sekvence $s_1 s_2 \dots s_l$. Dále mohou nastat dvě situace.

Pokud $l$ bylo liché, pak jsme zpracovali všechny stavy levé crossing posloupnosti a jak bylo naznačeno dříve poslední aktivní návštěva posunula hlavu vpravo na další nezpracovaný znak. Zbývá tedy zvětšit aktuální tabulku přechodů o tabulku pro konečný obsah políčka.

Pokud je ale $l$ sudé, zbývá zpracovat jeden stav $s_l$, přitom víme, že do něj automat vstoupí při odchodu z políčka vlevo. Nedeterministicky tedy vybereme další dva stavy $s$ a $t$. Stav $s$ má význam stavu při opětovném navštívení buňky, zatímco stav $t$ reprezentuje stav dosažený při prvním opuštění aktuální buňky směrem vpravo. Na řádku 18 provedeme kontrolu kompatibility stavu opuštění a opětovného příchodu pomocí algoritmu `connectLeft`($q,\ q'$). Tím jsou všechny stavy z levé crossing posloupnosti zpracovány a buňka nelze dále přepisovat. Můžeme tak opět zvětšit aktuální tabulku přechodů. Na řádku 20 potom ověříme správný odhad stavu $t$. Stav $t$ přidáme k pravé crossing posloupnosti, abychom tak zaručili lichost $r$. Pokud totiž bylo $l$ liché, $r$ muselo být také, aby bylo $l + r$ sudé. Pokud je ale $l$ sudé, potom $r$ bylo na řádku 20 také sudé a přidáním stavu $t$ k posloupnosti se $r$ stane liché.

V obou případech se ale na řádku 23 může stát, že všechny stavy pravé crossing sekvence levé hranice segmentu aktuální tabulky byly zpracovány. V takovém případě je na vrcholu zásobníku přechodová tabulka. Tuto tabulku můžeme bezpečně spojit s aktuální tabulkou.

Výpočet končí, jestliže byl zpracován i poslední symbol vstupu a tím je vždy pravá zarážka. Zbývá zkontrolovat jestli byly všechny tabulky i stavy crossing posloupností vygenerovány dobře, v takovém případě je totiž na konci výpočtu zásobník prázdný.

Dále je potřeba popsat algoritmus `connectLeft`$(q,\ q')$.

```
1  connectLeft(q, q')
2      (q, 1, p, d) ← (T|_(q,1,·,·))
3      while d = −1
4          Y ←pop()
5          if Y = p
6              p′ ←pop()
7              (p′, −1, p, d) ←nSelect(T|_(p′,−1,·,·))
8          else if Y is a table
9              (p, d) ←nSelect(exit(Y, p, −1, T))
10             T ← Y · T
11         else
12             REJECT
13     if p ≠ q′
14         REJECT
```

Tento algoritmus vybere z aktuální tabulky $T$ jednu čtveřici $(q, 1, p, d)$ s pevně daným směrem 1 a stavem $q$ zadaným parametrem. Jinými slovy vygenerujeme si, do jakého stavu a jakým směrem automat vyjde z aktuálního zamrzlého segmentu, vejde li do něj zprava ve stavu $q$. Poté v cyklu z informací dostupných na zásobníku hledáme, kdy z aktuálního segmentu (který se může v průběhu zvětšovat) vyjde směrem doprava, tedy na nový symbol.

V každé iteraci, kdy $d = -1$ víme, že z aktuálního segmentu hlava vyšla doleva. Pokud je na vrcholu zásobníku uložen nějaký nezpracovaný stav $p$, můžeme jej zpracovat, $p$ byl na vrcholu zásobníku jako nezpracovaný stav z crossing posloupnosti. Díky řádku 9 prvního algoritmu a použití `connectLeft`$(q,\ q')$ je v tomto algoritmu počet stavů mezi jednotlivými tabulkami na zásobníku vždy sudý. To znamená, že do stavu $p$ automat překračoval levou hranici aktuálního segmentu a ve stavu, který se nachází jako další na zásobníku opět do aktuálního segmentu vchází. Můžeme jej tedy také odstranit ze zásobníku a vygenerovat další výsledek výpočtu při vstupu do aktuálního segmentu.

Pokud se na vrcholu zásobníku místo stavu nacházela tabulka přechodů, nacházíme se mezi dvěma zamrzlými segmenty se směrem hlavy $d = -1$ a můžeme vygenerovat výsledek vstupu do celého segmentu a pomocí řádku 10 obě tabulky spojit.

Jakmile konečně z aktuálního segmentu hlava přejde doprava, stačí zkontrolovat, jestli se automat nachází ve stavu $q'$.

Známým trikem z předešlé simulace, totiž hádáním koncového symbolu, můžeme automat upravit tak, aby slova neobsahovaly zarážky.

## 3.4 Simulace 1-LA pomocí 0-LA

Pighizzini ve své práci [2] pojednává o této simulaci pomocí detailního popisu $\delta$ přechodové funkce automatu. Tuto simulaci však můžeme podobně jako v minulých podkapitolách jednoduše přepsat pomocí algoritmu.

Použijeme k tomu, již představované přechodové tabulky. V tomto případě

však půjde o jednodušší strukturu, jelikož každou buňku je možné přepsat pouze jednou. Proto každá buňka, která byla navštívena představuje pravou hranici zamrzlého segmentu, a ten začíná na začátku pásky, protože všechny buňky v tomto prostoru už musely být přepsány. Tabulka tedy obsahuje pouze dvojice $(p, q)$ z $Q \times Q$. Ve stavu $p$ automat do segmentu vchází zprava a ve stavu $q$ opět vychází na stejnou stranu.

```
1  T ← τ▷
2  q ← q0
3  repeat
4      a ←read()
5      (q, b, d) ←nSelect(δ(q, a))
6      while d = −1
7          (q, q′) ←nSelect(T|(q,·))
8          (q, b, d) ←nSelect(δ(q′, b))
9      T ← T · τb
10 until a = ◁
11 ACCEPT
```

Tabulku inicializujeme pro levou zarážku a poté podobně jako v podkapitole 3.2 procházíme v každé iteraci jeden nový symbol, dokud nezpracujeme i pravou zarážku. Zde opět uvažujeme, že automat přijímá v libovolném stavu překročením zarážky.

Po načtení znaku $a$ v každé iteraci nedeterministicky vybereme další krok simulovaného automatu. Zde dokud je dalším krokem posunuta hlava vlevo, opakujeme výběr pomocí přechodové tabulky a nového kroku simulovaného automatu na pravé hranici aktuálního segmentu. Připomeňme, že ačkoliv se v celém algoritmu nevyskytuje explicitně příkaz `REJECT`, automat může vstupní slovo nepřijímat buď cyklením ve vnějším cyklu, nebo pokud je argument volání `nSelect` prázdná množina.

Pokud konečně nastane posun hlavy vpravo, zbývá nám aktualizovat aktuální přechodovou tabulku $T$ o přechodovou tabulku symbolu $b$ a pokračovat na další iteraci, případně akceptovat. Pokud totiž na konci iterace platí $a = \triangleleft$, musí být $d = 1$.

Protože $|Q|$ je konečné a tím pádem tabulek $T$ je konečně mnoho, je možné si ji zapamatovat ve stavu simulujícího automatu, tedy jsme schopni simulaci provést pomocí NFA.

# 4 Vícepáskové limitované automaty

Nyní se podívejme na další rozšíření limitovaných automatů. Jakou sílu bude mít LA s více páskami? U klasických Turingových strojů a lineárně omezených automatů nehraje počet pásek na jejich sílu roli, více pásek zde může pouze usnadnit implementaci.

Simulovat $n$ pásek v LBA je jednoduché. Stačí každou buňku rozdělit na $n$-tice obsahující buňky ze všech $n$ pásek na odpovídajících místech. Protože

pásková abeceda $\Gamma$ původního $n$-páskového stroje je konečná a nová abeceda $\Gamma'$ musí mít velikost shora omezenou hodnotou $|\Gamma|^n$, bude nová abeceda opět konečná.

Stejnou konstrukci však pro vícepáskové limitované automaty (mt-LA) použít nelze, protože každá buňka na jednotlivých páskách má svých $k$ přepisů. Zhuštěním do jedné pásky by se tak mohlo stát nejen, že se zvýší parametr $k$, ale také, že žádný takový $k'$-LA neexistuje, neboť opakované průchody přes již „zamrzlé“ buňky z jedné pásky mohou také vyčerpat všechny možnosti přepisu u ostatních pásek na stejné buňce (a to pro libovolně velké $k$). Můžeme proto nabýt podezření, že mt-LA mají větší sílu. Podívejme se tedy na tyto automaty detailněji.

V původní definici stačí přidat více pásek a upravit náležitě přechodovou funkci $\delta$. Pro úplnost si ale uvedeme celou definici.

## 4.1 Synchronní vícepáskové limitované automaty

**Definice 22**

(Synchronním) $s$-páskovým $k$-limitovaným automatem ($s$t-$k$-LA) pro dané $s \in \mathbb{N}$ a $k \in \mathbb{N}_0$ je šestice $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde

- $Q$ je konečná množina stavů,
- $F \subseteq Q$,
- $\Gamma$ je konečná abeceda páskových symbolů taková, že $\Gamma = \Gamma_0 \cup \Gamma_1 \cup \cdots \cup \Gamma_k$ $\forall i, j, 0 \leq i \leq k, 0 \leq j \leq k\colon \quad \Gamma_i \cap \Gamma_j \neq \emptyset \;\Rightarrow\; i = j$,
- $\Sigma$, kde $\Sigma \cup \{\square\} = \Gamma_0$ je konečná abeceda vstupních symbolů, [1]
- $\triangleright, \triangleleft \in \Gamma_k$ značí konce pásky,
- $\square \in \Gamma_0$ a
- $\delta\colon Q \times (\Gamma^s) \to 2^{Q \times (\Gamma^s) \times (\{-1,1\}^s)}$ je přechodová funkce.

Navíc pro všechny $c_i, c'_i, d_i$, kde $c = (c_1, c_2, \ldots, c_s)$, $d = (d_1, d_2, \ldots, d_s)$, $c' = (c'_1, c'_2, \ldots, c'_s)$, $(q, c', d) \in \delta(p, c)$, $p, q \in Q$, $c_i \in \Gamma_h$, $c'_i \in \Gamma_{h'}$ a $d_i \in \{-1, 1\}$, vyžadujeme:

- $h = k \quad \Rightarrow \quad c_i = c'_i \wedge h = h'$,
- $h < k \wedge d_i = 1 \quad \Rightarrow \quad h' = \min\left(\lceil \frac{h}{2} \rceil \cdot 2 + 1, k\right)$ a
- $h < k \wedge d_i = -1 \quad \Rightarrow \quad h' = \min\left(\lceil \frac{h+1}{2} \rceil \cdot 2, k\right)$

[1] Symbol $\square$ je určen jako výchozí hodnota buněk ostatních pásek, není však vyloučen jeho výskyt ve vstupních slovech. ($\square \in \Sigma$)

a aby pro každé $p, q \in Q, i \in \{1, \ldots, s\}$ platilo:

$$\begin{aligned} &\forall c_i, c'_i, d_i, c = (c_0, c_1, \ldots c_i = \triangleright, \ldots, c_s), \\ &\quad c' = (c'_0, c'_1, \ldots, c'_s), (q, c', d) \in \delta(p, c): \\ &\qquad d_i = 1 \end{aligned} \tag{8}$$

$$\begin{aligned} &\forall c_i, c'_i, d_i, c = (c_0, c_1, \ldots c_i = \triangleleft, \ldots, c_s), \\ &\quad c' = (c'_0, c'_1, \ldots, c'_s), (q, c', d) \in \delta(p, c): \\ &\qquad d_i = -1 \end{aligned} \tag{9}$$

Konfigurace a relaci odvození konfigurace v jednom kroku potom můžeme definovat podobně jako v Definici 14. Konfigurace mt-LA je prvek množiny $C = Q \times ((\Gamma^*)^s) \times (\mathbb{N}^s)$ a relaci odvození $\vdash \subseteq C \times C$ splňuje následující vztah:

$$\begin{aligned} &(p, (\triangleright w_{1l} c_{1i_1} w_{1r} \triangleleft \,.\, \triangleright\, w_{2l} c_{2i_1} w_{2r} \triangleleft, \ldots, \triangleright w_{sl} c_{si_1} w_{sr} \triangleleft), (i_1, i_2, \ldots, i_s)) \vdash \\ &\vdash (q, (\triangleright w_{1l} c'_{1i_1} w_{1r} \triangleleft \,.\, \triangleright\, w_{2l} c'_{2i_1} w_{2r} \triangleleft, \ldots, \triangleright w_{sl} c'_{si_1} w_{sr} \triangleleft), (i'_1, i'_2, \ldots, i'_s)) \Leftrightarrow \\ &\Leftrightarrow (q, (c'_{1i_1}, c'_{2i_2}, \ldots c'_{si_s}), (i'_1 - i_1, i'_2 - i_2, \ldots i'_s - i_s)) \in \delta(p, (c_{1i_1}, c_{2i_2}, \ldots c_{si_s})) \end{aligned}$$

Počáteční konfigurací pak rozumíme libovolnou trojici ve tvaru

$$(q_0, (\triangleright w_1 \triangleleft, \triangleright w_2 \triangleleft, \ldots, \triangleright w_s \triangleleft), (i_1, i_2, \ldots, i_s)),$$

kde $w_1 \in \Sigma^*$, $w_2 = w_3 = \cdots = w_s = \square^{|w_1|}$ a $i_1 = i_2 = \cdots = i_s = 1$ a koncovou konfigurací rozumíme konfiguraci obsahující některý koncový stav.

V definici očividným způsobem rozšiřujeme standardní limitované automaty a přechody mezi konfiguracemi na více pásek.

Už jsme naznačili, že mt-LA se zdá být silnější než LA. A skutečně není těžké nalézt mt-LA, který rozpoznává jazyk nepatřící do třídy bezkontextových jazyků. Dokonce nám k tomu postačí determinismus.

Příklad 23

Mějme jazyk $L = \{a^i b^i c^i \mid i \geq 0\}$. Pro jeho rozpoznání použijeme 2t-1-dLA $\mathcal{A}$. $\mathcal{A}$ si nejprve na druhou pásku zkopíruje nejdelší prefix z $\{a\}^*$ a potom opakovaně prochází tento prefix společně s hlavou druhé pásky, která kontroluje, jestli sekvence dalšího symbolu končí společně s daným prefixem (neboli jestli mají stejnou délku).

Dále než na kontextové jazyky se však s mt-LA bohužel nedostaneme, protože lze triviálně simulovat pomocí vícepáskového LBA a ten pomocí LBA, jak už bylo popsáno.

Další jazyk, se kterým už jsme se setkali, je množina všech palindromů nad dvouprvkovou abecedou. Tento jazyk sice patří do CFL a je tak „jednodušší“ než předchozí, ukážeme si na něm ale dvě zajímavé věci, proto uvedeme celou

funkci $\delta$. Na první pohled by se mohlo zdát, že mají vícepáskové nula limitované automaty stejnou sílu jako ty jednopáskové, protože takový automat není schopný si ani přepsat vstupní slovo a ostatní pásky tak zůstanou prázdné. Dle definice však mají i tyto pásky délku vstupního slova a automat toho dokáže využít k rozpoznání větší třídy jazyků.

Příklad 24

Nechť $L_{\text{pal}} = \{w \in \{a,b\}^* \mid w = w^{\text{R}}\}$. $L_{\text{pal}}$ generuje třípáskový nula limitovaný deterministický automat $\mathcal{A}_{\text{pal}} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde

$Q \in \{q_{\text{double}}, q_{\text{double-}}, q'_{\text{double-}}, q_{\text{step1}}, q_{\text{step2}}, q_{\text{rej}}, q_{\text{acc}}, 1_{\text{sr}}, 1_{\text{sl}}, 1_{\text{check}(a)\text{r}}, 1_{\text{check}(b)\text{r}}, 1_{\text{check}(a)\text{l}}, 1_{\text{check}(b)\text{l}}, 2_{\text{sr+}}, 2_{\text{check}(a)+}, 2_{\text{check}(b)+}, 2_{\text{check}(a)}, 2_{\text{check}(b)}, 2_{\text{check}(a)\text{ret}}, 2_{\text{check}(b)\text{ret}}, 2_{\text{ret}}, 2_{\text{ret+}}\}$,

$\Sigma = \{a,b\}$, $\Gamma = \Sigma \cup \{\square\}$,$q_0 = q_{\text{double}}$, $F = \{q_{\text{acc}}\}$, a $\delta$ je definovaná následovně:

První část kódu využívá sílu pouze dvou pásek, třetí páska pouze kopíruje pohyb pásky druhé. V této části automat hledá střed pásky. Využíváme zde myšlenky, že druhou pásku můžeme posunovat dvojnásobnou rychlostí. Najde-li konec, musí být první páska ve středu. Technicky bohužel není možné, aby byly hlavy vždy přesně v dvojnásobných vzdálenostech, můžeme se ale pohybovat tak, že jsme vždy nanejvýš o dvě pole mimo aktuální střed, přitom automat zná jeho přesnou polohu.

1. $\delta(q_{\text{double}}, (x, \square, \square)) = (q_{\text{double-}}, (x, \square, \square), (1,1,1))$ $\quad x \in \Sigma$
2. $\delta(q_{\text{double-}}, (x, \square, \square)) = (q_{\text{step1}}, (x, \square, \square), (1,1,1))$ $\quad x \in \Sigma$
3. $\delta(q_{\text{step1}}, (x, \square, \square)) = (q_{\text{step2}}, (x, \square, \square), (-1,1,1))$ $\quad x \in \Sigma$
4. $\delta(q_{\text{step2}}, (x, \square, \square)) = (q_{\text{double}}, (x, \square, \square), (1,1,1))$ $\quad x \in \Sigma$
5. $\delta(q_{\text{double}}, (x, \triangleleft, \triangleleft)) = (1_{\text{sr}}, (x, \triangleleft, \triangleleft), (-1,-1,-1))$ $\quad x \in \Sigma$
6. $\delta(q_{\text{step1}}, (x, \triangleleft, \triangleleft)) = (1_{\text{sl}}, (x, \triangleleft, \triangleleft), (-1,-1,-1))$ $\quad x \in \Sigma$
7. $\delta(q_{\text{step2}}, (x, \triangleleft, \triangleleft)) = (2_{\text{sr+}}, (x, \triangleleft, \triangleleft), (-1,-1,-1))$ $\quad x \in \Sigma$
8. $\delta(q_{\text{double-}}, (x, \triangleleft, \triangleleft)) = (q'_{\text{double-}}, (x, \triangleleft, \triangleleft), (-1,-1,-1))$ $\quad x \in \Sigma$
9. $\delta(q'_{\text{double-}}, (x, \square, \square)) = (2_{\text{ret}}, (x, \square, \square), (-1,-1,1))$ $\quad x \in \Sigma$

V pravidlech 1-4 probíhá jakýsi cyklus, vždy ve stavu $q_{\text{double}}$ je hlava první pásky na pravém z prostředních dvou políček v úseku počínajícím za znakem $\triangleright$ a končícím před pozicí ostatních hlav. Ve stavu $q_{\text{double-}}$ je prostřední pole o jedno pole před pozicí první hlavy, a ve stavech $q_{\text{step1}}$ a $q_{\text{step2}}$ se opět dorovnáváme na střed do stavu $q_{\text{double}}$.

Pravidla 5-8 popisují čtyři různé situace, které mohou nastat až druhá a třetí hlava naleznou konec pásky. První hlava se teď nachází přibližně uprostřed, přesná poloha středu je však známá díky aktuálnímu stavu. Pravidlo 8 ještě

upravuje pozici hlav. Výstupem je teď tedy přibližná pozice středu pásky a jeden ze čtyř stavů, $1_{\mathrm{sl}}$ a $1_{\mathrm{sr}}$ pro sudou délku pásky, $2_{\mathrm{sr}+}$ a $2_{\mathrm{ret}}$ pro lichou délku pásky, jejich význam pochopíme v následujícím kódu.

Každý tento případ je v zásadě nutné řešit zvlášť. Naštěstí je však možné sjednotit další postup pro dvojice stavů představující lichou délku vstupu a sudou délku vstupu.

Následující část kódu ukazuje, jak automat porovnává jednotlivé znaky slova od středu směrem ke koncům. Simulujeme zde pohyb hlavy pro 1-LA, není však možné si značit, které symboly už jsme prošli. K tomu slouží druhá a třetí páska.

Kvůli nutnosti pohybovat všemi třemi hlavami současně je implementace opět složitější, ale intuice je taková, že jedna ze dvou k tomu určených hlav je na konci pásky, zatímco druhá je od konce pásky ve vzdálenosti, která je nutná přejít s první hlavou, abychom se dostali přes již zpracovanou oblast pásky. Tyto dvě hlavy na prázdných páskách potom jdou protichůdně dokud se jedna z nich nedostane na konec pásky a při další iteraci můžeme vzdálenost zvětšit o 2 tím, že hlavy posuneme obě stejným směrem. Následuje kód pro sudé délky vstupu.

10. $\delta(1_{\mathrm{sr}}, (x, \square, \square)) = (1_{\mathrm{check}(x)\mathrm{r}}, (x, \square, \square), (1, 1, -1))$ $x \in \Sigma$
11. $\delta(1_{\mathrm{check}(x')\mathrm{r}}, (x, \square, \square)) = (1_{\mathrm{check}(x')\mathrm{r}}, (x, \square, \square), (1, 1, -1))$ $x, x' \in \Sigma$
12. $\delta(1_{\mathrm{check}(x')\mathrm{r}}, (x', \triangleleft, \square)) = (1_{\mathrm{sl}}, (x', \triangleleft, \square), (1, -1, -1))$ $x' \in \Sigma$
13. $\delta(1_{\mathrm{check}(x')\mathrm{r}}, (y, \triangleleft, \square)) = (q_{\mathrm{rej}}, (y, \triangleleft, \square), (1, -1, -1))$ $x' \in \Sigma, x' \neq y$
14. $\delta(1_{\mathrm{sl}}, (x, \square, \square)) = (1_{\mathrm{check}(x)\mathrm{l}}, (x, \square, \square), (-1, -1, 1))$ $x \in \Sigma$
15. $\delta(1_{\mathrm{check}(x')\mathrm{l}}, (x, \square, \square)) = (1_{\mathrm{check}(x')\mathrm{l}}, (x, \square, \square), (-1, -1, 1))$ $x, x' \in \Sigma$
16. $\delta(1_{\mathrm{check}(x')\mathrm{l}}, (x', \square, \triangleleft)) = (1_{\mathrm{sr}}, (x', \square, \triangleleft), (-1, -1, -1))$ $x' \in \Sigma$
17. $\delta(1_{\mathrm{check}(x')\mathrm{l}}, (y, \square, \triangleleft)) = (q_{\mathrm{rej}}, (y, \square, \triangleleft), (-1, -1, -1))$ $x' \in \Sigma, x' \neq y$
18. $\delta(1_{\mathrm{sr}}, (\triangleright, \triangleright, \square)) = (q_{\mathrm{acc}}, (\triangleright, \triangleright, \square)), (1, 1, -1))$

Index sr, znamená, že je potřeba uložit aktuální znak na pozici první hlavy a poté zkontrolovat jeho pravý obraz, sl znamená, že je třeba zkontrolovat levý obraz.

V pravidlech 11-13 vidíme přechod na levý obraz, přitom si ve stavu pamatujeme znak $x$. Jakmile druhá hlava dojde na konec pásky, zamítáme, pokud se znak neshoduje s uloženým znakem. Pokud se shoduje, přecházíme do stavu $1_{\mathrm{sl}}$.

Pravidla 14-17 jsou analogická k pravidlům 10-13. Ať už jsme do tohoto bloku kódu vstoupili ze stavu $q_{\mathrm{double}}$, nebo $q_{\mathrm{step1}}$ při jakkoliv dlouhém vstupu (sudé délky), druhá páska dojde na levou zarážku pouze ve stavu $1_{\mathrm{sr}}$, v tomto případě podle pravidla 18 přijímáme slovo, protože již byl překontrolován celý vstup.

Další blok kódu se provádí pro vstupy liché délky. Je implementačně náročnější než předchozí, ale myšlenka zůstává stejná. Větší složitost je způsobena zejména tím, že je nutné střídavě vždy dvakrát provést zvětšení „zapamatované“ vzdálenosti a dvakrát ne. Zda je nutné zvětšovat si automat pamatuje pomocí

symbolu „+“ ve stavech. Mysleme také na to, že obrazem prostředního znaku je díky liché délce slova on sám, není tedy třeba kontrolovat.

| | | |
|---|---|---|
| 19. | $\delta(2_{\text{sr}+}, (x, \square, \square)) = (2_{\text{check}(x)+}, (x, \square, \square), (1, 1, -1))$ | $x \in \Sigma$ |
| 20. | $\delta(s, (x, \square, \square)) = (s, (x, \square, \square), (1, 1, -1))$ | $x \in \Sigma$, $s \in \{2_{\text{check}(x)}, 2_{\text{check}(x)+}\}$ |
| 21. | $\delta(2_{\text{check}(x')+}, (x, \triangleleft, \square)) = (2_{\text{check}(x')\text{ret}}, (x, \triangleleft, \square), (1, -1, -1))$ | $x, x' \in \Sigma$ |
| 22. | $\delta(2_{\text{check}(x')\text{ret}}, (y, \square, \square)) = (2_{\text{rej}}, (y, \square, \square), (1, 1, 1))$ | $x' \in \Sigma, x' \neq y$ |
| 23. | $\delta(2_{\text{check}(x')\text{ret}}, (x', \square, \square)) = (2_{\text{ret}}, (x', \square, \square), (-1, -1, 1))$ | $x' \in \Sigma$ |
| 24. | $\delta(s, (x, \square, \square)) = (s, (x, \square, \square), (-1, -1, 1))$ | $s \in \{2_{\text{ret}}, 2_{\text{ret}+}\}$ |
| 25. | $\delta(2_{\text{ret}}, (x, \square, \triangleleft)) = (2_{\text{check}(x)}, (x, \square, \triangleleft), (1, 1, -1))$ | $x \in \Sigma$ |
| 26. | $\delta(2_{\text{check}(x')}, (y, \triangleleft, \square)) = (q_{\text{rej}}, (y, \triangleleft, \square), (1, -1, 1))$ | $x' \in \Sigma, x' \neq y$ |
| 27. | $\delta(2_{\text{check}(x')}, (x', \triangleleft, \square)) = (2_{\text{ret}+}, (x', \triangleleft, \square), (-1, -1, 1))$ | $x' \in \Sigma$ |
| 28. | $\delta(2_{\text{ret}+}, (x, \square, \triangleleft)) = (2_{\text{sr}+}, (x, \square, \triangleleft), (-1, -1, -1))$ | $x \in \Sigma$ |
| 29. | $\delta(2_{\text{sr}+}, (\triangleright, \square, \square)) = (q_{\text{acc}}, (\triangleright, \square, \square), (1, 1, 1))$ | |
| 30. | $\delta(s, (\triangleright, \triangleright, \triangleleft)) = (q_{\text{acc}}, (\triangleright, \triangleright, \triangleleft), (1, 1, -1))$ | $s \in \{2_{\text{ret}}, 2_{\text{ret}+}\}$ |

Poznámka 25

V tomto příkladě nebylo vyhledání středu nutné, jelikož na rozdíl od jednopáskových LA nás počet přepisů nijak nelimituje, místo něj lze využít síly více pásek. Řešením by tedy bylo i porovnávání symbolů směrem od kraje ke středu.

Ve skutečnosti by jazyk z Příkladu 23 mohl být rozpoznávaný 2t-0-dLA. Idea spočívá v tom, že stejně jako dokáže stroj pomocí druhé pásky najít střed, dokáže najít i třetinu, a přitom kontrolovat přítomnost znaku „a“, poté následující třetinu pro přítomnost „b“ a tak dále. Povšimněme si, že v tomto konkrétním případě stačí jednodušší typ automatu k rozpoznání kontextového jazyka, který není bezkontextový, zatímco pro bezkontextový jazyk je třeba silnější typ mt-LA.

Jak již bylo naznačeno, není vždy možné držet jednu hlavu ve dvojnásobné vzdálenost od druhé. Je to důsledek následujícího lemma.

**Lemma 26**

*Nechť $\mathcal{A}$ je mt-LA s $s$ páskami. Potom pro libovolné dvě pásky $i, j$, $0 < i < j < s$ a libovolnou konfiguraci $c = (q, t, (n_1, n_2, \ldots, n_i, \ldots, n_j, \ldots, n_s))$ některého výpočtu $\mathcal{A}$ platí:*

$$n_i \equiv n_j (mod\ 2).$$

Nebylo by těžké sestrojit důkaz pro předchozí větu, jelikož v počáteční konfiguraci triviálně platí a každé odvození v jednom kroku je dle definice závislé na $\delta$

funkci a jednotlivé indexy v konfiguraci tedy na součtech indexů z předchozí konfigurace a čísel udávajících směr posunu hlavy. Protože pro směry jsou možné jen dvě hodnoty (-1 a 1) a pro libovolné dvě pásky dostáváme 4 možnosti s možnými součty -2, 0 a 2, je zřejmé, že rozdíly nových indexů budou opět kongruentní modulo 2.

## 4.2 Asynchronní vícepáskové limitované automaty

Technická náročnost implementace $\mathcal{A}_{\text{pal}}$ nás motivuje k přezkoumání Definice 22. U klasických LA nemá smysl uvažovat možnost nepohnout se s hlavou, jelikož při tomto pohybu lze jen měnit stav a žádná nová informace se během toho neobjeví. Dosáhli bychom tak jedině vyplýtvání počtu přepisů, popřípadě by tato možnost nepřinesla nové vlastnosti, pokud by instrukce bez přesunu hlavy aktuální buňku nepřepisovala. U mt-LA o tom však již má cenu uvažovat. Nejen, že potom bude možné docílit liché vzdálenosti mezi jednotlivými hlavami, ale nebude nutné s nimi neustále hýbat.

Tyto úvahy nás motivují k definici nového typu vícepáskových limitovaných automatů. Jednotlivé hlavy nebudou nuceny při každém kroku k pohybu, proto je nazveme *asynchronní* a mt-LA podle Definice 22 nazveme *synchronní*.

Definice asynchronních mt-LA je stejná jako v Definici 22 až na tři odlišnosti, přechodová funkce $\delta$ nabývá tvaru

$$\delta\colon Q \times (\Gamma^s) \to 2^{Q\times(\Gamma^s)\times(\{-1,\mathbf{0},1\}^s)},$$

bod

- $h = k \quad \Rightarrow \quad c_i = c'_i \ \wedge \ h = h'$

měníme na

- $h = k \ \vee \ d_i = 0 \quad \Rightarrow \quad c_i = c'_i \ \wedge \ h = h'$

a ve vztahu (8) a (9) povolíme i $d_i = 0$.

Tímto rozhodnutím dáváme hlavě možnost nepřepisovat žádné symboly, pokud se nehýbe. Dalším přirozeným rozhodnutím zachovávajícím původní myšlenku přepisovacích systémů by bylo, že buňka se při každém kroku, kdy $d_i = 0$, přepíše dvakrát, protože hlava přepisovacího systému by neustále přecházela nad buňkou a měnila svůj směr.

Otevřený problém 27

V této chvíli zůstává otevřeným problémem, jestli je třída asynchronních mt-LA silnější než třída synchronních mt-LA.

Domněnka 28

Stejně jako v Příkladu 24, u nula-limitovaných automatů se zdá, že si lze

vzájemné polohy hlav, kterých by jinak bylo možné docílit s asynchronním automatem, zapamatovat jako přibližnou polohu a konečný offset od této polohy si pamatovat ve stavu.

Naproti tomu některé jazyky, které lze přijímat asynchronními $i$-limitovanými automaty (pro $i > 0$), by nemuselo být možné akceptovat synchronními automaty. Simulace kroků, ve kterých se některá hlava nepohne, totiž může zbytečně plýtvat přepisy políček.

Implementace algoritmů pro asynchronní automaty je každopádně jednodušší, proto budeme dále pracovat s nimi.

Příklad 29

Nechť $L_{\text{exp2}} = \{w \in \{a\}^* \mid \exists n \in \mathbb{N}|w| = 2^n\}$. $L_{\text{exp2}}$ generuje dvoupáskový nula limitovaný deterministický automat $\mathcal{A}_{\text{exp2}} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde $Q = \{q_0, q_{\text{r1}}, q_{\text{r2}}, q_{\text{change}}, q_{\text{acc}}, q_{\text{rej}}\}$, $\Sigma = \{a\}$, $\Gamma = \Sigma \cup \{\square\}$, $F = \{q_{\text{acc}}\}$, a $\delta$ je definovaná následovně:

Horní hlava se inicializuje na konec vstupu, poté opakovaně kontroluje, jestli je tato pozice dělitelná dvěma, pokud je, zkrátí ji o polovinu, pokud ne, může automat zamítnout. Pro jednoduchost delta funkce se po každém zkrácení prohodí pozice pásek. Pokud detekuje automat aktuální délku 1, přijímá.

Zkrácení pozice hlavy o polovinu je uskutečnitelné za pomoci druhé hlavy, která je na pozici 0. Horní hlava se posouvá vlevo dvojnásobnou rychlostí oproti druhé hlavě, která se posunuje vpravo. Pokud tedy první hlava dojde na pozici nula (v sudém počtu kroků, jinak automat zamítá), druhá hlava je v poloviční vzdálenosti oproti první hlavě na začátku tohoto procesu.

Inicializace
$$\begin{cases} \delta(q_0, (\triangleleft, \triangleleft)) = (q_{\text{rej}}, (\triangleleft, \triangleleft), (0, 0)) \\ \delta(q_0, (\square, \square)) = (q_0, (\square, \square), (1, 0)) \\ \delta(q_0, (\triangleleft, \square)) = (q_{\text{r0}}, (\triangleleft, \square), (-1, -1)) \end{cases}$$

Zbytek po dělení dvěma pozice první hlavy je 0, pokud nebude automat zamítat
$$\begin{cases} \delta(q_{\text{r0}}, (a, x)) = (q_{\text{r1}}, (a, x), (-1, 0)) & x \in \{\triangleright, a\} \\ \delta(q_{\text{r0}}, (\triangleright, \square)) = (q_{\text{change}}, (\triangleright, \square), (0, 0)) \end{cases}$$

Zbytek po dělení pozice je 1, pokud bude automat zamítat
$$\begin{cases} \delta(q_{\text{r1}}, (a, x)) = (q_{\text{r0}}, (x, a), (-1, 1)) & x \in \{\triangleright, a\} \\ \delta(q_{\text{r1}}, (\triangleright, \square)) = (q_{\text{rej}}, (\triangleright, \square), (0, 0)) \end{cases}$$

1 je mocnina dvou $\left\{\delta(q_{\mathrm{r1}}, (\triangleright, \triangleright)) = (q_{\mathrm{acc}}, (\triangleright, \triangleright), (0, 0))\right.$

Záměna pozic pásek $\begin{cases}\delta(q_{\mathrm{change}}, (x, \square)) = (q_{\mathrm{change}}, (x, \square), (1, -1) & x \in \{\triangleright, a\} \\ \delta(q_{\mathrm{change}}, (a, \triangleright)) = (q_{\mathrm{r0}}, (a, \triangleright), (0, 0))\end{cases}$

**Inicializace**

Pokud je vstup prázdný, zamítáme. Jinak první hlavu nastavíme na poslední symbol vstupu (předposlední políčko pásky) a druhou hlavu na začátek pásky a nastavíme stav na $q_{\mathrm{r0}}$.

Pomocí dvou stavů $q_{\mathrm{r0}}$ a $q_{\mathrm{r1}}$ si pamatujeme sudost, či lichost vzdálenosti aktuální polohy první hlavy od její polohy na začátku aktuální kontroly.

**Zbytek po dělení dvěma pozice první hlavy je 0, pokud nebude automat zamítat**

Při kontrole sudosti vždy měníme stav na $q_{\mathrm{r1}}$ a přitom posuneme první hlavu o jedna. Při přechodu zpět na $q_{\mathrm{r0}}$ posuneme obě hlavy. Tím docílíme toho, že druhá hlava je od levého okraje vzdálena o polovinu vzdálenosti první hlavy od její původní polohy.

Pokud při kontrole sudosti dojde první hlava ve stavu $q_{\mathrm{r0}}$ na začátek pásky a druhá hlava je mezi zarážkami, víme, že test proběhl správně, navíc druhá páska je přesně v polovině původní polohy první hlavy. Přejdeme do stavu $q_{\mathrm{change}}$, pomocí kterého vzájemně vyměníme polohu hlav.

**Zbytek po dělení dvěma pozice první hlavy je 1, pokud bude automat zamítat**

Ve stavu $q_{\mathrm{r1}}$ opět přecházíme do $q_{\mathrm{r0}}$ a přitom posunujeme obě hlavy. Tím, že druhou hlavu posunujeme pouze ve stavu $q_{\mathrm{r1}}$ docílíme na konci kontroly poloviční vzdálenosti.

Pokud v tomto stavu dojde první hlava na začátek pásky, máme dvě možnosti. Buď je druhá hlava na symbolu $\square$, nebo $\triangleright$.

Pokud je hlava na symbolu $\square$, je zřejmé, že od začátku kontroly už musel být automat alespoň jednou ve stavu $q_{\mathrm{r1}}$ (jinak by se druhá hlava neposunula) a tedy kontrolované číslo není jedna a je liché. Celý vstup tak nemůže mít délku mocniny dvou a můžeme zamítnout.

V druhém případě rovnou přijímáme, protože jsme postupně dělili číslo dvěma, až jsme se dostali ke kontrole jedničky.

**Záměna pozic pásek**

Již zmíněná záměna poloh probíhá vzájemně protichůdným pohybem hlav,

dokud nedojde druhá hlava na začátek pásky. Pak přecházíme do stavu $q_{\mathrm{r}0}$, protože chceme kontrolovat sudost pozice (případně lichost v případě jedničky, to je však ošetřeno zvlášť).

### 4.2.1 Hierarchie asynchronních deterministických vícepáskových 0-limitovaných automatů

V této kapitole budeme zkratkou $m$t-0-LA rozumět asynchronní deterministickou verzi automatu. Pomocí množiny jazyků $\{L_{i\mathrm{pal}} \mid i > 0\}$ nad abecedou $\Sigma, |\Sigma| > 1$, kde

$$L_{i\mathrm{pal}} = \{w_1 w_2 \cdots w_{2^{i-2}} \mid \forall j, 1 \leq j \leq i \colon w_j \in \Sigma^+ \ \wedge \ w_j = w_j^{\mathrm{R}}\},$$

ukážeme, že $(k+1)$t-0-dLA je silnější než $k$t-0-dLA pro $k \geq 1$.

Všimněme si, že konkatenaci fixního počtu palindromů nad unární abecedou dokáže rozpoznat už DFA, proto se zaměříme na větší abecedy.

Nejprve ukážeme konstrukci automatů, které jazyky $L_{i\mathrm{pal}}$ rozpoznávají.

**Lemma 30**

*Existuje asynchronní deterministický dvoupáskový 0-LA akceptující $L_{2pal}$.*

*Důkaz*

V Příkladu 24 jsme se setkali s třípáskovým automatem akceptujícím jazyk $L_{2\mathrm{pal}}$. Na tomto příkladu jsme demonstrovali nepraktičnost synchronní verze. Nyní máme k dispozici asynchronicitu a za použití lepší strategie můžeme vytvořit dvoupáskový automat akceptující $L_{2\mathrm{pal}}$.

Pro jednoznačný popis automatu postačí definice $\delta$.

$$\text{Inicializace} \quad \begin{cases} \delta(q_0, (x, \square)) = (s_{\mathrm{l}x}(x, \square), (0, -1)) & x \in \Sigma \end{cases}$$

$$\text{Ukládání pozice } i \text{ symbolu } x \text{ na druhou pásku} \quad \begin{cases} \delta(s_{\mathrm{l}x}, (y, z)) = (s_{\mathrm{l}x}, (y, z), (-1, 1)) & x, y \in \Sigma \\ & z \in \{\square, \triangleright\} \\ \delta(s_{\mathrm{l}x}, (\triangleright, \square)) = (l_{\mathrm{r}x}, (\triangleright, \square), (0, 0)) & x \in \Sigma \end{cases}$$

$$\text{Načtení pozice } l+1-i, \text{ kde } l \text{ je délka vstupu pro porovnání se symbolem } x \quad \begin{cases} \delta(l_{\mathrm{r}x}, (y, \square)) = (l_{\mathrm{r}x}, (y, \square), (1, 1)) & x \in \Sigma \\ & y \in \Sigma \cup \{\triangleright\} \\ \delta(l_{\mathrm{r}x}, (x, \triangleleft)) = (c_{\mathrm{r}}, (x, \triangleleft), (-1, 0)) & x \in \Sigma \end{cases}$$

$$\text{Uložení symbolu } x \text{ pro porovnání} \quad \begin{cases} \delta(c_{\mathrm{r}}, (x, \triangleleft)) = (s_{\mathrm{r}x}, (x, \triangleleft), (0, 0)) & x, y \in \Sigma \\ \delta(c_{\mathrm{r}}, (\triangleright, \triangleleft)) = (q_{acc}, (\triangleright, \triangleleft), (0, 0)) & x, y \in \Sigma \end{cases}$$

$$\text{Zrcadlová pravidla předchozím} \begin{cases} \delta(s_{\mathrm{r}x}, (y, z)) = (s_{\mathrm{r}x}, (y, z), (1, -1)) & x, y \in \Sigma \\ & z \in \{\square, \triangleleft\} \\ \delta(s_{\mathrm{r}x}, (\triangleleft, \square)) = (l_{\mathrm{l}x}, (\triangleleft, \square), (0, 0)) & x \in \Sigma \\ \delta(l_{\mathrm{l}x}, (y, \square)) = (l_{\mathrm{l}x}, (y, \square), (-1, -1)) & x \in \Sigma \\ & y \in \Sigma \cup \{\triangleleft\} \\ \delta(l_{\mathrm{l}x}, (x, \triangleright)) = (c_{\mathrm{l}}, (x, \triangleright), (1, 0)) & x \in \Sigma \\ \delta(c_{\mathrm{l}}, (x, \triangleright)) = (s_{\mathrm{l}x}, (x, \triangleright), (0, 0)) & x, y \in \Sigma \\ \delta(c_{\mathrm{l}}, (\triangleleft, \triangleright)) = (q_{acc}, (\triangleleft, \triangleright), (0, 0)) & x, y \in \Sigma \end{cases}$$

Druhá páska automatu nedokáže rozlišovat jednotlivé symboly vstupu, ale lze využít jako counter o kapacitě délky vstupu. Uvedený automat střídavě na obou stranách pomocí stavu ukládá aktuální symbol $c_i$ a následnými kroky kontroluje, zda pro vstup $w = c_0 c_1 \dots c_i \dots c_{l-1-i} \dots c_{l-2} c_{l-1}$ platí $c_i = c_{l-1-i}$. Kontrola dílčích symbolů probíhá tak, že se nejdříve pomocí souběžného chodu obou hlav v protisměru uloží na druhou pásku číslo $i$ a poté se souběžným chodem obě hlavy posunou o $l - 1 - i$, čímž je nalezen prvek v symetrické pozici vůči středu vstupu k porovnání se zapamatovaným symbolem. Postup se potom provádí zrcadlově v sudých iteracích, aby se zamezilo zbytečnému pohybu hlav na původní pozici $i$.

Všimněme si, že ačkoliv je automat schopen po každé iteraci kontrolovat, jestli je už $i \geq \frac{l-1}{2}$, není to nutné. Jednodušší (z pohledu počtu instrukcí) je nechat automat kontrolovat už jednou kontrolované dvojice znovu (v opačném pořadí) dokud neprojde celý vstup. Tuto strategii nám umožňuje vlastnost palindromu.

Nyní si ukažme, že automat rozpoznává skutečně právě palindromy. Pro tento účel vhodné místo konfigurací $(p, (w_1, w_2), (i_1, i_2))$ použít úspornější $(p, i_1, i_2)$, jelikož $w_1 = \triangleright\, w \triangleleft$ i $w_2 = \triangleright \square^l \triangleleft$ jsou jednoznačně dané vstupním slovem $w$ délky $l$ a v průběhu výpočtu se nemohou změnit. Také definujme funkce

$$t_1(i) = \begin{cases} \triangleright & i = 0 \\ \triangleleft & i = l + 1 \\ (i-1)\text{tý prvek vstupu } w & \text{jinak} \end{cases}$$

a

$$t_2(i) = \begin{cases} \triangleright & i = 0 \\ \triangleleft & i = l + 1 \\ \square & \text{jinak} \end{cases}$$

vracející symbol na $i$-tém místě na páskách.

### Inicializace

Je-li vstupem prázdné slovo, nelze použít žádné pravidlo a automat neakceptuje. Pokud není vstup prázdný, víme, že $t_1(1), t_2(1) \notin \{\triangleright, \triangleleft\}$, tudíž všechny

symboly povedou ke konfiguraci $(s_{\mathrm{l}t_1(1)}, 1, 0)$. Přitom do stavu $s_{\mathrm{l}x}$ se lze dostat (kromě stavu samotného) jedině ze stavu $q_0$ nebo z $c_{\mathrm{l}}$.

### Ukládání pozice na druhou pásku

Ze stavu $q_0$ se do $s_{\mathrm{l}x}$ dostane pouze z první konfigurace, protože $q_0$ není na pravé straně žádného dalšího pravidla a při použití prvního pravidla ze sekce inicializace tedy musí být druhá hlava na pozici nula, při použití pátého pravidla ze sekce zrcadlových pravidel musí být druhá hlava též na pozici nula a po provedení se ani jedna z hlav nepohne. Sekce ukládání pozice $i$ tedy vždy bude začínat v konfiguraci $(s_{\mathrm{l}t_1(i)}, i, 0)$, kde $1 \leq i \leq l$.

Kdyby byla v předchozím kroku první hlava na pozici $l+1$, tedy hlava by četla znak $\triangleleft$, použilo by se šesté pravidlo sekce zrcadlových pravidel a na pozici 0 být nemohla, protože do stavu $c_l$ se automat dostane jedině za současného posunu hlavy vpravo. Následně se musí první pravidlo použít $i$krát, načež se automat dostane do konfigurace $(s_{\mathrm{l}t_1(i)}, 0, i)$.

### Načtení pozice

Načítání pozice $l + 1 - i$ probíhá postupným posunem obou hlav vpravo podle prvního pravidla, zastaví se až když druhá hlava dosáhne zarážky. Druhá hlava musí dosáhnout pravé zarážky dříve než první hlava, protože na začátku této sekce je pozice první hlavy menší než pozice druhé hlavy. Z toho plyne, že konfigurace po aplikaci maximálního počtu opakování prvního pravidla musí být $(s_{\mathrm{l}t_1(i)}, l + 1 - i, l + 1)$.

Pokud platí $t_1(l+1-i) = t_1(i)$, uplatní se druhé pravidlo a výpočet pokračuje v konfiguraci $(c_r, l - i, l + 1)$. Pokud vztah neplatí, nejedná se o palindrom a neexistuje pravidlo, které by mohl automat použít, vstup tedy neakceptuje.

### Uložení symbolu

První pravidlo sekce uložení aktuálního symbolu se uplatní, pokud je pozice první hlavy mezi zarážkami. Na pravé zarážce být vzhledem k předchozímu posunu vlevo (jedinému možnému předcházejícímu pravidlu) nemůže a pokud je na levé zarážce, automat akceptuje. K druhé možnosti se vrátíme později.

Po aplikaci prvního pravidla si automat ve stavu pamatuje aktuální symbol a dostává se tím do konfigurace $(s_{\mathrm{r}t_1(l-1)}, l - i, l + 1)$.

### Zrcadlová pravidla

Sekce zrcadlových pravidel lze popsat analogicky k předchozímu. Automat projde konfiguracemi $(s_{\mathrm{r}t_1(l-1)}, l+1, l-i)$, $(l_{\mathrm{l}t_1(l-1)}, i+1, 0)$, kde proběhne kontrola $t_1(i+1) = t_1(l-i)$ a v případě úspěchu se dostaneme do konfigurace $(c_{\mathrm{l}}, i+2, 0)$.

V průběhu jedné iterace celého cyklu byly zkontrolovány protějšky symbolů na pozicích $i$ a $i + 1$. Další opakování by kontrolovalo $i + 2$ a $i + 3$ atd. Zbývá prozkoumat poslední dvě možné konfigurace, a to $(c_{\mathrm{r}}, 0, l+1)$ a $(c_{\mathrm{l}}, l+1, 0)$. Pokud se automat během cyklu dostane do jedné z těchto konfigurací (které byly dříve

zmíněny), znamená to, že již byly všechny dvojice úspěšně porovnány a slovo může být akceptováno. Pro každý palindrom se automat musí do jedné z těchto konfigurací dostat a pokud některá porovnávaná dvojice symbolů není stejná, neexistuje další pravidlo, které by mohlo být provedeno. Proto jsou rozpoznávány právě palindromy. □

**Lemma 31**

*Pro každý jazyk $L_{ipal}$ ($i \geq 3$) existuje asynchronní it-0-dLA $\mathcal{A}_i$ akceptující $L_{ipal}$.*

*Důkaz*

Protože přímá definice funkce $\delta$ pro tyto automaty by byla příliš dlouhá a nepřehledná, vytvoříme si abstrakci nad určitými vzory $\delta$ pravidel v podobě slovně popsaných instrukcí a ty následně použijeme v pseudokódu. Z výsledného pseudokódu bude patrné, že odpovídající $\delta$ funkce lze vytvořit.

V následujících šablonách budou použity dva speciální symboly, symbol $\langle\rangle$ u stavů bude značit, že stav má kromě uvedeného jména ve skutečnosti i část reprezentující *konečný* zásobník volání funkcí a jednotlivých instrukcí a také paměť, pomocí které v algoritmu ukládá vždy jeden páskový symbol, nebo jednu pravdivostní hodnotu. Jelikož jsou obě dvě položky, paměť i zásobník, konečné, je možné vytvořit odpovídající stavy tak, aby množina $Q$ stavů automatu byla také konečná. Dále bude na konci každé sekvence pravidel automat přecházet do stavu *next*. Zde se nejedná o skutečný stav, ale pouze o zástupný symbol, který je ve skutečnosti dosazován podle stavu, zásobníku a paměti. Z pseudokódu by mělo být jasné, jak se tyto stavy *next* dosadí.

Proměnnými $h_i, 1 \leq i \leq m$ značíme $i$tou hlavu $m$ páskového automatu. Připomeňme, že zde pouze $h_1$ dokáže číst vstup, ostatní hlavy vidí na svých páskách pouze prázdné znaky, nebo zarážky. Budeme také používat funkci $\mathrm{v}(h_i)$, která udává aktuální symbol čtený hlavou $h_i$.

Opět užijeme úspornější verzi konfigurací.

1. `Move` $h_i$ `by value` $n$ `right`

$$\forall 1 \leq i \leq m\ \forall n \in \mathbb{N}, n \neq 1,\ d_1 = \cdots = d_{i-1} = d_{i+1} = \cdots = d_m = 0,\ d_i = 1$$
$$\delta(\mathrm{m}i\mathrm{bv}n\mathrm{r}\langle\rangle, x) = (\mathrm{m}i\mathrm{bv}(n-1)\mathrm{r}\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$$\forall 1 \leq i \leq m,\ d_1 = \cdots = d_{i-1} = d_{i+1} = \cdots = d_m = 0,\ d_i = 1\ \forall x = (c_1, \ldots c_m), c_i \neq \triangleleft$$
$$\delta(\mathrm{m}i\mathrm{bv1r}\langle\rangle, x) = (next, x, (d_1, d_2, \ldots d_m))$$

V šabloně je ošetřeno, aby neexistovalo pravidlo, které by posunulo některou hlavu za zarážku. Takto pravidla neodporují definici. Není však zaručeno, že někdo nepoužije např. instrukci `Move` $h_i$ `by value 5 right`

když bude hlava $h_i$ na předposlední pozici. Výpočet by skončil neakceptováním slova a v pseudokódu bude třeba ukázat, že k těmto situacím nedochází. Obecně instrukce přejde z konfigurace $(q, p_1, p_2, \ldots, p_i, \ldots)$ do $(q, p_1, p_2, \ldots, p_i + n, \ldots)$.

U tohoto pravidla je také důležité, aby byla hodnota $n$ v kódu konstantní, či omezena konstantou, protože toto pravidlo generuje nejméně $n$ pravidel automatu.

2. `Move` $h_i$ `by value` $n$ `left`
   Analogicky k předchozímu. Přejde z konfigurace $(q, p_1, p_2, \ldots, p_i, \ldots)$ do $(q, p_1, p_2, \ldots, p_i - n, \ldots)$.

3. `Exchange` $(h_i, h_j)$

$\forall i, j,\ 1 \leq i \leq m\ 1 \leq j \leq m\ d_1 = d_2 = \cdots d_m = 0,\ x = (x_1, x_2, \ldots, x_m),$ $x_i = \triangleright$

$$\delta(\mathrm{ex}(i,j)\langle\rangle, x) = (\mathrm{ex}(i,j)_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall i, j,\ 1 \leq i \leq m\ 1 \leq j \leq m\ \forall k, i \neq k \neq j \colon d_k = 0,\ d_i = 1,\ d_j = -1 x = (x_1, x_2, \ldots, x_m),\ x_j \neq \triangleright$

$$\delta(\mathrm{ex}(i,j)_1\langle\rangle, x) = (\mathrm{ex}(i,j)_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall i, j,\ 1 \leq i \leq m\ 1 \leq j \leq m\ d_1 = d_2 = \cdots d_m = 0,\ x = (x_1, x_2, \ldots, x_m),$ $x_j = \triangleright$

$$\delta(\mathrm{ex}(i,j)_1\langle\rangle, x) = (next, x, (d_1, d_2, \ldots d_m))$$

$\forall i, j,\ 1 \leq i \leq m\ 1 \leq j \leq m\ d_1 = d_2 = \cdots d_m = 0,\ x = (x_1, x_2, \ldots, x_m),\ x_i \neq \triangleright, x_j = \triangleright$

$$\delta(\mathrm{ex}(i,j)\langle\rangle, x) = (\mathrm{ex}(i,j)_2\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall i, j,\ 1 \leq i \leq m\ 1 \leq j \leq m\ \forall k, i \neq k \neq j \colon d_k = 0,\ d_i = -1,\ d_j = 1 x = (x_1, x_2, \ldots, x_m),\ x_i \neq \triangleright$

$$\delta(\mathrm{ex}(i,j)_2\langle\rangle, x) = (\mathrm{ex}(i,j)_2\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall i, j,\ 1 \leq i \leq m\ 1 \leq j \leq m\ d_1 = d_2 = \cdots d_m = 0,\ x = (x_1, x_2, \ldots, x_m),$ $x_i = \triangleright$

$$\delta(\mathrm{ex}(i,j)_2\langle\rangle, x) = (next, x, (d_1, d_2, \ldots d_m))$$

Automat přejde z konfigurace $(q, p_1, p_2, \ldots, p_i, \ldots, p_j, \ldots)$ do $(next, p_1, p_2, \ldots, p_j, \ldots, p_i, \ldots)$.

4. `Move` $h_1$ `by difference` $(h_2, h_i)$ `right`

$$\forall i, 3 \leq i \leq m \; d_1 = d_2 = \cdots d_m = 0, \; x = (x_1, x_2, \ldots, x_m), \; x_2 = \triangleright$$
$$\delta(\mathrm{m1bd}(2,i)\mathrm{r}\langle\rangle, x) = (\mathrm{m1bd}(2,i)\mathrm{r}_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$$\forall i, 3 \leq i \leq m \; d_1 = d_2 = 1, d_3 = \cdots = d_{i-1} = d_{i+1} = \cdots = d_m = 0, \; d_i = -1, \; x = (x_1, x_2, \ldots, x_m), \; x_1 \neq \triangleleft, x_2 \neq \triangleleft, x_i \neq \triangleright$$
$$\delta(\mathrm{m1bd}(2,i)\mathrm{r}_1\langle\rangle, x) = (\mathrm{m1bd}(2,i)\mathrm{r}_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$$\forall i, 3 \leq i \leq m \; d_1 = d_2 = \cdots d_m = 0, \; x = (x_1, x_2, \ldots, x_m), \; x_1 \neq \triangleleft, x_2 \neq \triangleleft, x_i = \triangleright$$
$$\delta(\mathrm{m1bd}(2,i)\mathrm{r}_1\langle\rangle, x) = (\mathrm{m1bd}(2,i)\mathrm{r}_2\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$$\forall i, 3 \leq i \leq m \; d_1 = 0, d_2 = -1, d_3 = \cdots = d_{i-1} = d_{i+1} = \cdots = d_m = 0, \; d_i = 1, \; x = (x_1, x_2, \ldots, x_m), \; x_2 \neq \triangleright, x_i \neq \triangleleft$$
$$\delta(\mathrm{m1bd}(2,i)\mathrm{r}_2\langle\rangle, x) = (\mathrm{m1bd}(2,i)\mathrm{r}_2\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$$\forall i, 3 \leq i \leq m \; d_1 = d_2 = \cdots d_m = 0, \; x = (x_1, x_2, \ldots, x_m), \; x_2 = \triangleright$$
$$\delta(\mathrm{m1bd}(2,i)\mathrm{r}_2\langle\rangle, x) = (next, x, (d_1, d_2, \ldots d_m))$$

Stejně jako v první instrukci i zde bude třeba v pseudokódu pohlídat, jestli není rozdíl pozic hlav $h_2$ a $h_i$ větší než vzdálenost $h_1$ od pravé zarážky. Automat přejde z konfigurace $(q, p_1, 0, \ldots, p_i, \ldots)$ do $(next, p_1{+}p_i, 0, \ldots, p_i, \ldots)$, kde *next* záleží na umístění instrukce v pseudokódu.

V kódu budeme tuto instrukci potřebovat i v případě, kdy $x_2 \neq \triangleright$, ale $x_i = \triangleright$. V takovém případě budeme stejný zápis instrukce brát za zkratku pro `Exchange` $(h_i, h_j)$ $\rightarrow$ `Move` $h_1$ `by difference` $(h_2, h_i)$ `right` $\rightarrow$ `Exchange` $(h_i, h_j)$.

5. `Move` $h_1$ `by difference` $(h_2, h_i)$ `left`
Funkce je analogická k předchozí funkci pro pohyb vpravo. Automat přejde z konfigurace $(q, p_1, 0, \ldots, p_i, \ldots)$ do $(next, p_1 - p_i, 0, \ldots, p_i, \ldots)$.

6. `Move` $h_i$ `rightmost/leftmost [and move` $h_j$ `right/left]`

Část instrukce v hranatých závorkách je volitelná. Tuto instrukci není třeba zvlášť představovat, jelikož je podobná instrukci `Move` $h_1$ `by difference`. Rozdíl je v tom, že zde není třeba třetí hlavy, která by resetovala pozici druhé hlavy. Automat přejde z konfigurace $(q, p_1, p_2, \ldots, p_i, \ldots, p_j, \ldots)$ do $(next, p_1, p_2, \ldots, (l+1)/0, \ldots, p_j[\pm p_i], \ldots)$. Verze, která pouze nastaví pozici hlavy nejvíce vlevo/vpravo je ještě jednodušší.

7. $=(h_i, h_j)$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_2 = \triangleright d_1 = d_2 = \cdots d_m = 0$

$$\delta(i{=}j\langle\rangle, x) = (i{=}j_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_i \neq \triangleright \neq x_j,\ \forall k, i \neq k \neq j, k \neq 2\colon d_k = 0,\ d_i, d_j = -1,\ d_2 = 1$

$$\delta(i{=}j_1\langle\rangle, x) = (i{=}j_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_i = \triangleright \not\Leftrightarrow x_j = \triangleright,\ d_1 = d_2 = \cdots d_m = 0$

$$\delta(i{=}j_1\langle\rangle, x) = (i{=}j_{\text{false}}\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_i = \triangleright = \triangleright,\ d_1 = d_2 = \cdots d_m = 0$

$$\delta(i{=}j_1\langle\rangle, x) = (i{=}j_{\text{true}}\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_2 \neq \triangleright \neq x_j,\ \forall k, i \neq k \neq j, k \neq 2\colon d_k = 0,\ d_i, d_j = 1,\ d_2 = -1$

$$\delta(i{=}j_{\text{true2}}\langle\rangle, x) = (i{=}j_{\text{true2}}\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_2 \neq \triangleright \neq x_j,\ \forall k, i \neq k \neq j, k \neq 2\colon d_k = 0,\ d_i, d_j = 1,\ d_2 = -1$

$$\delta(i{=}j_{\text{false2}}\langle\rangle, x) = (i{=}j_{\text{false2}}\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_2 = \triangleright \neq x_j,\ \forall k, i \neq k \neq j, k \neq 2\colon d_k = 0,\ d_i, d_j = 1,\ d_2 = -1$

$$\delta(i{=}j_{\text{true2}}\langle\rangle, x) = (next, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_2 = \triangleright \neq x_j,\ \forall k, i \neq k \neq j, k \neq 2\colon d_k = 0,\ d_i, d_j = 1,\ d_2 = -1$

$$\delta(i{=}j_{\text{false2}}\langle\rangle, x) = (next, x, (d_1, d_2, \ldots d_m))$$

V této instrukci můžeme vidět dva stavy *next*, každý z nich reprezentuje obecně jiný stav. V pseudokódu tomu odpovídají podmíněné instrukce. Konfigurace je po ukončení instrukce až na stav stejná jako před vykonáním instrukce.

8. `Pin` $h_i$ `to` $h_j$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_2 = \triangleright d_1 = d_2 = \cdots d_m = 0$

$$\delta(\mathrm{cit}j\langle\rangle, x) = (\mathrm{cit}j_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_i \neq \triangleright \wedge x_j \neq \triangleright,\ \forall k, i \neq k \neq j, k \neq 2\colon d_k = 0,\ d_i = d_j = -1,\ d_2 = 1$

$$\delta(\mathrm{cit}j_1\langle\rangle, x) = (\mathrm{cit}j_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_i = \triangleright \wedge x_j \neq \triangleright,\ \forall k, i \neq k \neq j, k \neq 2\colon d_k = 0,\ d_i = 0, d_j = -1,\ d_2 = 1$

$$\delta(\mathrm{cit}j_1\langle\rangle, x) = (\mathrm{cit}j_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_i \neq \triangleright \wedge x_j = \triangleright,\ \forall k, i \neq k \neq j, k \neq 2\colon d_k = 0,\ d_i = -1, d_j = 0,\ d_2 = 0$

$$\delta(\mathrm{cit}j_1\langle\rangle, x) = (\mathrm{cit}j_1\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_i = \triangleright \wedge x_j = \triangleright,\ d_1 = d_2 = \cdots d_m = 0$

$$\delta(\mathrm{cit}j_1\langle\rangle, x) = (\mathrm{cit}j_2\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_2 \neq \triangleright, x_j \neq \triangleright \neq x_i,\ \forall k, i \neq k \neq j, k \neq 2\colon d_k = 0,\ d_i = d_j = 1,\ d_2 = -1$

$$\delta(\mathrm{cit}j_2\langle\rangle, x) = (\mathrm{cit}j_2\langle\rangle, x, (d_1, d_2, \ldots d_m))$$

$\forall 1 \leq i \leq m\ \forall 1 \leq i \leq m\ i \neq 2 \neq j,\ x = (x_1, x_2, \ldots, x_m),\ x_2 = \triangleright,\ d_1 = d_2 = \cdots d_m = 0$

$$\delta(\mathrm{cit}j_2\langle\rangle, x) = (next, x, (d_1, d_2, \ldots d_m))$$

Tato instrukce připne hlavu $h_i$ na pozici hlavy $h_j$, přitom předpokládáme, že $h_2$ je na pozici 0. Konfigurace se změní z $(q, p_1, 0, \ldots, p_i, \ldots, p_j, \ldots)$ do $(next, p_1, 0, \ldots, p_j, \ldots, p_j, \ldots)$.

9. `Store(`$v$`)`, `Peek()`

Tyto funkce ukládají (případně přepisují) a načítají hodnotu uloženou ve stavu v části symbolicky značené $\langle\rangle$. V této části se ještě nachází „zásobník volání“ funkcí a instrukcí. S tím ale pracujeme pouze implicitně, a to

samotným poskládáním instrukcí v pseudokódu. Informace nutné k určení dalšího stavu (*next*) jsou získány právě z $\langle\rangle$.

Nyní máme dostatečnou abstrakci nad pravidly automatu. V následujícím kódu si ukážeme popis $m$páskového automatu $\mathcal{A}_m$ pro nějaké $m \in \mathbb{N}, m > 2$, rozpoznávajícího $L_{(m-1)\text{pal}}$.

```
1  RightReflection(l, r)
2     if l > 2
3        Move h1 by difference(h2, hl) left
4     Exchange(h1, h2)
5     if r ≤ m
6        Exchange(h1, hr)
7        Move h1 by difference(h2, hr) left
8     else
9        Move h1 rightmost
10       Move h2 leftmost and move h1 left
```

Funkce `RightReflection(`$l$`,`$r$`)` předpokládá, že hlava $h_1$ je na pozici vymezené hlavami $h_l$ a $h_r$ (přitom může být stejná jako $h_l$, protože $h_l$ značí první znak intervalu, zatímco $h_r$ značí první znak za intervalem. Výjimkou jsou případy $l = 2$, který značí, že aktuálně zkoumaný interval začíná na začátku vstupu (je omezen pouze $h_r$), a $r = m$, který značí, že interval končí na konci vstupu (je omezen pouze $h_l$).

Označme délku tohoto intervalu $w_{lr}$ jako $l_{lr}$ a pozici znaku čteného hlavou $h_1$ ve $w_{lr}$ označme $i$. Počáteční konfigurací je $(q_1\langle\rangle, p_1 = p_{lr}+i, 0, \ldots, p_l, p_r, \ldots)$. Pokud platí $l > 2$, přejde automat po instrukci na řádku 3 do konfigurace $(q_2\langle\rangle, p_1 = i, 0, \ldots, p_l, \ldots, p_r, \ldots)$. Je-li $l = 2$, pak $p_l = 0 \Rightarrow p_1 = i$, v obou případech proto bude automat v konfiguraci $(q_3\langle\rangle, p_1 = i, 0, \ldots, p_l, \ldots, p_r, \ldots)$ na řádku 4. Následuje konfigurace $(q_4\langle\rangle, 0, i, \ldots, p_l, \ldots, p_r, \ldots)$. Pokud platí $r \leq m$, vykoná se dále šestý řádek uvozující automat do konfigurace $(q_5\langle\rangle, p_r, i, \ldots, p_l, \ldots, 0, \ldots)$. Protože $i$ je index v nějakém slově na pozici hlavy $h_1$, musí platit $i \leq p_1$ a podle předpokladu $p_1 < p_r$, což vede k $i < p_r$. Proto lze provést instrukce na sedmém řádku vedoucí na konfiguraci $(q_5\langle\rangle, p_r - i, i, \ldots, p_l, \ldots, 0, \ldots)$. Pokud $r > m$, změní se na devátém řádku konfigurace z $(q_6\langle\rangle, 0, i, \ldots, p_l, \ldots, p_r, \ldots)$ na $(q_7\langle\rangle, l + 1, i, \ldots, p_l, \ldots, p_r, \ldots)$. A protože $l + 1$ je nejpravější pozice, určitě lze hlavou $h_1$ hýbat směrem vlevo současně s hlavou $h_2$. Tím se dostane automat do konfigurace $(q_8\langle\rangle, l + 1 - i, 0, \ldots, p_l, \ldots, p_r, \ldots)$. V obou těchto případech se hlava $h_1$ dostane na pozici zrcadlově převrácenou vzhledem ke středu slova $w_b$.

V předchozím popisu jsou stavy indexovány pouze aby bylo zřejmé, že se nejedná o jeden stejný stav. Ve skutečnosti je popis celého zásobníku volání funkcí i konkrétní řádek instrukce ve funkci zakódován v části označené $\langle\rangle$.

Splnění podmínky $p_1 \leq p_l < p_r$, stejně jako $p_2 = 0$, je zodpovědností volající funkce.

```
1  LeftReflection(l, r)
2     if r ≤ m
```

```
3        Move h1 by difference(h2, hr) right
4        Exchange(h1, hr)
5     else
6        Move h1 rightmost and move h2 right
7        Move h1 leftmost
8     Exchange(h1, h2)
9     if l > 2
10       Move h1 by difference(h2, hl) right
```

Funkce `LeftReflection(`$l$`,`$r$`)` popisuje postup opačný k funkci `RightReflection(`$l$`,`$r$`)`. Automat se proto při použití ihned po `RightReflection(`$l$`,`$r$`)` opět dostane do výchozí $(q_9\langle\rangle, p_1 = p_l + i, 0, \ldots, p_l, \ldots, p_r, \ldots)$.

```
1  CheckPalindrome(l, r)
2     if l > 2
3        Pin h1 to hl
4        Move hl by value 1 left
5     else
6        Move h1 leftmost
7        Move h1 by value 1 right
8     while true
9        Store(v(h1))
10       RightReflection(l, r)
11       Store(compare(Pop(),v(h1)))
12       if Pop() = false
13          Store(false)
14          break
15       LeftReflection(l, r)
16       Move h1 by value 1 right
17       if v(h1) = ◁ or (l < m and =(h1,hr))
18          Store(true)
19          break
20    if l > 2
21       Move hb by value 1 right
```

Tato funkce předpokládá $p_r > p_l > 0$ pro $2 < l < m$ a $p_m \leq l$.

Představme si výchozí konfiguraci $(q_1\langle\rangle, x, 0, \ldots, p_l, \ldots, p_r, \ldots)$. Všimněme si, že $x$ může být libovolné. Funkce `Store` označuje uložení hodnoty „do stavu“ automatu. Syntaxe `if Funkce()` předpokládá, že `Funkce()` vrací hodnotu pomocí `Store` a označuje její vyjmutí ze stavu.

Na řádcích 1-7 dojde k inicializaci hlavy $h_1$ pro kontrolu slova $w_{lr}$. Konkrétně se hlava nastaví na první znak $w_{lr}$. Pokud $l > 2$, je první symbol na pozici hlavy $h_l$, proto se třetím a čtvrtým řádkem přechází do konfigurace $(q_2\langle\rangle, p_l, 0, \ldots, p_l - 1, \ldots, p_r, \ldots)$. Pohyb hlavy $h_l$ doleva lze provést, protože podle předpokladu je $p_l > 0$ pro $l > 2$. V případě $l = 2$ se $h_1$ nastaví na 1, což nám dává $(q_3\langle\rangle, 1, 0, \ldots, p_r, \ldots)$. Touto cestou sjednotíme způsob kontroly intervalu na palindrom v případech $l = 2$ a $l > 2$. Vždy bude zarážka před prvním symbolem intervalu (ať už $\triangleright$, nebo virtuální zarážka tvořená hlavou $h_l$).

Z předchozích analýz vyplývá, že kód na řádcích 9-12 kromě stavu nezmění

konfiguraci. Automat si zapamatuje symbol vstupu, což je v kódu naznačeno jako $\mathrm{v}(h_1)$, a porovná jej se symbolem na zrcadlové pozici, bylo-li porovnání úspěšné, vrátí se zpět přičemž si ve stavu pamatuje výsledek porovnání. Pokud porovnání úspěšné nebylo, cyklus končí. Výsledek porovnání je stále uložen ve stavu a je později použit jako výsledek celé kontroly na palindrom. Můžeme si všimnout, že zde na rozdíl od „pozitivního“ ukončení kontroly na řádku 16 hlava $h_1$ může mít různou pozici. Tato pozice však není nijak důležitá, jelikož budoucí zavolání `CheckPalindrome(`$l$`,`$r$`)` si pozici upraví a mezi těmito voláními není $h_1$ důležitá.

Po každém úspěšném porovnání se na řádku 15 zvětší pozice $p_1$. Zde může být zavádějící použití názvu `RightReflection(`$l$`,`$r$`)`, protože od druhé poloviny slova se obraz symbolu nachází nalevo, nicméně cyklus projde při úspěšných porovnáních celý interval a zastaví se až při splnění podmínky na řádku 16, tedy při dosažení pravé zarážky hlavou $h_1$. Použití funkce `=(`$h_i$`,`$h_j$`)` v tomto případě je možné, protože pozice $p_2 = 0$.

Po ukončení cyklu si stav pamatuje „návratovou hodnotu“ a pokud $l > 2$ je navíc potřeba posunout hlavu $h_l$ doprava, čímž se vrátíme kromě stavu a $p_1$ do původní konfigurace.

Dále budeme potřebovat dvě jednoduché funkce `IsValid(`$i$`)` a `Reset(`$i$`,`$j$`)`. Funkce `IsValid(`$i$`)` zjistí o hlavě $h_i$, jestli je na validní pozici, tedy jestli není na nejpravějším políčku pásky, nebo na stejné pozici, jako jedna z hlav s vyšším indexem.

```
1  IsValid(i)
2     for j ← i + 1 to m
3        if =(h_j, h_i)
4           Store(false)
5           return
6     if v(h_i) = ◁
7        Store(false)
8        return
9     Store(true)
```

Funkce `Reset(`$i$`,`$j$`)` potom nastaví všechny hlavy $h_3, \ldots, h_i$ na pozice $h_j + 1, \ldots, h_j + i - 2$, přitom vrátí false pokud by některá z těchto pozic nebyla validní ve smyslu předchozí funkce.

```
1  Reset(i, j)
2     if i = 2
3        Store(true)
4        return
5     for k ← 3 to i
6        Pin h_k to h_j
7        if j = 2
8           Move h_k by value 1 right
9        Move h_k by value k − 2 right
10       if (!IsValid(k))
11          Store(false)
12          return
13    Store(true)
```

```
1  p←NewPointer(m)
2
3  CheckPalindromes(i)
4     if i = 3
5        CheckPalindrome(ToHeads(p))
6        return
7     while(IsValid(i))
8        if CheckPalindromes(i − 1)
9           ResetPointer(i − 2)
10          p[i − 2]← R
11          Reset(i − 1, i)
12          if (CheckPalindromes(i − 1))
13             Store(true)
14             return
15       Move h_i by value 1 right
16       ResetPointer(i − 2)
17       Reset(i − 1, FirstR(p) + 2)
18    Store(false)
```

Pomocí funkce `CheckPalindrome(l,r)` a pomocných funkcí teď můžeme zkontrolovat, zda se vstup $x$ skládá ze zřetězení $2^{m-2}$ palindromů. Pro kontrolu každého palindromu budeme používat hlavu $h_1$, $h_2$ a dvě zarážky, tvořené buď hlavami, nebo koncem pásky. Díky chytrému způsobu průchodu všech možností oddělení palindromů nepotřebujeme hlavu pro každý oddělovač v každém okamžiku výpočtu. Hlavy $h_3 \dots h_m$ budou tyto oddělení procházet postupně tak, aby nebylo potřeba pamatovat si pozici, kterou právě opustili. K tomu využijeme následujícího poznatku: pokud je $x_1$ prefix slova $x = x_1x_2$ obsahující zřetězení $n_1$ palindromů a testujeme, zda je $x$ zřetězením $n_1 + n_2$ palindromů, pak si není potřeba pamatovat poslední odzkoušené rozdělení slova $x_1$ na $n_1$ palindromů pro případné pokračování testů následujících rozdělení ve zvoleném pořadí. Důvodem je to, že pokud v $x_2$ existuje rozdělení na $n_2$ palindromů, známe výsledek celého testu a pokud neexistuje, není potřeba prohledávat ani žádné jiné rozdělení $x_1$ na $n_1$ palindromů.

Pseudokód popisuje, jak automat použije hlavy jako rozdělovače pro hledané rozdělení a to tak, že $h_3$ rozděluje vstup $x$ na poloviny. Pokud je k dispozici hlava $h_4$, dokáže každou tuto polovinu postupně projít rekurzivním rozdělením na poloviny, a to díky výše zmíněnému pozorování. Tedy čtyřpáskový automat zvládne otestovat, jestli je vstup zřetězením čtyř palindromů, pětipáskový automat osmi a tak dál.

Pro potřeby funkce také zavedeme globální proměnnou $p \in \{\mathrm{L}, \mathrm{R}\}^{m-2}$, která se při inicializaci pomocí `NewPointer(m)` nastaví na řetězec $\mathrm{L}^{m-2}$. Na tento řetězec se můžeme dívat jako na binární číslo popisující index podřetězce mezi oddělovači (zde myslíme i „virtuální“ oddělovače, tedy ne nutně přímo pozice hlav automatu), který je aktuálně testován. Pro jednoduchost budeme jednotlivé pozice tohoto řetězce indexovat zezadu od 1, tedy přiřazení `p[2]←R` by změnilo LLLLL na LLLRL. `ResetPointer(i)` nastaví posledních $i$ míst řetězce na „nuly“, tedy P.

Funkce je rekurzivní, nicméně hloubka rekurze je vždy v $\mathcal{O}(m)$, tedy konstantní vzhledem k délce vstupu, ve skutečné reprezentaci tohoto pseudokódu pomocí delta funkce lze tedy paměť potřebnou pro řízení rekurze realizovat pomocí stavu automatu. Důležitá je také funkce `ToHeads(`$p$`)`, převádějící ukazatel ve formě řetězce do dvojice indexů, určujících hlavy automatu, které vymezují aktuálně zkoumaný podřetězec vstupu. Tato funkce se pro různá $m$ liší, lze však sestavit velice jednoduše a napevno uložit do delta funkce automatu.

Ukažme si, jak sestrojit `ToHeads(`$p$`)` pro $m = 3$ a jak ji sestavit pro $m - 1$ z `ToHeads(`$p$`)` pro $m$. Připomeňme, že pointer L zde chápeme jako číslo 0 a R jako číslo 1. Pak

$$\texttt{ToHeads}_3(i) = (2, 3)$$

$$\texttt{ToHeads}_3(i) = (3, 4)$$

a pro $i \in \mathbb{N}_0,\ i < 2^{m-2}$ :

$$\texttt{ToHeads}_m(i) = \begin{cases} \texttt{ToHeads}_{m-1}(i) & i < 2^{m-3} \\ (\texttt{ToHeads}_{m-1}(2^{m-3} - 1 - i))^{\mathrm{R}} & 2^{m-3} \leq i < 2^{m-2} - 1 \\ (\texttt{ToHeads}_{m-1}(2^{m-3} - 1 - i)[2], \quad m + 1) & i = 2^{m-2} - 1 \end{cases}$$

Následující tabulka ukazuje hodnoty $\texttt{ToHeads}_m(i)$ pro některé $m$.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $m = 3$ | (2,3) | (3,4) | | | | | | |
| $m = 4$ | (2,3) | (3,4) | (4,3) | (3,5) | | | | |
| $m = 5$ | (2,3) | (3,4) | (4,3) | (3,5) | (5,3) | (3,4) | (4,3) | (3,6) |

Z výše uvedených úvah a pozorování vlastností zřetězení palindromů vyplývá, že funkce pro správné počáteční rozložení hlav projde všechny kombinace rozdělení slova $x$ nutné k určení, zda se jedná o zřetězení daného počtu palindromů. Je třeba myslet i na to, že vstup může být kratší než je třeba ke smysluplnému rozložení hlav, proto použijeme už implementovanou funkci `Reset`, která tento případ zohledňuje.

```
1 CheckPalindromes()
2    Move h2 by value 1 left
3    return Reset(m, 2) and CheckPalindromes(m)
```

□

**Věta 32**

*Pro každý jazyk $L_{ipal}$ $(2 \leq i)$ existuje asynchronní it-0-dLA $\mathcal{A}_i$ akceptující $L_{ipal}$.*

*Důkaz*

Vyplývá přímo z předchozích dvou lemmat. □

### 4.2.2 Konstrukce oddělovačů

Předchozí konstrukci 0-limitovaných vícepáskových automatů pro rozpoznávání konkatenace palindromů ve vstupu nazvěme *konstrukce oddělovačů*. Nazveme tak každou konstrukci 0-limitovaných automatů, která si informace ze vstupní pásky ukládá pomocí stavu a ostatní pásky používá pouze pro uchování pozic testovaného rozdělení a systematický průchod všech takových rozdělení.

Z předchozích vět a příkladů jsme také mohli vypozorovat, že automat všechna nezbytně nutná rozdělení prochází v takovém pořadí, že je schopný operovat s více *oddělovači*, než má k dispozici hlav. Místa, která slouží jako oddělovač pro právě testované rozdělení nezávisle na tom, jestli se na jejich pozici nachází hlava, proto nazývejme *virtuální oddělovač*. *Oddělovačem* potom místo, které je virtuální oddělovač, a kde se zároveň hlava nachází.

Mohlo by nás napadnout, zda lze tato myšlenka využít k sestavení automatu, který daný počet palindromů dokáže rozpoznávat s menším počtem pásek.

Domněnka 33

Neexistuje žádný asynchronní $(m-1)$t-0-dLA $\mathcal{A}_{m-1}$ , který pomocí konstrukce oddělovačů akceptuje $L_{m\text{pal}}$ pro $2 \leq m$.

Domněnku opodstatníme v několika krocích. Budeme předpokládat, že existuje automat, který zvládá rozpoznávat stejný počet palindromů, ale s menším počtem pásek. Dále si ukážeme, že potom některá z hlav spravuje právě jeden virtuální oddělovač. V posledním kroku ukážeme, že potom po odstranění této pásky rozpoznává automat nejméně polovinu palindromů z původního počtu a díky tomu dojdeme indukcí ke sporu.

Předpokládejme tedy, že existuje nějaký $s$-páskový nula limitovaný automat $\mathcal{A}$ přijímající jazyk $L_{j\text{pal}}$, kde $j > s$.

Dále si pomocí $\mathrm{d}(h_i)$ označíme počet virtuálních oddělovačů, které hlava $i$ spravuje. To může být nula, jedna i víc, dokonce může některé oddělovače spravovat v různých časech více hlav. Pokud by se stalo, že některé hlavy si počet spravovaných oddělovačů v průběhu vzájemně mění, lze sestrojit automat, který toto nedělá. Umíme totiž vzájemně vyměnit pozici hlav.

Nyní si zaveďme číslo

$$\mathrm{d}_{\min} = \max(1, \min_{i \in \{1,\ldots,s\}} (\mathrm{d}(h_i))).$$

Dále zaveďme index hlavy spravující tento počet virtuálních oddělovačů, označme jej $i_{\min}$.

Předpokládejme, že hlava $h_{i_{\min}}$ spravuje více než jeden virtuální oddělovač, konkrétně oddělovače $o_1$ a $o_2$. Protože je spravuje pouze jedna fyzická hlava a ta nemůže být v jedné chvíli na dvou různých místech, je zřejmé, že $o_1$ a $o_2$ nemohou být sousední oddělovače. Ke kontrole palindromu mezi dvěma pozicemi potřebujeme, aby se na nich nacházely hlavy, případně, aby tyto pozice byly na konci pásky.

Pokud ale nejsou tyto oddělovače sousední, musí mezi nimi být nějaký oddělovač $o'_1$ spravovaný jinou hlavou $h'$. Jenže $\mathrm{d}_{\min} > 1$, takže potřebujeme ještě minimálně jeden virtuální oddělovač $o'_2$ spravovaný $h'$. Tento oddělovač může být mezi $o_1$ a $o'_1$, pak ale máme stejný problém se sousedními virtuálními oddělovači, nebo mezi $o'_1$ a $o_2$ (analogický případ k předchozímu), nebo před $o_1$, či za $o_2$. V obou případech se musí jedna z hlav při kontrole rozdělení přesunout přes tu druhou, přičemž zapomene původní umístění (ve stavu jej uložit nelze a ostatní hlavy drží pozice jiných oddělovačů, zde intuitivně předpokládáme, že ostatní hlavy nemohou své pozice opustit).

Například v případě virtuálních oddělovačů v pozicích $o_1 > o'_1 > o_2 > o'_2$ bychom zkontrolovali jestli je řetězec na pozicích $o_1, \ldots, o'_1$ palindrom, pokud ano, přesunuli bychom hlavu $h_{i_{\min}}$ na virtuální oddělovač $o_2$. To ještě není problém pokud bychom již nepotřebovali polohu $o_1$, pokud ale i řetězec na pozicích $o'_1, \ldots, o_2$ bude palindrom, nemůžeme oddělat hlavu $h_{i_{\min}}$, protože pokud by řetězec napravo od $o_2$ nebyl konkatenací odpovídajícího počtu palindromů, nelze již najít předešlé virtuální oddělovače a pokračovat ve smysluplném deterministickém průchodu všech možných rozdělení. Předpoklad tedy neplatí a $\mathrm{d}_{\min} = 1$.

Všimněme si, že v případě konstrukce z minulé podkapitoly bylo $\mathrm{d}_{\min}$ právě 1, proto nebyl se „zapomínáním“ pozic problém.

Jestliže víme, že hlava $h_{i_{\min}}$ spravuje pouze oddělovač $o_x$ a ten rozděluje vstup na $x$ palindromů a $2^{j-2} - x$ palindromů. Ať je $x$ jakékoliv, vždy můžeme bez $j$-té pásky vytvořit automat, který rozpoznává větší počet palindromů z těchto dvou. Náš automat tedy bude rozpoznávat nejméně $\min_x(\max(x, 2^{j-2}-x)) = 2^{j-3}$ palindromů, tedy nejméně polovinu. Automat bude místo hlavy $h_{i_{\max}}$ používat jeden z konců pásky.

Induktivně dostaneme, že potom automat s jednou páskou musí zvládnout rozpoznat konkatenaci $2^{j-2-(s-1)} = 2^{j-s-1}$ palindromů. Protože však $j > s$, musí automat s jednou páskou rozpoznávat alespoň $2^0 = 1$ palindromů. Je však dokázáno, že žádný jednopáskový deterministický limitovaný automat nedokáže palindrom rozpoznat.

### 4.2.3 Nedeterministické 0-limitované automaty

Nedeterministické vícepáskové 0-limitované automaty se zdají být silnější, než jejich deterministická varianta. Například každý z jazyků $L_{i\mathrm{pal}}$ z minulé podkapitoly jsme schopni přijímat už pomocí čtyř pásek. První páska slouží opět ke čtení vstupu, druhá k posunu první hlavy mezi požadovanými oddělovači stejně jako v deterministické verzi. Třetí a čtvrtá hlava obstarávají všechny virtuální oddělovače.

Automat nejprve nedeterministicky odhadne pozici prvního a druhého oddělovače (třetí a čtvrtou hlavou, přitom je jednoduché zajistit aby pozice čtvrté hlavy byla větší než třetí hlavy). Dále se deterministicky zkontroluje, zda je pozice vymezená oddělovači palindromem. Pokud ne, zamítáme, pokud ano nastavíme hlavu $h_3$ na pozici hlavy $h_4$, tu potom opět nedeterministicky posuneme

o libovolný počet vpravo a opět kontrolujeme palindrom.

Tímto způsobem kontrolujeme palindromy a jejich počet (který je nezávislý na délce vstupu) si můžeme pamatovat ve stavu. Pokud tedy rozdělení na palindromy existuje, některá výpočetní větev jej nalezne.

### 4.2.4 1-limitované automaty a bezkontextové jazyky

**Věta 34**

*Pro každý bezkontextový jazyk $L$ existuje (nedeterministický) $2t$-1-$LA$ $A$ přijímající $L$.*

*Důkaz*

Díky větě 20 lze libovolný bezkontextový jazyk zapsat jako $L = h(D_{j,k} \cap R)$. Pokusíme se tedy tuto konstrukci simulovat automatem $A$. Ten nejprve na druhou pásku nedeterministicky vygeneruje správné uzávorkování podle $D_{j,k}$, přičemž paralelně kontroluje, zda generovaný řetězec patří do zvoleného regulárního jazyka $R$ (k tomu nám postačí stav automatu). Ve druhé fázi stačí zkontrolovat, jestli je vstup obrazem vygenerovaného řetězce podle homomorfizmu $h$. Všimněme si, že pokud je $L$ bezkontextový, tak musí existovat alespoň jeden takový homomorfismus, jehož všechny obrazy mají délku jedna. To znamená, že prostor na druhé pásce musí stačit pro vygenerování tohoto řetězce.

Protože je $R$ regulární jazyk, musí existovat deterministický konečný automat $A_R$ s množinou stavů $Q_R$ obsahující počáteční stav $q_0$ a koncový stav $q_{\text{ACC}}$ a přechodovou funkcí $\delta_R$. Pro přehledný popis konstruovaného automatu také použijeme funkci $r$, která je však ve skutečném automatu realizována přímým dosazením obrazů $r$ do šablon přechodové funkce.

$$r((q_D, q_R), (i, b), d) = \begin{cases} ((q_D, q'_R), (i, b), d) & \delta_R(q_R, b) = q'_R \\ (q_{\text{REJ}}, (i, b), d) & \text{jinak} \end{cases}$$

Jak je vidět, stav automatu sestává z dvojice stavů, znázorňující paralelní vykonávání konečného automatu. Označme také pro $D_{j,k}$ množinu pravých závorek, levých závorek a neutrálních symbolů:

$$B_{\text{right}} = \{\ )_1\ , \ldots,\ )_j\ \}$$
$$B_{\text{left}} = \{\ (_1\ , \ldots,\ (_j\ \}$$
$$B_{\text{neutral}} = \{1, \ldots, k\}$$

Pro každé prázdné políčko automat vybere následující závorku. Buď levou závorku, neutrální znak, nebo pravou závorku. V případě pravé závorky však musí nejprve zkontrolovat, zda je tímto uzavřená část řetězce správně uzávorkovaná, tedy že řetězec patří do jazyka generovaného gramatikou:

$$S \to (_x S \qquad x \in \{1, \ldots, j\}$$
$$S \to DS$$
$$S \to D$$

(správné uzávorkování)

$$
\begin{aligned}
&D \to ({}_x D)_x \qquad && x \in \{1, \ldots, j\} \\
&D \to DD \\
&D \to c && c \in \{1, \ldots, k\} \\
&D \to \epsilon
\end{aligned}
$$

V každém kroku přidání dalšího symbolu buď přidáme další otevírací závorku (či neutrální symbol), nebo přidáme uzavírací závorku, v tom případě však zkontrolujeme, že byl zvolen správný typ a že neobsahuje řetězec méně otevíracích závorek. Snadno vidíme, že výsledný řetězec musí patřit do výše uvedeného jazyka, protože všechny prefixy tuto vlastnost splňují také.

### Generování řetězce Dyckova jazyka

1. $\delta(q_0, (i, \square)) = \{(g, q_0), (\bar{i}, \square), (-1, 0))\}$
2. $\delta((g, q_R), (\triangleright, i)) = \{(q, q_R), (\triangleright, i), (0, 1)\} \qquad i \notin \{\square, \triangleleft\}$
3. $\delta((g, q_R), (\triangleright, \square)) = \{r((g, q_R), (\triangleright, x), (0, 1)) \mid x \in B_{\text{left}} \cup B_{\text{neutral}}\} \cup$
   $\cup \ \{r((v_r, q_R), (\triangleright, x), (0, -1)) \mid x \in B_{\text{right}}\} \qquad q_R \in Q_R$

Pokud vybereme pravou závorku, je potřeba provést samotnou kontrolu správnosti zvoleného typu závorky. K tomu použijeme pozice první hlavy jakožto zásobníku. Na první pásku není třeba zapisovat, protože, jak již bylo uvedeno, výraz mezi nově přidanou, kontrolovanou závorkou a hledanou odpovídající závorkou musí být správně uzávorkovaný. Není tedy třeba si pamatovat typ procházených závorek.

Při průchodu uzavírací závorkou se hlava první pásky posune doprava a při průchodu otevírací doleva. Přes hledanou otevírací závorku projde, dostane-li se první hlava na pozici 0. Poté stačí porovnat typ závorky se zapamatovanou závorkou. Formálně se (i když to nepotřebujeme) po každém průchodu políčka jeho obsah přepíše, pokud není z množiny symbolů, které se již nepřepisují, proto znakem $\bar{i}$ naznačíme, že znak $c \in \Gamma_0$ se přepíše na $\bar{c} \in \Gamma_1$, dále se ale nemůže přepisovat, tedy $\bar{c} = \bar{\bar{c}}$.

Význam pravidla 2 si vysvětlíme později. Prozatím nám postačí, že pokud je symbol na druhé pásce již přepsán, prostě jej přeskočíme.

### Kontrola správnosti výběru pravé závorky

4. $\delta((v_r, q_R), (i, r_2)) = \{((v_r, q_R), (\bar{i}, r_2), (1, -1))\} \qquad i \in \{\triangleleft\} \cup \Gamma, r, r_2 \in B_{\text{right}}$
5. $\delta((v_r, q_R), (i, l)) = \{((v_r, q_R), (\bar{i}, l), (-1, -1))\} \qquad i \in \{\triangleleft\} \cup \Gamma, l \in B_{\text{left}}$
6. $\delta((v_r, q_R), (i, n)) = \{((v_r, q_R), (i, n), (0, -1))\} \qquad i \in \{\triangleright, \triangleleft\} \cup \Gamma, n \in B_{\text{neutral}}$
7. $\delta((v_r, q_R), (i, \triangleright)) = \{q_{\text{REJ}}, (i, \triangleright), (0, 0)\} \qquad i \neq \triangleright$
8. $\delta((v_r, q_R), (\triangleright, i)) = \{((v_{2r}, q_R), (\triangleright, i), (0, 1))\}$
9. $\delta((v_{2r}, q_R), (\triangleright, i)) = \{((g, q_R), (\triangleright, i), (0, 1))\} \qquad \exists x \colon i = ({}_x \ \wedge \ r =)_x$
10. $\delta((v_{2r}, q_R), (\triangleright, i)) = \{(q_{\text{REJ}}, (\triangleright, i), (0, 0))\} \qquad \nexists x \colon i = ({}_x \ \wedge \ r =)_x$

Pokud neexistuje žádné další prázdné pole a konečný automat zároveň přijímá vygenerovaný řetězec, je možné se posunout do druhé fáze kontroly uzávorkování. Tuto fázi popisují pravidla 4-10.

Ve stavu $v_r$ si pamatujeme pravou závorku $r$ a hledáme odpovídající levou závorku. V pravidlech 4-6 procházíme jednotlivé symboly a podle typu (levá závorka, neutrální symbol, pravá závorka) posunujeme první hlavu. Díky tomu, že v pravidle 6 může $i$ být $\triangleright$, dokáže automat přeskočit neutrální symboly až na případnou levou závorku.

Pokud přečteme celý vygenerovaný řetězec, ale nenalezli jsme místo, kde by měla být hledaná závorka, podle pravidla 7 zamítáme. Jinak by měla být hledaná závorka na pozici, kterou druhá hlava právě přešla. Tam může být i neutrální symbol, to však nevadí, protože automat je nedeterministický a existuje výpočetní větev, kdy automat přeskočí všechny neutrální symboly až na hledanou závorku.

Poté se ve stavu $q_{2r}$ podle pravidla 9 a 10 buď zamítá, nebo přecházíme do stavu $q$ abychom vygenerovali a zkontrolovali další část řetězce.

**Kontrola správnosti celého vygenerovaného řetězce**

Víme, že každá uzavírací závorka v doposud vygenerovaném řetězci je vytvořena správně, celý řetězec je tedy správným uzávorkováním ve smyslu $D_{j,k}$ právě tehdy, má-li stejný počet otevíracích a zavíracích závorek.To si ověříme pravidly 11-15.

11. $\delta((g, q_R), (\triangleright, \triangleleft)) = \{(c, (\triangleright, \triangleleft), (0, -1))\} \qquad q_R = q_{\text{ACC}}$
12. $\delta(c, (i, r)) = \{(c, (i, r), (1, -1))\} \qquad r \in B_{\text{right}}$
13. $\delta(c, (i, l)) = \{(c, (i, l), (-1, -1))\} \qquad i \neq \triangleright, l \in B_{\text{left}}$
14. $\delta(c, (i, n)) = \{(c, (i, n), (0, -1))\} \qquad n \in B_{\text{neutral}}$
15. $\delta(c, (\triangleright, l)) = \{(q_{\text{REJ}}, (\triangleright, l), (0, 0))\} \qquad l \in B_{\text{left}}$

Podle pravidla 11 tuto kontrolu započneme v případě, že vygenerovaný řetězec patří do zvoleného regulárního jazyka $R$. Zde se dozvídáme důvod pravidla 2.

Poté už nebudeme potřebovat mít stav složený z dvojic, protože nepotřebujeme simulovat paralelní vykonávání konečného automatu. V pravidlech 12-15 používáme stejný trik jako v předchozích pravidlech a počítáme si „zanoření“ ve struktuře závorek.

**Kontrola homomorfismu**

Při úspěšné kontrole stačí porovnat, zda je vstup homomorfní s nedeterministicky sestaveným průnikem regulárního jazyka a Dyckova jazyka. Toto porovnání

proběhne v pravidlech 17-19.

16. $\delta(c, (\triangleright, \triangleright)) = \{(q_h, (\triangleright, \triangleright), (1, 1))\}$
17. $\delta(q_h, (i, i_2)) = \{(q_h, (i, i_2), (1, 1))\} \qquad i = h(i_2)$
18. $\delta(q_h, (i, i_2)) = \{(q_{\text{REJ}}, (i, i_2), (0, 0))\} \qquad i \neq h(i_2)$
19. $\delta(q_h, (\triangleleft, \triangleleft)) = \{q_{\text{ACC}}, (\triangleleft, \triangleleft), (0, 0))\}$

□

Označme třídu jazyků pro dvoupáskové jedna-limitované automaty $L_{\text{2t-1-LA}} = \{L \mid \exists \text{2t-1-LA } A\colon L = L(A)\}$.

**Věta 35**
$CFL \subset L_{2t-1LA}$.

*Důkaz*
Pro každý bezkontextový jazyk existuje 2t-1-LA. Navíc z příkladu 23 víme, že existuje 2t-1-dLA rozpoznávající kontextový jazyk. Nedeterministická verze automatu (2t-1-LA) tedy musí také rozpoznávat tento kontextový jazyk. □

# 5 Otevřené problémy

Během této práce vzniklo mnoho nových otevřených problémů týkajících se nového modelu vícepáskových limitovaných automatů. Shrňme si nyní ty nejdůležitější.

Hned při úvodních příkladech jsme narazili na nepraktičnost implementace synchronních limitovaných automatů, proto jsme definovali asynchronní automaty. Z následných úvah nám bylo jasné, že třída jazyků rozpoznávaných synchronními automaty je podmnožinou jazyků rozpoznávaných asynchronními automaty. Podle Otevřeného problému 27 však nevíme, jestli je tato podmnožina vlastní.

Dále jsme vykonstruovali hierarchii jazyků a pro ně odpovídající 0-limitované vícepáskové stroje.

Otevřený problém 36
Pokud použijeme konstrukci oddělovačů, nelze dané jazyky $L_{i\text{pal}}$ rozpoznávat automaty s méně než $i$ páskami. Nevíme však, zda toto tvrzení platí obecně. Pokud ano, pak jsou $s$t-0-dLA silnější než $(s-1)$t-0-dLA.

Pokud by se prokázalo, že nelze jazyky $L_{i\text{pal}}$ rozpoznávat automaty s méně než $i$ páskami, budeme znát odpověď i na další otázku, a sice jestli jsou nedeterministické 0-limitované automaty silnější než deterministické.

Otevřený problém 37

Není známo, jestli dokáží $s$-$k$-LA rozpoznávat více jazyků než $s$-$k$-dLA.

Otevřený problém 38

Neznáme přesnou třídu jazyků žádného vícepáskového limitovaného automatu.

## 5.1 Prvočíselnost

Zřejmým způsobem dokážeme rozpoznat pomocí 3t-0-dLA, jestli je číslo $p$ reprezentované řetězcem $a^p$ prvočíslem. Pomocí hlavy $h_2$ a $h_3$ postupně ukládáme čísla $2, 3, \ldots$ tak, že hlava $h_3$ je na pozici daného čísla a $h_2$ je na začátku pásky. Potom známým trikem dokážeme o vzájemnou vzdálenost $h_2$ a $h_3$ posouvat hlavu $h_1$ a tak zkontrolovat, jestli je délka vstupu tímto číslem dělitelná. Na konci každého posunu se dokáží hlavy vrátit zpět do původní pozice. Potom můžeme postupně kontrolovat dělitelnost ostatními čísly.

Problém se stává zajímavějším pro 2t-1-dLA.

Otevřený problém 39

Nevíme, jestli dokáže některý 2t-$k$-dLA rozpoznat prvočíselnost délky vstupu.

Doplněk tohoto jazyka díky nedeterminismu rozpoznávat dokážeme a stačí nám k tomu 2t-1-dLA. Stačí nedeterministicky na druhé pásce odhadnout číslo, kterým lze délku vstupu vydělit beze zbytku. Protože automat je jedna-limitovaný, můžeme si toto číslo na pásku poznačit a ověřit, že je délka vstupu tímto číslem skutečně dělitelná.

Mohlo by se zdát, že se podobný postup dá použít i pro rozpoznání prvočíselnosti. Postupně bychom si na pásku značili všechna čísla, kterými se snažíme délku vstupu dělit. Problém je v tom, že pokud si pomocí jedna-limitovaného automatu označíme určitý počet políček na druhé pásce, když se budeme opakovaně snažit na tomto konci zastavit, vždy musíme pro detekci konce označení tento konec překročit. Tím si přepíšeme následující políčko a tak dále.

Tento postup tedy rozhodně nedovoluje 2t-1-dLA rozpoznávat prvočísla. Navíc se zdá, že to nelze ani pokud zvýšíme počet přepisů. Vždy se najde dostatečně dlouhý vstup tak, že budeme kvůli opakované detekci konce aktuálně poznačených polí nuceni tento konec překročit.

# Závěr

V této práci jsme si představili limitované automaty a odpovídající třídy jazyků. Shrnuli jsme metody vzájemné simulace některých modelů a tím i důkaz, že automaty rozpoznávají právě tyto třídy.

Ve druhé části práce jsme definovali nové modely – takzvané vícepáskové limitované automaty. Tyto automaty se ukázaly jako zajímavé verze limitovaných automatů, neboť rozšiřují třídy rozpoznávaných jazyků a jejich vztah s jednopáskovými verzemi není tak jednoduchý, jak by se na první pohled mohlo zdát.

Zajímavým jazykem, který nás v této části práce doprovázel, je jazyk všech palindromů, či konkatenace více palindromů. Ten pomáhal demonstrovat sílu mnoha konkrétních modelů vícepáskových automatů a ačkoliv se nepovedl najít důkaz, zdá se, že třída jazyků s konkatenací exponenciálního počtu palindromů definuje nekonečnou hierarchii vícepáskových automatů s různou výpočetní silou.

Celkově druhá část práce představuje velké množství otevřených problémů a neověřených domněnek. Jedná se o zajímavé a pravděpodobně nesnadné problémy, na jejichž řešení bych chtěl v budoucnu nadále pracovat.

# Conclusions

In this thesis, we have presented limited automata and their corresponding classes of languages. We have sumarized the methods of mutual simulation of some of these models and proof that these automata recognize their corresponding classes of languages.

In the second part of the thesis, we defined new models – so-called multi-tape limited automata. These machines turned out to be interesting versions of limited automata, as they expand the recognized classes and their relationship with the single-tape automata is not that simple, as it might seem at first glance.

The interesting language that accompanied us in this part of the thesis is the language of all palindromes or the concatenation of several palindromes. The latter helped demonstrate the power of many specific models of multi-tape automata, and although we failed to find proof, it appears that the class of languages of concatenations of an exponential number of palindromes defines an infinite hierarchy of automata.

Overall, the second part of the work presents a large number of open problems and unverified assumptions. These are interesting and probably difficult problems, the solution of which I would like to continue to work on in the future.

# A Obsah elektronických dat

Elektronická data odevzdaná v systému katedry mají následující strukturu:

**`limited-automata.pdf`**
: Samostatný text práce ve formátu PDF.

**`text/`**
: Adresář s textem práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech souborů potřebných pro bezproblémové vytvoření PDF dokumentu textu, tj. zdrojový text textu.

**`README.md`**
: Textový soubor s informacemi o práci, struktuře adresáře a použitých materiálech včetně zdrojů.

# Literatura


[1] Hibbard, Thomas N. A generalization of context-free determinism. *Information and Control*. 1967, roč. 11, č. 1, s. 196–238. Dostupný také z: ⟨http://dx.doi.org/https://doi.org/10.1016/S0019-9958(67)90513-X⟩. ISSN 0019-9958.

[2] Pighizzini, Giovanni; Pisoni, Andrea. Limited automata and regalar languages. *International Journal of Foundations of Computer Science*. 2014, roč. 25, č. 07, s. 897–916. Dostupný také z: ⟨http://dx.doi.org/10.1142/S0129054114400140eprint: https://doi.org/10.1142/S0129054114400140⟩.

[3] Pighizzini, Giovanni; Pisoni, Andrea. Limited automata and context-free languages. *Fundamenta Informaticae*. 2015, roč. 136, č. 1-2, s. 157–176. Dostupný také z: ⟨http://dx.doi.org/10.3233/fi-2015-1148⟩.

[4] Kutrib, Martin; Pighizzini, Giovanni; Wendlandt, Matthias. Descriptional complexity of limited automata. *Information and Computation*. 2018, roč. 259, s. 259–276. DCFS 2015. Dostupný také z: ⟨http://dx.doi.org/https://doi.org/10.1016/j.ic.2017.09.005⟩. ISSN 0890-5401.

[5] Okhotin, Alexander. Non-erasing Variants of the Chomsky–Schützenberger Theorem. In. *Non-erasing Variants of the Chomsky–Schützenberger Theorem*. 2012. Dostupný také z: ⟨http://dx.doi.org/10.1007/978-3-642-31653-1_12⟩. ISBN 978-3-642-31652-4.

[6] Pighizzini, Giovanni. Limited Automata: Properties, Complexity and Variants. In. *Descriptional Complexity of Formal Systems*. 2019, s. 57–73. Dostupný také z: ⟨https://doi.org/10.1007/978-3-030-23247-4_4⟩.